VMime Book

A Developer's Guide To VMime

Vincent Richard
vincent@vmime.org

July 13, 2017

Contents

1 Introduction **5**

1.1 Overview 5

1.2 Features 5

1.3 Copyright and license 6

2 Building and Installing VMime **8**

2.1 Introduction 8

2.2 What you need 8

2.3 Obtaining source files 8

2.4 Compiling and installing 9

2.5 Customizing build 10

2.6 Build options 10

3 Getting Started **12**

3.1 Using VMime in your programs 12

3.2 If you can not (or do not want to) use `pkg-config` 13

3.3 Platform-dependent code 13

4 Basics **15**

4.1 Reference counting 15

4.1.1 Introduction 15

4.1.2 Instanciating reference-counted objects 16

4.1.3 Using smart pointers 16

4.2 Error handling . . . 18
4.3 Basic objects . . . 19
4.3.1 The `component` class . . . 19
4.3.2 Date and time . . . 20
4.3.3 Media type . . . 20
4.3.4 Mailbox and mailbox groups . . . 21
4.4 Message, body parts and header . . . 22
4.4.1 Introduction to MIME messages . . . 22
4.4.2 Header and header fields . . . 23
4.4.2.1 Standard header fields . . . 23
4.4.2.2 Parameterized fields . . . 24
4.5 Streams . . . 25
4.5.1 Streams and stream adapters . . . 25
4.5.2 Stream filters . . . 26
4.6 Content handlers . . . 26
4.6.1 Introduction . . . 26
4.6.2 Extracting data from content handlers . . . 26
4.6.3 Creating content handlers . . . 27
4.7 Character sets, charsets and conversions . . . 28
4.8 Non-ASCII text in header fields . . . 29
4.9 Encodings . . . 30
4.9.1 Introduction . . . 30
4.9.2 Using encoders . . . 30
4.9.3 Enumerating available encoders . . . 31
4.10 Progress listeners . . . 32

5 Parsing and Building Messages **33**

5.1 Parsing messages . . . 33
5.1.1 Introduction . . . 33

5.1.2 Using the `vmime::messageParser` object . . . 34
5.2 Building messages . . . 36
5.2.1 A simple message . . . 36
5.2.2 Adding an attachment . . . 38
5.2.3 HTML messages and embedded objects . . . 39
5.3 Working with attachments: the attachment helper . . . 40

6 Working with Messaging Services **42**

6.1 Introduction . . . 42
6.2 Working with sessions . . . 44
6.2.1 Setting properties . . . 44
6.2.2 Available properties . . . 45
6.2.3 Instanciating services . . . 46
6.3 User credentials and authenticators . . . 48
6.4 Using transport service . . . 50
6.5 Using store service . . . 52
6.5.1 Connecting to a store . . . 52
6.5.2 Opening a folder . . . 52
6.5.3 Fetching messages . . . 53
6.5.4 Extracting messages and parts . . . 55
6.5.5 Deleting messages . . . 55
6.5.6 Events . . . 56
6.6 Handling timeouts . . . 56
6.7 Secured connection using TLS/SSL . . . 59
6.7.1 Introduction . . . 59
6.7.2 Setting up a secured connection . . . 59
6.7.2.1 Connecting to a “secured” port . . . 59
6.7.2.2 Using STARTTLS . . . 60
6.7.3 Certificate verification . . . 60

6.7.3.1 How it works 60
6.7.3.2 Using the default certificate verifier 60
6.7.3.3 Writing your own certificate verifier 62
6.7.4 SSL/TLS Properties 63

Listings **67**

List of figures **68**

List of tables **69**

A The GNU General Public License **70**

Chapter 1

Introduction

1.1 Overview

VMime is a powerful C++ class library for working with MIME messages and Internet messaging services like IMAP, POP or SMTP.

With VMime you can parse, generate and modify messages, and also connect to store and transport services to receive or send messages over the Internet. The library offers all the features to build a complete mail client.

The main objectives of this library are:

- fully RFC-compliant implementation;
- object-oriented and modular design;
- very easy-to-use (intuitive design);
- well documented code;
- very high reliability;
- maximum portability.

1.2 Features

MIME features:

- Full support for RFC-2822 and multipart messages (RFC-1521)
- Aggregate documents (MHTML) and embedded objects (RFC-2557)
- Message Disposition Notification (RFC-3798)

- 8-bit MIME (RFC-2047)
- Encoded word extensions (RFC-2231)
- Attachments

Network features:

- Support for IMAP, POP3 and maildir stores
- Support for SMTP and sendmail transport methods
- Extraction of whole message or specific parts
- TLS/SSL security layer
- SASL authentication

1.3 Copyright and license

VMime library is Free Software and is licensed under the terms of the GNU General Public License[1] (GPL) version 3:

```
Copyright (C) 2002-2013 Vincent Richard

VMime library is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License as
published by the Free Software Foundation; either version 3 of
the License, or (at your option) any later version.

VMime is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

Linking this library statically or dynamically with other
modules is making a combined work based on this library.
Thus, the terms and conditions of the GNU General Public
License cover the whole combination.
```

[1] See Appendix A and http://www.gnu.org/copyleft/gpl.html

This document is released under the terms of the GNU Free Documentation License[2] (FDL):

Copyright (C) 2004-2013 Vincent Richard

Permission is granted to copy, distribute and/or modify
this document under the terms of the GNU Free Documentation
License, Version 1.2 or any later version published by the
Free Software Foundation; with no Invariant Sections, no
Front-Cover Texts, and no Back-Cover Texts.

[2]See http://www.gnu.org/copyleft/fdl.html

Chapter 2

Building and Installing VMime

2.1 Introduction

If no pre-build packages of VMime is available for your system, or if for some reason you want to compile it yourself from scratch, this section will guide you through the process.

2.2 What you need

To build VMime from the sources, you will need the following:

- a working C++ compiler with good STL implementation and also a good support for templates (for example, GNU GCC) ;
- CMake build system ;
- either ICU library or an usable `iconv()` implementation (see libiconv of GNU Project) ;
- the GNU SASL Library if you want SASL[1] support ;
- either the OpenSSL library or the GNU TLS Library if you want SSL and TLS[2] support ;
- the Boost C++ library if you are not using C++11 (or your compiler does not support it), for `shared_ptr<>`.

2.3 Obtaining source files

You can download a package containing the source files of the latest release of the VMime library from the VMime web site.

[1] Simple Authentication and Security Layer
[2] Transport Layer Security

You can also obtain the current development version from the Git repository, which is currently hosted at GitHub. It can be checked out through anonymous access with the following instruction:

```
git clone git://github.com/kisli/vmime
```

2.4 Compiling and installing

VMime relies on CMake for building. CMake is an open source, cross-platform build system. It will generate all build scripts required to compile VMime on your platform.

First, extract the tarball or checkout the VMime source code into a directory somewhere on your system, let's call it `/path/to/vmime-source`. Then, create a build directory, which will contain all intermediate build files and the final libraries, let's call it `/path/to/vmime-build`.

From the build directory, run `cmake` with the `-G` argument corresponding to your platform/choice. For example, if you are on a Unix-compatible platform (like GNU/Linux or MacOS) and want to use the `make` utility for building, type:

```
$ cd /path/to/vmime-build
$ cmake -G "Unix Makefiles" /path/to/vmime-source
```

CMake will perform some tests on your system to check for libs installed and some platform-specific includes, and create all files needed for compiling the project. Additionally, a `src/vmime/config.hpp` file with the parameters detected for your system will be created.

Next, you can start the compilation process:

```
$ cmake --build .
```

Please wait a few minutes while the compilation runs (you should have some time to have a coffee right now!). If you get errors during the compilation, be sure your system meet the requirements given at the beginning of the chapter. You can also try to get a newer version (from the Git repository, for example) or to get some help on VMime user forums.

If everything compiled successfully, you can install the library and the development files on your system:

```
# make install
```

NOTE: you must do that with superuser rights (root) if you chose to install the library into the default location (ie: /usr/lib and /usr/include).

Now, you are done! You can jump to the next chapter to know how to use VMime in your program...

2.5 Customizing build

You should not modify the `config.hpp` file directly. Instead, you should run `cmake` again, and specify your build options on the command line. For example, to force using OpenSSL library instead of GnuTLS for TLS support, type:

```
$ cmake -G "Unix Makefiles" -DVMIME_TLS_SUPPORT_LIB=openssl
```

If you want to enable or disable some features in VMime, you can obtain some help by typing `cmake -L`. The defaults should be OK though. For a complate list of build options, you can also refer to section 2.6, page 10. For more information about using CMake, go to the CMake web site.

NOTE: Delete the `CMakeCache.txt` file if you changed configuration or if something changed on your system, as CMake may cache some values to speed things up.

You can also use another build backend, like Ninja[3], if you have it on your system:

```
$ cd /path/to/vmime-build
$ cmake -G Ninja /path/to/vmime-source
$ ninja
# ninja install
```

To install VMime in a directory different from the default directory (`/usr` on GNU/Linux systems), set the `CMAKE_INSTALL_PREFIX` option:

```
$ cmake -DCMAKE_INSTALL_PREFIX=/opt/ ...
```

2.6 Build options

Some options can be given to CMake to control the build:

[3] https://ninja-build.org/

Option name	Description
VMIME_BUILD_SHARED_LIBRARY	Set to ON to build a shared version (.so) of the library (default is ON).
VMIME_BUILD_STATIC_LIBRARY	Set to ON to build a static version (.a) of the library (default is ON).
VMIME_BUILD_TESTS	Set to ON to build unit tests (default is OFF).
VMIME_TLS_SUPPORT_LIB	Set to either "openssl" or "gnutls" to force using either OpenSSL or GNU TLS for SSL/TLS support (default depends on which libraries are available on your system).
VMIME_CHARSETCONV_LIB	Set to either "iconv", "icu" or "win" (Windows only) to force using iconv, ICU or Windows built-in API for converting between charsets (default value depends on which libraries are available on your system).
CMAKE_BUILD_TYPE	Set the build type: either "Release" or "Debug". In Debug build, optimizations are disabled and debugging information are enabled.

Table 2.1: CMake build options

Chapter 3

Getting Started

3.1 Using VMime in your programs

First, make sure you have successfully compiled and installed VMime using the instructions described in Chapter 1.3. To use VMime in your program, you simply have to include VMime headers:

```
#include <vmime/vmime.hpp>
```

NOTE: for versions older than 0.6.1, include <vmime/vmime>.

As of version 0.6.1, VMime uses `pkg-config` to simplify compiling and linking with VMime. The `pkg-config` utility is used to detect the appropriate compiler and linker flags needed for a library.

You can simply build your program with:

```
$ g++ `pkg-config --cflags --libs vmime` -static -o myprog myprog.cpp
```

to use the static version, or with:

```
$ g++ `pkg-config --cflags --libs vmime` -o myprog myprog.cpp
```

to use the shared version.

NOTE: it is highly recommended that you link your program against the shared version of the library.

All VMime classes and global functions are defined in the namespace `vmime`, so prefix explicitely all your declarations which use VMime with `vmime::`, or import the `vmime` namespace into the global namespace with the C++ keywork `using` (not recommended, though).

3.2 If you can not (or do not want to) use `pkg-config`

Linking with the shared library (.so): compile your program with the `-lvmime` flag. You can use the -L path flag if the library file is not in a standard path (ie. not in /usr/lib or /usr/local/lib).

NOTE: if you want to link your program with the shared version of VMime library, make sure the library has been compiled using CMake build system (`make`, then `make install`). When you compile with SCons, only the static library is built and installed.

Linking with the static library (.a): follow the same procedure as for shared linking and append the flag -static to force static linking. Although static linking is possible, you are encouraged to use the shared (dynamic) version of the library.

3.3 Platform-dependent code

While the most part of VMime code is pure ANSI C++, there are some features that are platform-specific: file management (opening/reading/writing files), network code (socket, DNS resolution) and time management. All the non-portable stuff is done by a bridge object called a platform handler (see `vmime::platform`).

If your platform is POSIX-compatible (eg. GNU/Linux, *BSD) or is Windows, then you are lucky: VMime has built-in support for these platforms. If not, don't worry, the sources of the built-in platform handlers are very well documented, so writing you own should not be very difficult.

If your VMime version is <= 0.9.1, you should tell VMime which platform handler you want to use at the beginning of your program (before using *any* VMime object, or calling *any* VMime global function).

So, if your platform is POSIX, your program should look like this:

```
#include <vmime/vmime.hpp>
#include <vmime/platforms/posix/posixHandler.hpp>

int main()
{
    vmime::platform::
        setHandler <vmime::platforms::posix::posixHandler>();

    // Now, you can use VMime
    // ...do what you want, it's your program...
}
```

Listing 3.1: Initializing VMime and the platform handler

For using VMime on Windows, include `vmime/platforms/windows/windowsHandler.hpp` and use the following line to initialize the platform handler:

```
vmime::platform::
    setHandler <vmime::platforms::windows::windowsHandler>();
```

NOTE: since version 0.9.2, this is not needed any more: the platform handler is installed automatically using the platform detected during the build configuration.

NOTE: since version 0.8.1, `vmime::platformDependant` was renamed to `vmime::platform`. The old name has been kept for compatibility but it is recommended that you update your code, if needed.

Chapter 4

Basics

4.1 Reference counting

4.1.1 Introduction

Since version 0.7.2cvs, VMime use smart pointers to simplify memory management. Smart pointers rely on RAII[1] so that we do not need to bother with deleting an object (freeing memory) when it is not used anymore.

There are two possibilities for owning a reference to an object. We can own a strong reference to an object: as long as we keep this reference, the object is not destroyed. Or we can own a weak reference to the object: the object can be destroyed if nobody owns a strong reference to it, in which case the weak reference becomes invalid.

An object is destroyed as soon as the last strong reference to it is released. At the same tine, all weak references (if any) are automatically set to point to `NULL`.

In VMime, these two types of references are known as `vmime::shared_ptr` and `vmime::weak_ptr`, respectively.

NOTE: since November 2013, we switched from an old, intrusive implementation of smart pointers to a more standard one: either Boost `shared_ptr<>` implementation or standard C++ one if we are compiling in C++11. Here are the changes:

`vmime::ref <>` is replaced with `vmime::shared_ptr <>`

`vmime::weak_ref <>` is replaced with `vmime::weak_ptr <>`

`vmime::create <>` is replaced with `vmime::make_shared <>`

[1] Ressource Allocation is Initialisation

4.1.2 Instanciating reference-counted objects

In VMime, all objects that support reference counting inherit from the `vmime::object` class, which is responsible for incrementing/decrementing the counter and managing the object's life cycle. If you want to create a smart pointer to a new object instance, you should use the function `vmime::make_shared` instead of the `new` operator.

```
class myObject : public vmime::object
{
public:

    myObject(const vmime::string& name)
        : m_name(name)
    {
    }

    void sayHello()
    {
        std::cout << "Hello " << m_name << std::endl;
    }

private:

    vmime::string m_name;
};

int main()
{
    vmime::shared_ptr <myObject> obj =
        vmime::make_shared <myObject>("world");

    obj->sayHello();

    return 0;

} // Here, 'obj' gets automatically destroyed
```

Listing 4.1: Smarts pointers and creating objects

4.1.3 Using smart pointers

Smart pointers are copiable, assignable and comparable. You can use them like you would use normal ("raw") C++ pointers (eg. you can write !ptr, ptr != NULL, ptr->method(), *ptr...).

Type safety is also guaranteed, and you can type cast smart pointers using the `static_cast()`, `dynamic_cast()` and `const_cast()` equivalents on `vmime::shared_ptr` and `vmime::weak_ptr` objects:

```
class myBase : public vmime::object { }
class myObject : public myBase { }

vmime::shared_ptr <myObject> obj = vmime::make_shared <myObject>();

// Implicit downcast
vmime::shared_ptr <myBase> base = obj;

// Explicit upcast
vmime::shared_ptr <myObject> obj2 = vmime::dynamicCast <myObject>(base);
```

Listing 4.2: Casting smart pointers

Weak references are used to resolve reference cycles (an object which refers directly or indirectly to itself). The following example illustrates a typical problem of reference counting:

```
class parent : public vmime::object
{
public:

   void createChild(vmime::shared_ptr <child> c)
   {
      m_child = c;
   }

private:

   vmime::shared_ptr <child> m_child;
};

class child : public vmime::object
{
public:

   child(vmime::shared_ptr <parent> p)
      : m_parent(p)
   {
   }

private:
```

```
    vmime::shared_ptr <parent> m_parent;
};

int main()
{
    vmime::shared_ptr <parent> p = vmime::make_shared <parent>();
    vmime::shared_ptr <child> c = vmime::make_shared <child>();

    p->setChild(c);
}
```

In this example, neither `p` nor `c` will be deleted when exiting `main()`. That's because `p` indirectly points to itself *via* `c`, and *vice versa*. The solution is to use a weak reference to the parent:

```
vmime::weak_ptr <parent> m_parent;
```

The decision to make the parent or the child a weak reference is purely semantic, and it depends on the context and the relationships between the objects. Note that when the parent is deleted, the `m_parent` member of the child points to `NULL`.

More information about reference counting can be found on Wikipedia[2].

4.2 Error handling

In VMime, error handling is exclusively based on exceptions, there is no error codes, or things like that.

VMime code may throw exceptions in many different situations: an unexpected error occured, an operation is not supported, etc. You should catch them if you want to report failures to the user. This is also useful when debugging your program.

VMime exceptions support chaining: an exception can be encapsulated into another exception to hide implementation details. The function `exception::other()` returns the next exception in the chain, or `NULL`.

Following is an example code for catching VMime exceptions and writing error messages to the console:

```
std::ostream& operator<<(std::ostream& os, const vmime::exception& e)
{
    os << "* vmime::exceptions::" << e.name() << std::endl;
    os << "      what = " << e.what() << std::endl;
```

[2] http://en.wikipedia.org/wiki/Reference_counting

```
    // Recursively print all encapsuled exceptions
    if (e.other() != NULL)
        os << *e.other();

    return os;
}

...

try
{
    // ...some call to VMime...
}
catch (vmime::exception& e)
{
    std::cerr << e;             // VMime exception
}
catch (std::exception& e)
{
    std::cerr << e.what();   // standard exception
}
```

Listing 4.3: Catching VMime exceptions

Read the source of EXAMPLE6 if yo want to see a more complete example of using VMime exceptions (such as getting more detailed information by using specialized classes of `vmime::exception`).

4.3 Basic objects

4.3.1 The `component` class

In VMime, all the components of a message inherit from the same class `component`. This includes the message itself (classes `message` and `bodyPart`), the header, the header fields and the value of each header field, the body and all the parts in the message.

The class component provide a common interface for parsing or generating all these components (methods `parse()` and `generate()`). It also provides additional functions to get some information about the parsing process or the structure (methods `getParsedOffset()`, `getParsedLength()` and `getChildComponents()`).

VMime also provides a set of classes corresponding to the basic types found in a message; for example a mailbox, a mailbox list, date/time information, media type, etc. They all inherit from `component` too.

4.3.2 Date and time

Date and time are used in several places in VMime, particularly in header fields (Date, Received, ...). VMime fully supports RFC-2822's date and time specification. The object `vmime::datetime` is used to manipulate date and time information, and to parse/generate it from/to RFC-2822 format.

The following code snippet show various manners of using the `vmime::datetime` object:

```
// Creating from string in RFC-2822 format
vmime::datetime d1("Sat, 08 Oct 2005 14:07:52 +0200");

// Creating from components
vmime::datetime d2(
    /* date */ 2005, vmime::datetime::OCTOBER, 8,
    /* time */ 14, 7, 52,
    /* zone */ vmime::datetime::GMT2);

// Getting day of week
const int dow = d2.getWeekDay();   // 'dow' should be datetime::SATURDAY
```

Listing 4.4: Using `vmime::datetime` object

4.3.3 Media type

In MIME, the nature of the data contained in parts is identified using a media type. A general type (eg. *image*) and a sub-type (eg. *jpeg*) are put together to form a media type (eg. *image/jpeg*). This is also called the MIME type.

There are a lot of media types officially registered, and vendor-specific types are possible (they start with "x-", eg. *application/x-zip-compressed*).

In VMime, the object `vmime::mediaType` represents a media type. There are also some constants for top-level types and sub-types in the `vmime::mediaTypes` namespace. For example, you can instanciate a new media type with:

```
vmime::mediaType theType(
    /* top-level type */ vmime::mediaTypes::IMAGE,
    /* sub-type */       vmime::mediaTypes::IMAGE_JPEG);

// theType.getType() is "image"
// theType.getSubType() is "jpeg"
// theType.generate() returns "image/jpeg"
```

For more information about media types, see RFC-2046[3].

4.3.4 Mailbox and mailbox groups

VMime provides several objects for working with mailboxes and addresses.

The `vmime::address` class is an abstract type for representing an address: it can be either a mailbox (type `vmime::mailbox`) or a mailbox group (type `vmime::mailboxGroup`). A mailbox is composed of an email address (mandatory) and possibly a name. A mailbox group is simply a named list of mailboxes (see Figure 4.1).

```
vmime::shared_ptr <vmime::mailbox> mbox1 = vmime::make_shared <vmime::mailbox>
    (/* name */ vmime::text("John Doe"), /* email */ "john.doe@acme.com");
vmime::shared_ptr <vmime::mailbox> mbox2 = vmime::make_shared <vmime::mailbox>
    (/* no name, email only */ "bill@acme.com");

vmime::shared_ptr <vmime::mailboxGroup> grp = vmime::make_shared <vmime::mailboxGroup>();
grp->appendMailbox(mbox1);
grp->appendMailbox(mbox2);
```

Listing 4.5: Using mailboxes and mailbox groups

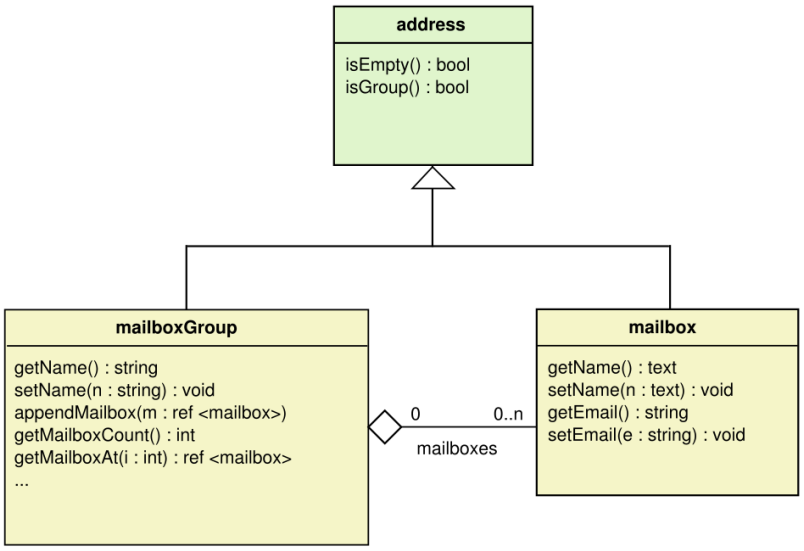

Figure 4.1: Diagram for address-related classes

[3] http://www.faqs.org/rfcs/rfc2046.html

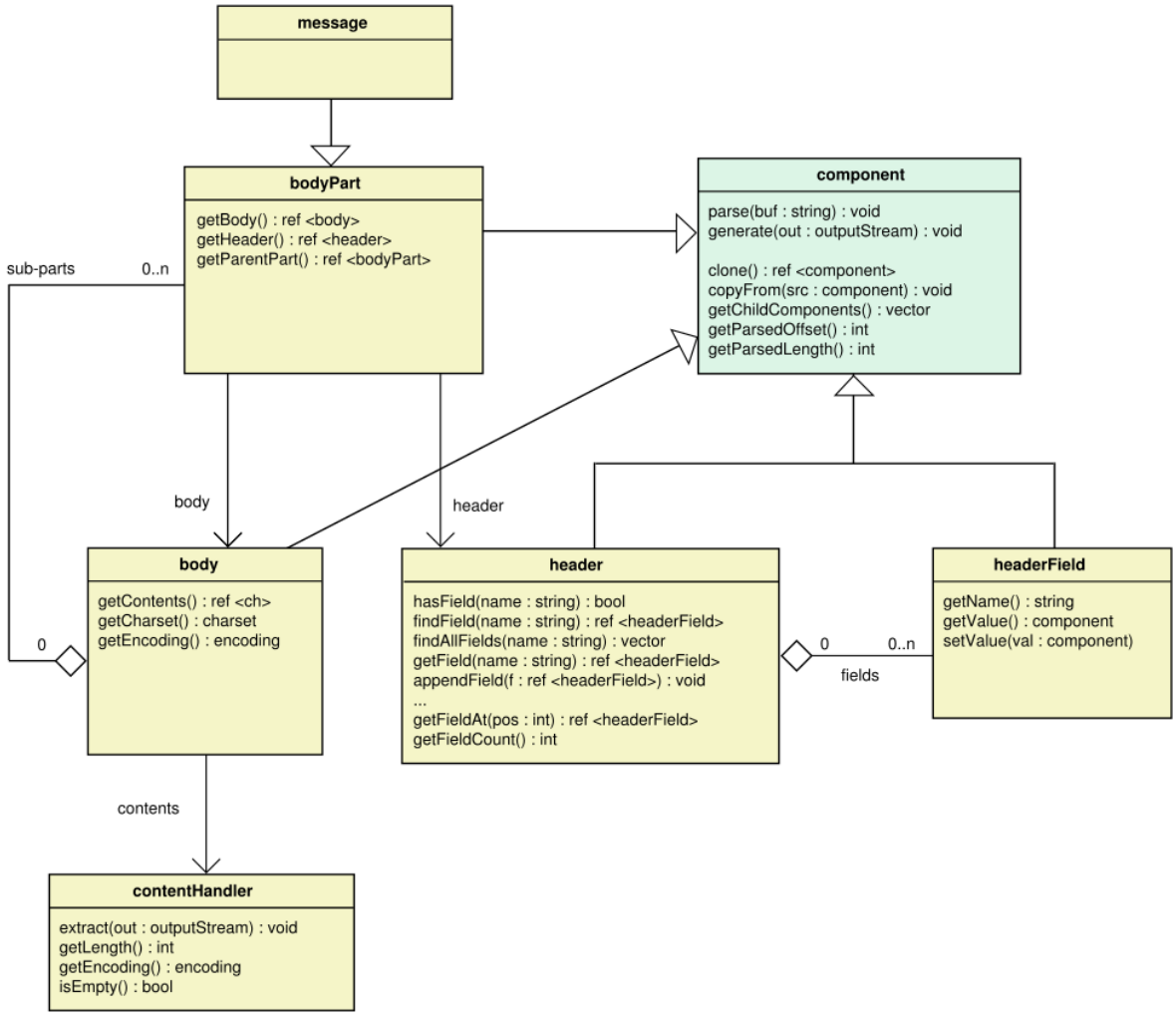

Figure 4.2: Overall structure of MIME messages

4.4 Message, body parts and header

4.4.1 Introduction to MIME messages

A MIME message is a recursive structure in which each part can contains one or more parts (or *entities*). Each part is composed of a header and a body (actual contents). Figure 4.2 shows how this model is implemented in VMime, and all classes that take part in it.

4.4.2 Header and header fields

4.4.2.1 Standard header fields

Header fields carry information about a message (or a part) and its contents. Each header field has a name and a value. All types that can be used as a field value inherit from the `headerFieldValue` class.

You cannot instanciate header fields directly using their constructor. Instead, you should use the `headerFieldFactory` object. This ensures the right field type and value type is used for the specified field name. For more information about how to use header fields and the factory, see section 5.2.1.

Some standard fields are officially registered and have their value type specified in a RFC. Table 4.4.2.1 lists all the fields registered by default in VMime and the value type they contains.

By default, all unregistered fields have a value of type `text`.

Field Name	**Value Type**
From	mailbox
To	addressList
Cc	addressList
Bcc	addressList
Sender	mailbox
Date	datetime
Received	relay
Subject	text
Reply-To	mailbox
Delivered-To	mailbox
Organization	text
Return-Path	path
Mime-Version	text
Content-Type	mediaType
Content-Transfer-Encoding	encoding
Content-Description	text
Content-Disposition	contentDisposition
Content-Id	messageId
Content-Location	text
Message-Id	messageId
In-Reply-To	messageIdSequence
References	messageIdSequence
Original-Message-Id	messageId
Disposition	disposition
Disposition-Notification-To	mailboxList

Table 4.1: Standard fields and their types

4.4.2.2 Parameterized fields

In addition to a value, some header fields can contain one or more *name=value* couples which are called *parameters*. For example, this is used in the *Content-Type* field to give more information about the content:

```
Content-Type: text/plain; charset="utf-8"
```

Fields that support parameters inherit from the `parameterizedHeaderField` class which provides methods to deal with these parameters: `appendParameter()`, `getParameterAt()`...

A parameter is identified by a name (eg. *charset*) and associated to a value of type `vmime::text`. Parameters provide helper functions to convert automatically from basic types to text, and *vice versa*. The following example illustrates it:

```
vmime::shared_ptr <vmime::parameterizedField> field =
    header->findField <vmime::parameterizedField>("X-Field-That-Contains-Parameters");

// Use setValue() to convert from a basic type to 'text'
vmime::shared_ptr <vmime::parameter> prm = field->getParameter("my-date-param");
prm->setValue(vmime::datetime::now());

// Use getValueAs() to convert from 'text' to a basic type
prm = field->getParameter("my-charset-param");
const vmime::charset ch = prm->getValueAs <vmime::charset>();
```

Listing 4.6: Getting and setting parameter value in fields

Some fields provide easy access to their standard parameters (see Table 4.4.2.2). This avoids finding the parameter and *dynamic-casting* its value to the right type. The following code illustrates how to use it:

```
vmime::shared_ptr <vmime::contentTypeField> field =
    header->getField <vmime::contentTypeField>(vmime::fields::CONTENT_TYPE);

// 1. First solution: the "hard" way
vmime::shared_ptr <vmime::parameter> prm = field->findParameter("charset");
const charset ch1 = prm->getValueAs <vmime::charset>();

// 2. Second solution: the simple way
const charset ch2 = field->getCharset();
```

NOTE: In both cases, an exception `no_such_parameter` can be thrown if the parameter does not exist, so be sure to catch it.

Field Name	Field Type	Parameters
Content-Type	contentTypeField	boundary, charset, report-type
Content-Disposition	contentDispositionField	creation-date, modification-date, read-date, filename, size

Table 4.2: Standard parameterized fields

4.5 Streams

4.5.1 Streams and stream adapters

Streams permit reading or writing data whatever the underlying system is: a file on a hard disk, a socket connected to a remote service...

There are two types of streams: input streams (from which you can read data) and output streams (in which you can write data). Some adapters are provided for compatibility and convenience, for example:

- `inputStreamAdapter` and `outputStreamAdapter`: allow to use standard C++ iostreams with VMime;
- `inputStreamStringAdapter` and `outputStreamStringAdapter`: use a `vmime::string` object to read/write data.

The following example shows two ways of writing the current date to the standard output, using stream adapters:

```
// Get current date and time
const vmime::datetime date = vmime::datetime::now();

// 1. Using outputStreamAdapter
vmime::utility::outputStreamAdapter out(std::cout);

std::cout << "Current date is: ";
date.generate(out);
std::cout << std::endl;

// 2. Using outputStreamStringAdapter
vmime::string dateStr;
vmime::utility::outputStreamStringAdapter outStr(dateStr);

date.generate(outStr);
```

```
std::cout << "Current date is: " << dateStr << std::endl;
```

Listing 4.7: Using stream adapters

4.5.2 Stream filters

Input and output streams can be filtered to perform inline conversions (for example, there is a filter to convert "\r\n" sequences to "\n"). They inherit from `vmime::utility::filteredInputStream` or `vmime::utility::filteredOutputStream` and are used like adapters (some filters also accept parameters; read the documentation).

The most useful filter in VMime (and probably the only one you will need) is the `charsetFilteredOutputStream`, which performs inline conversion of charsets. See 4.7 to know how to use it.

NOTE: After you have finished to use a filtered output stream, it is important to call `flush()` on it to flush the internal buffer. If `flush()` is not called, not all data may be written to the underlying stream.

4.6 Content handlers

4.6.1 Introduction

Content handlers are an abstraction for data sources. They are currently used when some data need to be stored for later use (eg. body part contents, attachment data, ...). Data can be stored encoded or unencoded (for more information about encodings, see 4.9).

4.6.2 Extracting data from content handlers

You can extract data in a content handler using the `extract()` method (which automatically decodes data if encoded) or `extractRaw()` (which extracts data without perfoming any decoding).

The following example shows how to extract the body text from a message, and writing it to the standard output with charset conversion:

```
// Suppose we already have a message
vmime::shared_ptr <vmime::message> msg;

// Obtains a reference to the body contents
vmime::shared_ptr <vmime::body> body = msg->getBody();
vmime::shared_ptr <vmime::contentHandler> cts = body->getContents();
```

```
vmime::utility::outputStreamAdapter out(std::cout);
cts->extract(out);
```

Listing 4.8: Using content handlers to extract body text from a message

NOTE: The body contents is extracted "as is". No charset conversion is performed. See 4.7 to know more about conversion between charsets.

4.6.3 Creating content handlers

When you are building a message, you may need to instanciate content handlers if you want to set the contents of a body part. The following code snippet shows how to set the body text of a part from a string:

```
vmime::shared_ptr <vmime::bodyPart> part;  // suppose we have a body part

// Create a new content handler from a string
vmime::shared_ptr <vmime::contentHandler> cth =
    vmime::make_shared <vmime::stringContentHandler>("Put body contents here");

// Set the contents
part->getBody()->setContents(cth);
```

Listing 4.9: Setting the contents of a body part

Content handlers are also used when creating attachments. The following example illustrates how to create an attachment from a file:

```
// Create a stream from a file
std::ifstream* fileStream = new std::ifstream();

fileStream->open("/home/vincent/paris.jpg", std::ios::binary);

if (!*fileStream)
    // handle error

vmime::shared_ptr <utility::stream> dataStream =
    vmime::make_shared <vmime::utility::inputStreamPointerAdapter>(fileStream);

    // NOTE: 'fileStream' will be automatically deleted
    // when 'dataStream' is deleted

// Create a new content handler
vmime::shared_ptr <contentHandler> data =
```

```
    vmime::make_shared <vmime::streamContentHandler>(dataStream, 0);

// Now create the attachment
ref <vmime::attachment> att = vmime::make_shared <vmime::defaultAttachment>
    (
        /* attachment data */ data,
        /* content type */    vmime::mediaType("image/jpeg"),
        /* description */     vmime::text("Holiday photo"),
        /* filename */        vmime::word("paris.jpg")
    );
```

Listing 4.10: Creating an attachment from a file

You will see later that the `vmime::fileAttachment` class already encapsulates all the mechanics to create an attachment from a file.

4.7 Character sets, charsets and conversions

Quoting from RFC-2278: *" The term 'charset' is used to refer to a method of converting a sequence of octets into a sequence of characters."*

With the `vmime::charset` object, VMime supports conversion between charsets using the *iconv* library, which is available on almost all existing platforms. See `vmime::charset` and `vmime::charsetConverter` in the class documentation to know more about charset conversion.

The following example shows how to convert data in one charset to another charset. The data is extracted from the body of a message and converted to UTF-8 charset:

```
vmime::shared_ptr <vmime::message> msg;  // we have a message

// Obtain the content handler first
vmime::shared_ptr <vmime::body> body = msg->getBody();
vmime::shared_ptr <const vmime::contentHandler> cth = body->getContents();

// Then, extract and convert the contents
vmime::utility::outputStreamAdapter out(std::cout);
vmime::utility::charsetFilteredOutputStream fout
    (/* source charset */ body->getCharset(),
     /* dest charset */   vmime::charset("utf-8"),
     /* dest stream */    out);

cth->extract(fout);
```

```
fout.flush();   // Very important!
```

Listing 4.11: Extracting and converting body contents to a specified charset

4.8 Non-ASCII text in header fields

MIME standard defines methods[4] for dealing with data which is not 7-bit only (ie. the ASCII character set), in particular in header fields. For example, the field "Subject:" use this data type.

VMime is fully compatible with RFC-2047 and provides two objects for manipulating 8-bit data: `vmime::text` and `vmime::word`. A word represents textual information encoded in a specified charset. A text is composed of one or more words.

RFC-2047 describes the process of encoding 8-bit data into a 7-bit form; basically, it relies on Base64 and Quoted-Printable encoding. Hopefully, all the encoding/decoding process is done internally by VMime, so creating text objects is fairly simple:

```
vmime::string inText = "Linux dans un tÃ©lÃ©phone mobile";
vmime::charset inCharset = "utf-8";

vmime::text outText;
outText.createFromString(inText, inCharset);

// 'outText' now contains 3 words:
//    . <us-ascii>   "Linux dans un "
//    . <utf-8>      "tÃ©lÃ©phone "
//    . <us-ascii>   "mobile"

vmime::shared_ptr <vmime::header> header = myMessage->getHeader();
header->Subject()->setValue(outText);
```

Listing 4.12: Creating `vmime::text objects`

In general, you will not need to decode RFC-2047-encoded data as the process is totally transparent in VMime. If you really have to, you can use the `vmime::text::decodeAndUnfold()` static method to create a text object from encoded data.

For example, say you have the following encoded data:

```
Linux dans un =?UTF-8?B?dMOpbMOpcGhvbmUgbW9iaWxl?=
```

[4] See RFC-2047: Message Header Extensions for Non-ASCII Text

You can simply decode it using the following code:

```
vmime::string inData =
    "Linux dans un =?UTF-8?B?dMOpbMOpcGhvbmUgbW9iaWxl?=";

vmime::text outText;
vmime::text::decodeAndUnfold(inData, &outText);
```

Listing 4.13: Decoding RFC-2047-encoded data

`vmime::text` also provides a function to convert all the words to another charset in a single call. The following example shows how to convert text stored in the Subject field of a message:

```
vmime::shared_ptr <vmime::message> msg;  // we have a message

vmime::text subject = msg->getHeader()->Subject()->getValue();

const vmime::string subjectText =
   subject.getConvertedText(vmime::charset("utf-8"));

// 'subjectText' now contains the subject in UTF-8 encoding
```

Listing 4.14: Converting data in a `vmime::text` to a specified charset

4.9 Encodings

4.9.1 Introduction

The MIME standard defines a certain number of encodings to allow data to be safely transmitted from one peer to another. VMime provides data encoding and decoding using the `vmime::utility::encoder::encoder` object.

You should not need to use encoders directly, as all encoding/decoding process is handled internally by the library, but it is good to know they exist and how they work.

4.9.2 Using encoders

You can create an instance of an encoder using the 'vmime::utility::encoder::encoderFactory' object, giving the encoding name (*base64*, *quoted-printable*, ...). The following example creates an instance of the Base64 encoder to encode some data:

```
vmime::shared_ptr <vmime::utility::encoder::encoder> enc =
    vmime::utility::encoder::encoderFactory::getInstance()->create("base64");
```

```
vmime::string inString("Some data to encode");
vmime::utility::inputStreamStringAdapter in(inString);

vmime::string outString;
vmime::utility::outputStreamStringAdapter out(outString);

enc->encode(in, out);

std::cout << "Encoded data is:" << outString << std::endl;
```

Listing 4.15: A simple example of using an encoder

4.9.3 Enumerating available encoders

The behaviour of the encoders can be configured using properties. However, not all encoders support properties. The following example[5] enumerates available encoders and the supported properties for each of them:

```
vmime::shared_ptr <vmime::utility::encoder::encoderFactory> ef =
    vmime::utility::encoder::encoderFactory::getInstance();

std::cout << "Available encoders:" << std::endl;

for (int i = 0 ; i < ef->getEncoderCount() ; ++i)
{
    // Output encoder name
    vmime::shared_ptr <const vmime::utility::encoder::encoderFactory::registeredEncoder>
        enc = ef->getEncoderAt(i);

    std::cout << "  * " << enc->getName() << std::endl;

    // Create an instance of the encoder to get its properties
    vmime::shared_ptr <vmime::utility::encoder::encoder> e = enc->create();

    std::vector <vmime::string> props = e->getAvailableProperties();
    std::vector <vmime::string>::const_iterator it;

    for (it = props.begin() ; it != props.end() ; ++it)
```

[5] This is an excerpt from EXAMPLE6

```
        std::cout << "        - " << *it << std::endl;
```

Listing 4.16: Enumerating encoders and their properties

4.10 Progress listeners

Progress listeners are used with objects that can notify you about the state of progress when they are performing an operation.

The `vmime::utility::progressListener` interface is rather simple:

```
void start(const int predictedTotal);
void progress(const int current, const int currentTotal);
void stop(const int total);
```

`start()` and `stop()` are called at the beginning and the end of the operation, respectively. `progress()` is called each time the status of progress changes (eg. a chunk of data has been processed). There is no unit specified for the values passed in argument. It depends on the notifier: it can be bytes, percent, number of messages...

Chapter 5

Parsing and Building Messages

5.1 Parsing messages

5.1.1 Introduction

Parsing is the process of creating a structured representation (for example, a hierarchy of C++ objects) of a message from its "textual" representation (the raw data that is actually sent on the Internet).

For example, say you have the following email in a file called "hello.eml":

```
Date: Thu, Oct 13 2005 15:22:46 +0200
From: Vincent <vincent@vmime.org>
To: you@vmime.org
Subject: Hello from VMime!

A simple message to test VMime
```

The following code snippet shows how you can easily obtain a `vmime::message` object from data in this file:

```
// Read data from file
std::ifstream file;
file.open("hello.eml", std::ios::in | std::ios::binary);

vmime::utility::inputStreamAdapter is(file);

vmime::string data;
vmime::utility::outputStreamStringAdapter os(data);

vmime::utility::bufferedStreamCopy(is, os);
```

```
// Actually parse the message
vmime::shared_ptr <vmime::message> msg = vmime::make_shared <vmime::message>();
msg->parse(data);

vmime::shared_ptr <vmime::header> hdr = msg->getHeader();
vmime::shared_ptr <vmime::body> bdy = msg->getBody();

// Now, you can extract some of its components
vmime::charset ch(vmime::charsets::UTF_8);

std::cout
   << "The subject of the message is: "
   << hdr->Subject()->getValue <vmime::text>()->getConvertedText(ch)
   << std::endl
   << "It was sent by: "
   << hdr->From()->getValue <vmime::mailbox>()->getName().getConvertedText(ch)
   << " (email: " << hdr->From()->getValue <vmime::mailbox>()->getEmail() << ")"
   << std::endl;
```

Listing 5.1: Parsing a message from a file

The output of this program is:

```
The subject of the message is: Hello from VMime!
It was sent by: Vincent (email: vincent@vmime.org)
```

5.1.2 Using the `vmime::messageParser` object

The `vmime::messageParser` object allows to parse messages in a more simple manner. You can obtain all the text parts and attachments as well as basic fields (expeditor, recipients, subject...), without dealing with MIME message structure.

```
// Read data from file
std::ifstream file;
file.open("hello.eml", std::ios::in | std::ios::binary);

vmime::utility::inputStreamAdapter is(file);

vmime::string data;
vmime::utility::outputStreamStringAdapter os(data);

vmime::utility::bufferedStreamCopy(is, os);
```

```
// Actually parse the message
vmime::shared_ptr <vmime::message> msg = vmime::make_shared <vmime::message>();
msg->parse(data);

// Here start the differences with the previous example
vmime::messageParser mp(msg);

// Output information about attachments
std::cout << "Message has " << mp.getAttachmentCount()
   << " attachment(s)" << std::endl;

for (int i = 0 ; i < mp.getAttachmentCount() ; ++i)
{
   vmime::shared_ptr <const vmime::attachment> att = mp.getAttachmentAt(i);
   std::cout << "  - " << att->getType().generate() << std::endl;
}

// Output information about text parts
std::cout << "Message has " << mp.getTextPartCount()
   << " text part(s)" << std::endl;

for (int i = 0 ; i < mp.getTextPartCount() ; ++i)
{
   vmime::shared_ptr <const vmime::textPart> tp = mp.getTextPartAt(i);

   // text/html
   if (tp->getType().getSubType() == vmime::mediaTypes::TEXT_HTML)
   {
      vmime::shared_ptr <const vmime::htmlTextPart> htp =
         vmime::dynamicCast <const vmime::htmlTextPart>(tp);

      // HTML text is in tp->getText()
      // Plain text is in tp->getPlainText()

      // Enumerate embedded objects
      for (int j = 0 ; j < htp->getObjectCount() ; ++j)
      {
         vmime::shared_ptr <const vmime::htmlTextPart::embeddedObject> obj =
            htp->getObjectAt(j);

         // Identifier (Content-Id or Content-Location) is obj->getId()
         // Object data is in obj->getData()
      }
   }
   // text/plain or anything else
   else
```

```
    {
        // Text is in tp->getText()
    }
}
```

Listing 5.2: Using `vmime::messageParser` to parse more complex messages

5.2 Building messages

5.2.1 A simple message

Of course, you can build a MIME message from scratch by creating the various objects that compose it (parts, fields, etc.). The following is an example of how to achieve it:

```
vmime::shared_ptr <vmime::message> msg = vmime::make_shared <vmime::message>();

vmime::shared_ptr <vmime::header> hdr = msg->getHeader();
vmime::shared_ptr <vmime::body> bdy = msg->getBody();

vmime::shared_ptr <vmime::headerFieldFactory> hfFactory =
    vmime::headerFieldFactory::getInstance();

// Append a 'Date:' field
vmime::shared_ptr <vmime::headerField> dateField =
    hfFactory->create(vmime::fields::DATE);

dateField->setValue(vmime::datetime::now());
hdr->appendField(dateField);

// Append a 'Subject:' field
vmime::shared_ptr <vmime::headerField> subjectField =
    hfFactory->create(vmime::fields::SUBJECT);

subjectField->setValue(vmime::text("Message subject"));
hdr->appendField(subjectField);

// Append a 'From:' field
vmime::shared_ptr <vmime::headerField> fromField =
    hfFactory->create(vmime::fields::FROM);

fromField->setValue
    (vmime::make_shared <vmime::mailbox>("me@vmime.org"));
hdr->appendField(fromField);
```

```
// Append a 'To:' field
vmime::shared_ptr <vmime::headerField> toField =
    hfFactory->create(vmime::fields::TO);

vmime::shared_ptr <vmime::mailboxList> recipients =
    vmime::make_shared <vmime::mailboxList>();

recipients->appendMailbox
    (vmime::make_shared <vmime::mailbox>("you@vmime.org"));

toField->setValue(recipients);
hdr->appendField(toField);

// Set the body contents
bdy->setContents(vmime::make_shared <vmime::stringContentHandler>
    ("This is the text of your message..."));

// Output raw message data to standard output
vmime::utility::outputStreamAdapter out(std::cout);
msg->generate(out);
```

Listing 5.3: Building a simple message from scratch

As you can see, this is a little fastidious. Hopefully, VMime also offers a more simple way for creating messages. The `vmime::messageBuilder` object can create basic messages that you can then customize.

The following code can be used to build exactly the same message as in the previous example, using the `vmime::messageBuilder` object:

```
try
{
    vmime::messageBuilder mb;

    // Fill in some header fields and message body
    mb.setSubject(vmime::text("Message subject"));
    mb.setExpeditor(vmime::mailbox("me@vmime.org"));
    mb.getRecipients().appendAddress
        (vmime::make_shared <vmime::mailbox>("you@vmime.org"));

    mb.getTextPart()->setCharset(vmime::charsets::ISO8859_15);
    mb.getTextPart()->setText(vmime::make_shared <vmime::stringContentHandler>
        ("This is the text of your message..."));

    // Message construction
    vmime::shared_ptr <vmime::message> msg = mb.construct();
```

```
    // Output raw message data to standard output
    vmime::utility::outputStreamAdapter out(std::cout);
    msg->generate(out);
}
// VMime exception
catch (vmime::exception& e)
{
    std::cerr << "vmime::exception: " << e.what() << std::endl;
}
// Standard exception
catch (std::exception& e)
{
    std::cerr << "std::exception: " << e.what() << std::endl;
}
```

Listing 5.4: Building a simple message using `vmime::messageBuilder`

5.2.2 Adding an attachment

Dealing with attachments is quite simple. Add the following code to the previous example to attach a file to the message:

```
// Create an attachment
vmime::shared_ptr <vmime::fileAttachment> att =
    vmime::make_shared <vmime::fileAttachment>
    (
      /* full path to file */ "/home/vincent/paris.jpg",
      /* content type */      vmime::mediaType("image/jpeg"),
      /* description */       vmime::text("My holidays in Paris")
    );

// You can also set some infos about the file
att->getFileInfo().setFilename("paris.jpg");
att->getFileInfo().setCreationDate
    (vmime::datetime("30 Apr 2003 14:30:00 +0200"));

// Add this attachment to the message
mb.appendAttachment(att);
```

Listing 5.5: Building a message with an attachment using `vmime::messageBuilder`

5.2.3 HTML messages and embedded objects

VMime also supports aggregate messages, which permits to build MIME messages containing HTML text and embedded objects (such as images). For more information about aggregate messages, please read RFC-2557 (*MIME Encapsulation of Aggregate Documents, such as HTML*).

Creating such messages is quite easy, using the `vmime::messageBuilder` object. The following code constructs a message containing text in both plain and HTML format, and a JPEG image:

```
// Fill in some header fields
mb.setSubject(vmime::text("An HTML message"));
mb.setExpeditor(vmime::mailbox("me@vmime.org"));
mb.getRecipients().appendAddress
    (vmime::make_shared <vmime::mailbox>("you@vmime.org"));

// Set the content-type to "text/html": a text part factory must be
// available for the type you are using. The following code will make
// the message builder construct the two text parts.
mb.constructTextPart(vmime::mediaType
    (vmime::mediaTypes::TEXT, vmime::mediaTypes::TEXT_HTML));

// Set contents of the text parts; the message is available in two formats:
// HTML and plain text. The HTML format also includes an embedded image.
vmime::shared_ptr <vmime::htmlTextPart> textPart =
    vmime::dynamicCast <vmime::htmlTextPart>(mb.getTextPart());

// -- Add the JPEG image (the returned identifier is used to identify the
// -- embedded object in the HTML text, the famous "CID", or "Content-Id").
// -- Note: you can also read data from a file; see the next example.
const vmime::string id = textPart->addObject("<...image data...>",
    vmime::mediaType(vmime::mediaTypes::IMAGE, vmime::mediaTypes::IMAGE_JPEG));

// -- Set the text
textPart->setCharset(vmime::charsets::ISO8859_15);

textPart->setText(vmime::make_shared <vmime::stringContentHandler>
    ("This is the <b>HTML text</b>, and the image:<br/>"
     "<img src=\"") + id + vmime::string("\"/>"));

textPart->setPlainText(vmime::make_shared <vmime::stringContentHandler>
    ("This is the plain text."));
```

Listing 5.6: Building an HTML message with an embedded image using the `vmime::messageBuilder`

This will create a message having the following structure:

```
multipart/alternative
   text/plain
   multipart/related
      text/html
      image/jpeg
```

You can easily tell VMime to read the embedded object data from a file. The following code opens the file */path/to/image.jpg*, connects it to an input stream, then add an embedded object:

```
vmime::utility::fileSystemFactory* fs =
    vmime::platform::getHandler()->getFileSystemFactory();

vmime::shared_ptr <vmime::utility::file> imageFile =
    fs->create(fs->stringToPath("/path/to/image.jpg"));

vmime::shared_ptr <vmime::contentHandler> imageCts =
    vmime::make_shared <vmime::streamContentHandler>
        (imageFile->getFileReader()->getInputStream(), imageFile->getLength());

const vmime::string cid = textPart.addObject(imageCts,
    vmime::mediaType(vmime::mediaTypes::IMAGE, vmime::mediaTypes::IMAGE_JPEG));
```

5.3 Working with attachments: the attachment helper

The `attachmentHelper` object allows listing all attachments in a message, as well as adding new attachments, without using the `messageParser` and `messageBuilders` objects. It can work directly on messages and body parts.

To use it, you do not need any knowledge about how attachment parts should be organized in a MIME message.

The following code snippet tests if a body part is an attachment, and if so, extract its contents to the standard output:

```
vmime::shared_ptr <vmime::bodyPart> part;  // suppose we have a body part

if (vmime::attachmentHelper::isBodyPartAnAttachment(part))
{
    // The body part contains an attachment, get it
    vmime::shared_ptr <const vmime::attachment> attach =
        attachmentHelper::getBodyPartAttachment(part);
```

```
    // Extract attachment data to standard output
    vmime::utility::outputStreamAdapter out(std::cout);
    attach->getData()->extract(out);
}
```

Listing 5.7: Testing if a body part is an attachment

You can also easily extract all attachments from a message:

```
vmime::shared_ptr <vmime::message> msg;  // suppose we have a message

const std::vector <ref <const attachment> > atts =
    attachmentHelper::findAttachmentsInMessage(msg);
```

Listing 5.8: Extracting all attachments from a message

Finally, the `attachmentHelper` object can be used to add an attachment to an existing message, whatever it contains (text parts, attachments, ...). The algorithm can modify the structure of the message if needed (eg. add a *multipart/mixed* part if no one exists in the message). Simply call the `addAttachment` function:

```
vmime::shared_ptr <vmime::message> msg;  // suppose we have a message

// Create an attachment
vmime::shared_ptr <vmime::fileAttachment> att =
    vmime::make_shared <vmime::fileAttachment>
    (
      /* full path to file */ "/home/vincent/paris.jpg",
      /* content type */      vmime::mediaType("image/jpeg"),
      /* description */       vmime::text("My holidays in Paris")
    );

// Attach it to the message
vmime::attachmentHelper::addAttachment(msg, att);
```

Listing 5.9: Adding an attachment to an existing message

Chapter 6

Working with Messaging Services

6.1 Introduction

In addition to parsing and building MIME messages, VMime also offers a lot of features to work with messaging services. This includes connecting to remote messaging stores (like IMAP or POP3), local stores (maildir) and transport services (send messages over SMTP or local sendmail), through an unified interface (see Figure 6.1). That means that you can use independently IMAP of POP3 without having to change any line of code.

Source code of EXAMPLE6 covers all features presented in this chapter, so it is important you take some time to read it.

The interface is composed of five classes:

- `vmime::net::service`: this is the base interface for a messaging service. It can be either a store service or a transport service.
- `vmime::net::serviceFactory`: create instances of a service. This is used internally by the session object (see below).
- `vmime::net::store`: interface for a store service. A store service offers access to a set of folders containing messages. This is used for IMAP, POP3 and maildir.
- `vmime::net::transport`: interface for a transport service. A transport service is capable of sending messages. This is used for SMTP and sendmail.
- `vmime::net::session`: a session object is used to store the parameters used by a service (eg. connection parameters). Each service instance is associated with only one session. The session object is capable of creating instances of services.

The following classes are specific to store services:

- `vmime::net::folder`: a folder can either contain other folders or messages, or both.

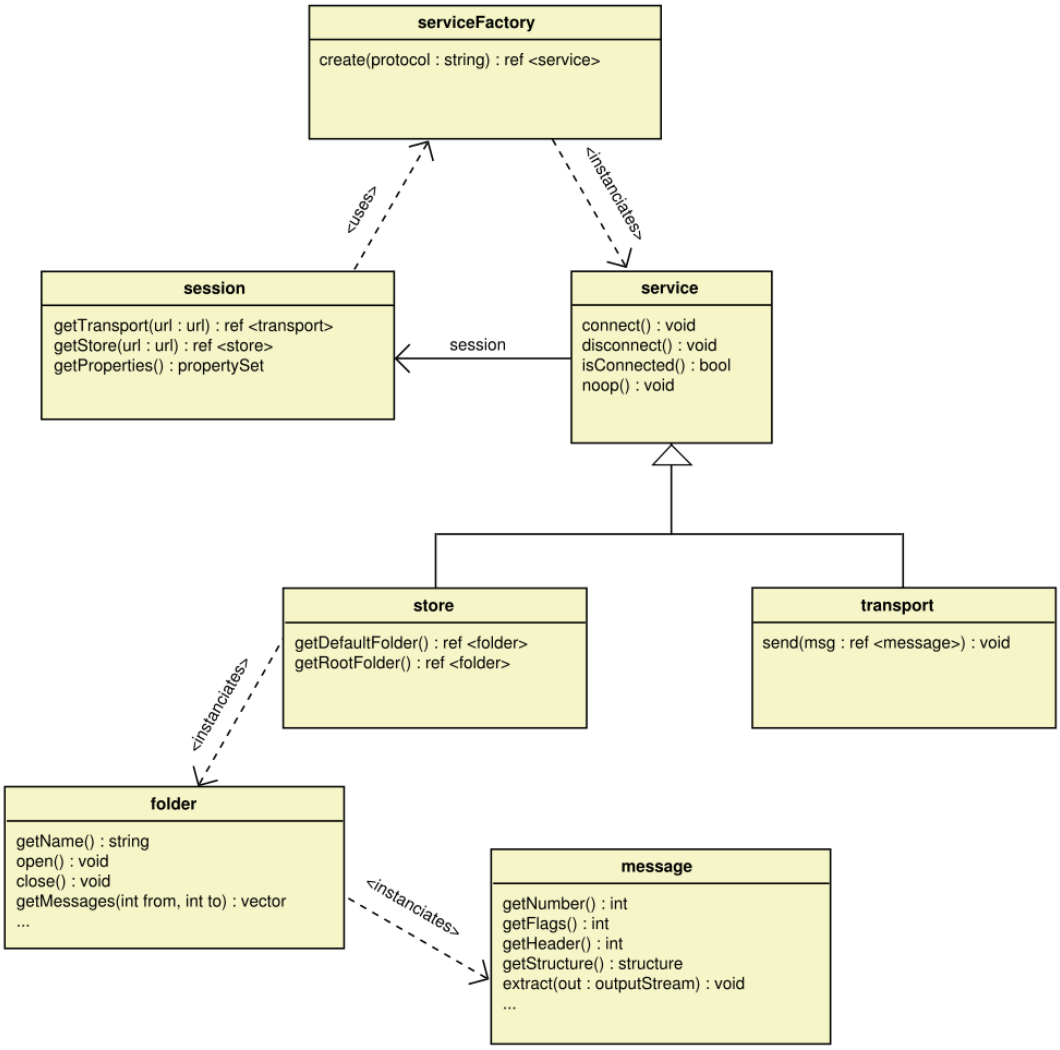

Figure 6.1: Overall structure of the messaging module

- `vmime::net::message`: this is the interface for dealing with messages. For a given message, you can have access to its flags, its MIME structure and you can also extract the whole message data or given parts (if supported by the underlying protocol).

6.2 Working with sessions

6.2.1 Setting properties

Sessions are used to store configuration parameters for services. They contains a set of typed properties that can modify the behaviour of the services. Before using a messaging service, you must create and initialize a session object:

```
vmime::shared_ptr <vmime::net::session> theSession = vmime::net::session::create();
```

Session properties include:

- connection parameters: host and port to connect to;
- authentication parameters: user credentials required to use the service (if any);
- protocol-specific parameters: enable or disable extensions (eg. APOP support in POP3).

Properties are stored using a dotted notation, to specify the service type, the protocol name, the category and the name of the property:

```
{service_type}.{protocol}.category.name
```

An example of property is *store.pop3.options.apop* (used to enable or disable the use of APOP authentication). The *store.pop3* part is called the *prefix*. This allow specifying different values for the same property depending on the protocol used.

The session properties are stored in a `vmime::propertySet` object. To set the value of a property, you can use either:

```
theSession->getProperties().setProperty("property-name", value);
```

or:

```
theSession->getProperties()["property-name"] = value;
```

6.2.2 Available properties

Following is a list of available properties and the protocols they apply to, as the time of writing this documentation[1]. For better clarity, the prefixes do not appear in this table.

Property name	Type	Description	POP3	POP3S	IMAP	IMAPS	SMTP	SMTPS	maildir	sendmail
options.sasl	bool	Set to `true` to use SASL authentication, if available.	•	•	•	•	•	•		
options.sasl.fallback	bool	Fail if SASL authentication failed (do not try other authentication mechanisms).	•	•	•	•	•	•		
auth.username[2]	string	Set the username of the account to connect to.	•	•	•	•	•	•		
auth.password[??]	string	Set the password of the account.	•	•	•	•	•	•		
connection.tls	bool	Set to `true` to start a secured connection using STARTTLS extension, if available.	•		•		•			
connection.tls.required	bool	Fail if a secured connection cannot be started.	•		•		•			
server.address	string	Server host name or IP address.	•	•	•	•	•	•		
server.port	int	Server port.	•	•	•	•	•	•		
server.rootpath	string	Root directory for mail repository (eg. */home/vincent/Mail*).							•	

Table 6.1: Properties common to all protocols

[1] You can get an up-to-date list of the properties by running EXAMPLE7

These are the protocol-specific options:

Property name	Type	Description
POP3, POP3S		
store.pop3.options.apop	bool	Enable or disable authentication with APOP (if SASL is enabled, this occurs after all SASL mechanisms have been tried).
store.pop3.options.apop.fallback	bool	If set to `true` and APOP fails, the authentication process fails (ie. unsecure plain text authentication is not used).
SMTP, SMTPS		
transport.smtp.options.need-authentication	bool	Set to *true* if the server requires to authenticate before sending messages.
transport.smtp.options.pipelining	bool	Set to `false` to disable command pipelining, if the server supports it (default is `true`).
transport.smtp.options.chunking	bool	Set to `false` to disable CHUNKING extension, if the server supports it (default is `true`).
sendmail		
transport.sendmail.binpath	string	The path to the *sendmail* executable on your system. The default is the one found by the configuration script when VMime was built.

Table 6.2: Protocol-specific options

6.2.3 Instanciating services

You can create a service either by specifying its protocol name, or by specifying the URL of the service. Creation by name is deprecated so this chapter only presents the latter option.

The URL scheme for connecting to services is:

```
protocol://[username[:password]@]host[:port]/[root-path]
```

NOTE: For local services (ie. *sendmail* and *maildir*), the host part is not used, but it must not be empty (you can use "localhost").

The following table shows an example URL for each service:

Service	Connection URL
imap, imaps	`imap://imap.example.com`, `imaps://vincent:pass@example.com`
pop3, pop3s	`pop3://pop3.example.com`
smtp, smtps	`smtp://smtp.example.com`
maildir	`maildir://localhost/home/vincent/Mail` (host not used)
sendmail	`sendmail://localhost` (host not used, always localhost)

When you have the connection URL, instanciating the service is quite simple. Depending on the type of service, you will use either `getStore()` or `getTransport()`. For example, for store services, use:

```
vmime::utility::url url("imap://user:pass@imap.example.com");
vmime::shared_ptr <vmime::net::store> st = sess->getStore(url);
```

and for transport services:

```
vmime::utility::url url("smtp://smtp.example.com");
vmime::shared_ptr <vmime::net::transport> tr = sess->getTransport(url);
```

6.3 User credentials and authenticators

Some services need some user credentials (eg. username and password) to open a session. In VMime, user credentials can be specified in the session properties or by using a custom authenticator (callback).

```
vmime::shared_ptr <vmime::net::session> sess;  // Suppose we have a session

sess->getProperties()["store.imap.auth.username"] = "vincent";
sess->getProperties()["store.imap.auth.password"] = "my-password";
```

Listing 6.1: Setting user credentials using session properties

Although not recommended, you can also specify username and password directly in the connection URL, ie: *imap://username:password@imap.example.com/*. This works only for services requiring an username and a password as user credentials, and no other information.

Sometimes, it may not be very convenient to set username/password in the session properties, or not possible (eg. extended SASL mechanisms) . That's why VMime offers an alternate way of getting user credentials: the `authenticator` object. Basically, an authenticator is an object that can return user credentials on-demand (like a callback).

Currently, there are two types of authenticator in VMime: a basic authenticator (class `vmime::security::authenticator`) and, if SASL support is enabled, a SASL authenticator (class `vmime::security::sasl::SASLAuthenticator`). Usually, you should use the default implementations, or at least make your own implementation inherit from them.

The following example shows how to use a custom authenticator to request the user to enter her/his credentials:

```
class myAuthenticator : public vmime::security::defaultAuthenticator
{
    const string getUsername() const
```

```
    {
        std::cout << "Enter your username: " << std::endl;

        vmime::string res;
        std::getline(std::cin, res);

        return res;
    }

    const string getPassword() const
    {
        std::cout << "Enter your password: " << std::endl;

        vmime::string res;
        std::getline(std::cin, res);

        return res;
    }
};
```

Listing 6.2: A simple interactive authenticator

This is how to use it:

```
// First, create a session
vmime::shared_ptr <vmime::net::session> sess = vmime::net::session::create();

// Next, initialize a service which will use our authenticator
vmime::shared_ptr <vmime::net::store> st =
    sess->getStore(vmime::utility::url("imap://imap.example.com"),
        /* use our authenticator */ vmime::make_shared <myAuthenticator>());
```

NOTE: An authenticator object should be used with one and only one service at a time. This is required because the authentication process may need to retrieve the service name (SASL).

Of course, this example is quite simplified. For example, if several authentication mechanisms are tried, the user may be requested to enter the same information multiple times. See EXAMPLE6 for a more complex implementation of an authenticator, with caching support.

If you want to use SASL (ie. if *options.sasl* is set to *true*), your authenticator must inherit from `vmime::security::sasl::SASLAuthenticator` or `vmime::security::sasl::defaultSASLAuthenticator`, even if you do not use the SASL-specific methods `getAcceptableMechanisms()` and `setSASLMechanism()`. Have a look at EXAMPLE6 to see an implementation of an SASL authenticator.

```
class mySASLAuthenticator : public vmime::security::sasl::defaultSASLAuthenticator
```

```
{
    typedef vmime::security::sasl::SASLMechanism mechanism;
// save us typing

    const std::vector <vmime::shared_ptr <mechanism> > getAcceptableMechanisms
            (const std::vector <vmime::shared_ptr <mechanism> >& available,
             vmime::shared_ptr <mechanism> suggested) const
    {
        // Here, you can sort the SASL mechanisms in the order they will be
        // tried. If no SASL mechanism is acceptable (ie. for example, not
        // enough secure), you can return an empty list.
        //
        // If you do not want to bother with this, you can simply return
        // the default list, which is ordered by security strength.
        return defaultSASLAuthenticator::
            getAcceptableMechanisms(available, suggested);
    }

    void setSASLMechanism(vmime::shared_ptr <mechanism> mech)
    {
        // This is called when the authentication process is going to
        // try the specified mechanism.
        //
        // The mechanism name is in mech->getName()

        defaultSASLAuthenticator::setSASLMechanism(mech);
    }

    // ...implement getUsername() and getPassword()...
};
```

Listing 6.3: A simple SASL authenticator

6.4 Using transport service

You have two possibilities for giving message data to the service when you want to send a message:

- either you have a reference to a message (type `vmime::message`) and you can simply call `send(msg);`
- or you only have raw message data (as a string, for example), and you have to call the second overload of `send()`, which takes additional parameters (corresponding to message envelope);

The following example illustrates the use of a transport service to send a message using the second method:

```
const vmime::string msgData =
    "From: me@example.org \r\n"
    "To: you@example.org \r\n"
    "Date: Sun, Oct 30 2005 17:06:42 +0200 \r\n"
    "Subject: Test \r\n"
    "\r\n"
    "Message body";

// Create a new session
vmime::utility::url url("smtp://example.com");

vmime::shared_ptr <vmime::net::session> sess = vmime::net::session::create();

// Create an instance of the transport service
vmime::shared_ptr <vmime::net::transport> tr = sess->getTransport(url);

// Connect it
tr->connect();

// Send the message
vmime::utility::inputStreamStringAdapter is(msgData);

vmime::mailbox from("me@example.org");
vmime::mailboxList to;
to.appendMailbox(vmime::make_shared <vmime::mailbox>("you@example.org"));

tr->send(
    /* expeditor */     from,
    /* recipient(s) */ to,
    /* data */          is,
    /* total length */ msgData.length());

// We have finished using the service
tr->disconnect();
```

Listing 6.4: Using a transport service

NOTE: Exceptions can be thrown at any time when using a service. For better clarity, exceptions are not caught here, but be sure to catch them in your own application to provide error feedback to the user.

If you use SMTP, you can enable authentication by setting some properties on the session object (`service::setProperty()` is a shortcut for setting properties on the session with

the correct prefix):

```
tr->setProperty("options.need-authentication", true);
tr->setProperty("auth.username", "user");
tr->setProperty("auth.password", "password");
```

6.5 Using store service

6.5.1 Connecting to a store

The first basic step for using a store service is to connect to it. The following example shows how to initialize a session and instanciate the store service:

```
// Create a new session
vmime::utility::url url("imap://vincent:password@imap:example.org");

vmime::shared_ptr <vmime::net::session> sess = vmime::net::session::create();

// Create an instance of the transport service
vmime::shared_ptr <vmime::net::store> store = sess->getStore(url);

// Connect it
store->connect();
```

Listing 6.5: Connecting to a store service

NOTE: EXAMPLE6 contains a more complete example for connecting to a store service, with support for a custom authenticator.

6.5.2 Opening a folder

You can open a folder using two different access modes: either in *read-only* mode (where you can only read message flags and contents), or in *read-write* mode (where you can read messages, but also delete them or add new ones). When you have a reference to a folder, simply call the `open()` method with the desired access mode:

```
folder->open(vmime::net::folder::MODE_READ_WRITE);
```

NOTE: Not all stores support the *read-write* mode. By default, if the *read-write* mode is not available, the folder silently fall backs on the *read-only* mode, unless the *failIfModeIsNotAvailable* argument to `open()` is set to true.

Call `getDefaultFolder()` on the store to obtain a reference to the default folder, which is usually the INBOX folder (where messages arrive when they are received).

You can also open a specific folder by specifying its path. The following example will open a folder named *bar*, which is a child of *foo* in the root folder:

```
vmime::net::folder::path path;
path /= vmime::net::folder::path::component("foo");
path /= vmime::net::folder::path::component("bar");

vmime::shared_ptr <vmime::net::folder> fld = store->getFolder(path);
fld->open(vmime::net::folder::MODE_READ_WRITE);
```

Listing 6.6: Opening a folder from its path

NOTE: You can specify a path as a string as there is no way to get the separator used to delimitate path components. Always use `operator/=` or `appendComponent`.

NOTE: Path components are of type `vmime::word`, which means that VMime supports folder names with extended characters, not only 7-bit US-ASCII. However, be careful that this may not be supported by the underlying store protocol (IMAP supports it, because it uses internally a modified UTF-7 encoding).

6.5.3 Fetching messages

You can fetch some information about a message without having to download the whole message. Moreover, folders support fetching for multiple messages in a single request, for better performance. The following items are currently available for fetching:

- **envelope**: sender, recipients, date and subject;
- **structure**: MIME structure of the message;
- **content-info**: content-type of the root part;
- **flags**: message flags;
- **size**: message size;
- **header**: retrieve all the header fields of a message;
- **uid**: unique identifier of a message;
- **importance**: fetch header fields suitable for use with `misc::importanceHelper`.

NOTE: Not all services support all fetchable items. Call `getFetchCapabilities()` on a folder to know which information can be fetched by a service.

The following code shows how to list all the messages in a folder, and retrieve basic information to show them to the user:

```
std::vector <ref <vmime::net::message> > allMessages =
    folder->getMessages(vmime::net::messageSet::byNumber(1, -1));
        // -1 is a special value to mean "the number of the last message in the folder"

folder->fetchMessages(allMessages,
    vmime::net::fetchAttributes::FLAGS |
    vmime::net::fetchAttributes::ENVELOPE);

for (unsigned int i = 0 ; i < allMessages.size() ; ++i)
{
    vmime::shared_ptr <vmime::net::message> msg = allMessages[i];

    const int flags = msg->getFlags();

    std::cout << "Message " << i << ":" << std::endl;

    if (flags & vmime::net::message::FLAG_SEEN)
        std::cout << " - is read" << std::endl;
    if (flags & vmime::net::message::FLAG_DELETED)
        std::cout << " - is deleted" << std::endl;

    vmime::shared_ptr <const vmime::header> hdr = msg->getHeader();

    std::cout << " - sent on " << hdr->Date()->generate() << std::endl;
    std::cout << " - sent by " << hdr->From()->generate() << std::endl;
}
```

Listing 6.7: Fetching information about multiple messages

IMAP supports fetching specific header fields of a message. Here is how to use the `fetchAttributes` object to do it:

```
// Fetch message flags and the "Received" and "X-Mailer" header fields
vmime::net::fetchAttributes fetchAttribs;
fetchAttribs.add(vmime::net::fetchAttributes::FLAGS);
fetchAttribs.add("Received");
fetchAttribs.add("X-Mailer");

folder->fetchMessages(allMessages, fetchAttribs);
```

Listing 6.8: Using fetchAttributes object to fetch specific header fields of a message

6.5.4 Extracting messages and parts

To extract the whole contents of a message (including headers), use the `extract()` method on a `vmime::net::message` object. The following example extracts the first message in the default folder:

```
// Get a reference to the folder and to its first message
vmime::shared_ptr <vmime::net::folder> folder = store->getDefaultFolder();
vmime::shared_ptr <vmime::net::message> msg = folder->getMessage(1);

// Write the message contents to the standard output
vmime::utility::outputStreamAdapter out(std::cout);
msg->extract(out);
```

Listing 6.9: Extracting messages

Some protocols (like IMAP) also support the extraction of specific MIME parts of a message without downloading the whole message. This can save bandwidth and time. The method `extractPart()` is used in this case:

```
// Fetching structure is required before extracting a part
folder->fetchMessage(msg, vmime::net::fetchAttributes::STRUCTURE);

// Now, we can extract the part
msg->extractPart(msg->getStructure()->getPartAt(0)->getPartAt(1));
```

Listing 6.10: Extracting a specific MIME part of a message

Suppose we have a message with the following structure:

```
multipart/mixed
   text/html
   image/jpeg [*]
```

The previous example will extract the header and body of the *image/jpeg* part.

6.5.5 Deleting messages

The following example will delete the second and the third message from the store.

```
vmime::shared_ptr <vmime::net::folder> folder = store->getDefaultFolder();

folder->deleteMessages(vmime::net::messageSet::byNumber(/* from */ 2, /* to */ 3));
```

```
// This is equivalent
std::vector <int> nums;
nums.push_back(2);
nums.push_back(3);
folder->deleteMessages(vmime::net::messageSet::byNumber(nums));

// This is also equivalent (but will require 2 roundtrips to server)
folder->deleteMessages(vmime::net::messageSet::byNumber(2));
folder->deleteMessages(vmime::net::messageSet::byNumber(2)); // renumbered, 3 becomes 2
```

Listing 6.11: Deleting messages

6.5.6 Events

As a result of executing some operation (or from time to time, even if no operation has been performed), a store service can send events to notify you that something has changed (eg. the number of messages in a folder). These events may allow you to update the user interface associated to a message store.

Currently, there are three types of event:

- **message change**: sent when the number of messages in a folder has changed (ie. some messages have been added or removed);
- **message count change**: sent when one or more message(s) have changed (eg. flags or deleted status);
- **folder change**: sent when a folder has been created, renamed or deleted.

You can register a listener for each event type by using the corresponding methods on a `folder` object: `addMessageChangedListener()`, `addMessageCountListener()` or `addFolderListener()`. For more information, please read the class documentation for `vmime::net::events` namespace.

6.6 Handling timeouts

Unexpected errors can occur while messaging services are performing operations and waiting a response from the server (eg. server stops responding, network link falls down). As all operations as synchronous, they can be "blocked" a long time before returning (in fact, they loop until they either receive a response from the server, or the underlying socket system returns an error).

VMime provides a mechanism to control the duration of operations. This mechanism allows the program to cancel an operation that is currently running.

An interface called `timeoutHandler` is provided:

```
class timeoutHandler : public object
{
   /** Called to test if the time limit has been reached.
     *
     * @return true if the timeout delay is elapsed
     */
   virtual const bool isTimeOut() = 0;

   /** Called to reset the timeout counter.
     */
   virtual void resetTimeOut() = 0;

   /** Called when the time limit has been reached (when
     * isTimeOut() returned true).
     *
     * @return true to continue (and reset the timeout)
     * or false to cancel the current operation
     */
   virtual const bool handleTimeOut() = 0;
};
```

While the operation runs, the service calls `isTimeout()` at variable intervals. If the `isTimeout()` function returns `true`, then `handleTimeout()` is called. If the `handleTimeout()` function returns `false`, the operation is cancelled and an `operation_timed_out` exception is thrown. Else, if `handleTimeout()` returns true, the operation continues and the timeout counter is reset. The function `resetTimeout()` is called each time data has been received from the server to reset the timeout delay.

When using a service, a default timeout handler is set: if an operation is blocked for more than 30 seconds (ie. network link is down and no data was received since 30 seconds), an `operation_timed_out` exception is thrown.

The following example shows how to implement a simple timeout handler:

```
class myTimeoutHandler : public vmime::net::timeoutHandler
{
public:

   myTimeoutHandler()
   {
      m_startTime = time(NULL);
   }

   const bool isTimeOut()
   {
      return (time(NULL) >= m_startTime + 30);   // 30 seconds timeout
```

```
    }

    void resetTimeOut()
    {
        m_startTime = time(NULL);
    }

    const bool handleTimeOut()
    {
        std::cout << "Operation timed out." << std::endl;
                  << "Press [Y] to continue, or [N] to "
                  << "cancel the operation." << std::endl;

        std::string response;
        std::cin >> response;

        return (response == "y" || response == "Y");
    }

private:

    time_t m_startTime;
};
```

Listing 6.12: Implementing a simple timeout handler

To make the service use your timeout handler, you need to write a factory class, to allow the service to create instances of the handler class. This is required because the service can use several connections to the server simultaneously, and each connection needs its own timeout handler.

```
class myTimeoutHandlerFactory : public vmime::net::timeoutHandlerFactory
{
public:

    ref <timeoutHandler> create()
    {
        return vmime::make_shared <myTimeoutHandler>();
    }
};
```

Then, call the `setTimeoutHandlerFactory()` method on the service object to set the timeout handler factory to use during the session:

```
theService->setTimeoutHandlerFactory(vmime::make_shared <myTimeoutHandlerFactory>());
```

6.7 Secured connection using TLS/SSL

6.7.1 Introduction

If you have enabled TLS support in VMime, you can configure messaging services so that they use a secured connection.

Quoting from RFC-2246 - the TLS 1.0 protocol specification: *" The TLS protocol provides communications privacy over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery."*

TLS has the following advantages:

- authentication: server identity can be verified;
- privacy: transmission of data between client and server cannot be read by someone in the middle of the connection;
- integrity: original data which is transferred between a client and a server can not be modified by an attacker without being detected.

NOTE: What is the difference between SSL and TLS? SSL is a protocol designed by Netscape. TLS is a standard protocol, and is partly based on version 3 of the SSL protocol. The two protocols are not interoperable, but TLS does support a mechanism to back down to SSL 3.

VMime offers two possibilities for using a secured connection:

- you can connect to a server listening on a special port (eg. IMAPS instead of IMAP): this is the classical use of SSL, but is now deprecated;
- connect to a server listening on the default port, and then begin a secured connection: this is STARTTLS.

6.7.2 Setting up a secured connection

6.7.2.1 Connecting to a "secured" port

To use the classical SSL/TLS way, simply use the "S" version of the protocol to connect to the server (eg. *imaps* instead of *imap*). This is currently available for SMTP, POP3 and IMAP.

```
vmime::shared_ptr <vmime::net::store> store =
    theSession->getStore(vmime::utility::url("imaps://example.org"));
```

6.7.2.2 Using STARTTLS

To make the service start a secured session using the STARTTLS method, simply set the *connection.tls* property:

```
theService->setProperty("connection.tls", true);
```

NOTE: If, for some reason, a secured connection cannot be started, the default behaviour is to fallback on a normal connection. To make `connect()` fail if STARTTLS fails, set the *connection.tls.required* to *true*.

6.7.3 Certificate verification

6.7.3.1 How it works

If you tried the previous examples, a `certificateException` might have been thrown. This is because the default certificate verifier in VMime did not manage to verify the certificate, and so could not trust it.

Basically, when you connect to a server using TLS, the server responds with a list of certificates, called a certificate chain (usually, certificates are of type X.509[3]). The certificate chain is ordered so that the first certificate is the subject certificate, the second is the subject's issuer one, the third is the issuer's issuer, and so on.

To decide whether the server can be trusted or not, you have to verify that *each* certificate is valid (ie. is trusted). For more information about X.509 and certificate verification, see related articles on Wikipedia [4].

6.7.3.2 Using the default certificate verifier

The default certificate verifier maintains a list of root (CAs) and user certificates that are trusted. By default, the list is empty. So, you have to initialize it before using the verifier.

The algorithm[5] used is quite simple:

1. for every certificate in the chain, verify that the certificate has been issued by the next certificate in the chain;
2. for every certificate in the chain, verify that the certificate is valid at the current time;
3. ensure that the first certificate's subject name matches the hostname of the server;

[3] And VMime currently supports only X.509 certificates
[4] See http://wikipedia.org/wiki/Public_key_certificate
[5] See http://wikipedia.org/wiki/Certification_path_validation_algorithm

4. decide whether the subject's certificate can be trusted:
    - first, verify that the the last certificate in the chain was issued by a third-party that we trust (root CAs);
    - if the issuer certificate cannot be verified against root CAs, compare the subject's certificate against the trusted certificates (the certificates the user has decided to trust).

First, we need some code to load existing X.509 certificates:

```
vmime::shared_ptr <vmime::security::cert::X509Certificate>
    loadX509CertificateFromFile(const std::string& path)
{
    std::ifstream certFile;
    certFile.open(path.c_str(), std::ios::in | std::ios::binary);

    if (!certFile)
    {
        // ...handle error...
    }

    vmime::utility::inputStreamAdapter is(certFile);
    vmime::shared_ptr <vmime::security::cert::X509Certificate> cert;

    // Try DER format
    cert = vmime::security::cert::X509Certificate::import
        (is, vmime::security::cert::X509Certificate::FORMAT_DER);

    if (cert != NULL)
        return cert;

    // Try PEM format
    is.reset();
    cert = vmime::security::cert::X509Certificate::import
        (is, vmime::security::cert::X509Certificate::FORMAT_PEM);

    return cert;
}
```

Listing 6.13: Reading a X.509 certificate from a file

Then, we can use the `loadX509CertificateFromFile` function to load certificates and initialize the certificate verifier:

```
vmime::shared_ptr <vmime::security::cert::defaultCertificateVerifier> vrf =
```

```
    vmime::make_shared <vmime::security::cert::defaultCertificateVerifier>();

// Load root CAs (such as Verisign or Thawte)
std::vector <vmime::shared_ptr <vmime::security::cert::X509Certificate> > rootCAs;

rootCAs.push_back(loadX509CertificateFromFile("/path/to/root-ca1.cer");
rootCAs.push_back(loadX509CertificateFromFile("/path/to/root-ca2.cer");
rootCAs.push_back(loadX509CertificateFromFile("/path/to/root-ca3.cer");

vrf->setX509RootCAs(rootCAs);

// Then, load certificates that the user explicitely chose to trust
std::vector <vmime::shared_ptr <vmime::security::cert::X509Certificate> > trusted;

trusted.push_back(loadX509CertificateFromFile("/path/to/trusted-site1.cer");
trusted.push_back(loadX509CertificateFromFile("/path/to/trusted-site2.cer");

vrf->setX509TrustedCerts(trusted);
```

Listing 6.14: Using the default certificate verifier

6.7.3.3 Writing your own certificate verifier

If you need to do more complex verifications on certificates, you will have to write your own verifier. Your verifier should inherit from the `vmime::security::cert::certificateVerifier` class and implement the method `verify()`. Then, if the specified certificate chain is trusted, simply return from the function, or else throw a `certificateException`.

The following example shows how to implement an interactive certificate verifier which relies on the user's decision, and nothing else (you SHOULD NOT use this in a production application as this is obviously a serious security issue):

```
class myCertVerifier : public vmime::security::cert::certificateVerifier
{
public:

    void verify(vmime::shared_ptr <certificateChain> certs)
    {
        // Obtain the subject's certificate
        vmime::shared_ptr <vmime::security::cert::certificate> cert = chain->getAt(0);

        std::cout << std::endl;
        std::cout << "Server sent a '" << cert->getType() << "'"
            << " certificate." << std::endl;
```

```
        std::cout << "Do you want to accept this certificate? (Y/n) ";
        std::cout.flush();

        std::string answer;
        std::getline(std::cin, answer);

        if (answer.length() != 0 && (answer[0] == 'Y' || answer[0] == 'y'))
            return;  // OK, we trust the certificate

        // Don't trust this certificate
        throw vmime::security::cert::certificateException();
    }
};
```

Listing 6.15: A custom certificate verifier

NOTE: In production code, it may be a good idea to remember user's decisions about which certificates to trust and which not. See EXAMPLE6 for a basic cache implementation.

Finally, to make the service use your own certificate verifier, simply write:

```
theService->setCertificateVerifier(vmime::make_shared <myCertVerifier>());
```

6.7.4 SSL/TLS Properties

If you want to customize behavior or set some options on TLS/SSL connection, you may use the TLSProperties object, and pass it to the service session. The TLS/SSL options must be set *before* creating any service with the session (ie. before calling either `getStore()` or `getTransport()` on the session), or they will not be used.

The following example shows how to set the cipher suite preferences for TLS:

```
vmime::shared_ptr <vmime::net::session> sess = /* ... */;

vmime::shared_ptr <vmime::net::tls::TLSProperties> tlsProps =
        vmime::make_shared <vmime::net::tls::TLSProperties>();

// for OpenSSL
tlsProps->setCipherString("HIGH:!ADH:@STRENGTH");

// for GNU TLS
tlsProps->setCipherString("NORMAL:%SSL3_RECORD_VERSION");
```

```
sess->setTLSProperties(tlsProps);
```

Listing 6.16: Setting TLS cipher suite preferences

Please note that the cipher suite string format and meaning depend on the underlying TLS library (either OpenSSL or GNU TLS):

- for GNU TLS, read this:
  http://gnutls.org/manual/html_node/Priority-Strings.html
- for OpenSSL, read this:
  http://www.openssl.org/docs/apps/ciphers.html#CIPHER_STRINGS

You may also set cipher suite preferences using predefined constants that map to generic security modes:

```
sess->setCipherSuite(vmime::net::tls::TLSProperties::CIPHERSUITE_HIGH);
```

Listing 6.17: Setting TLS cipher suite preferences using predefined modes

The following constants are available:

Constant	**Meaning**
CIPHERSUITE_HIGH	High encryption cipher suites (> 128 bits)
CIPHERSUITE_MEDIUM	Medium encryption cipher suites (>= 128 bits)
CIPHERSUITE_LOW	Low encryption cipher suites (>= 64 bits)
CIPHERSUITE_DEFAULT	Default cipher suite (actual cipher suites used depends on the underlying SSL/TLS library)

Listings

3.1 Initializing VMime and the platform handler . . . 13

4.1 Smarts pointers and creating objects . . . 16

4.2 Casting smart pointers . . . 17

4.3 Catching VMime exceptions . . . 18

4.4 Using `vmime::datetime` object . . . 20

4.5 Using mailboxes and mailbox groups . . . 21

4.6 Getting and setting parameter value in fields . . . 24

4.7 Using stream adapters . . . 25

4.8 Using content handlers to extract body text from a message . . . 26

4.9 Setting the contents of a body part . . . 27

4.10 Creating an attachment from a file . . . 27

4.11 Extracting and converting body contents to a specified charset . . . 28

4.12 Creating `vmime::text objects` . . . 29

4.13 Decoding RFC-2047-encoded data . . . 30

4.14 Converting data in a `vmime::text` to a specified charset . . . 30

4.15 A simple example of using an encoder . . . 30

4.16 Enumerating encoders and their properties . . . 31

5.1 Parsing a message from a file . . . 33

5.2 Using `vmime::messageParser` to parse more complex messages . . . 34

5.3 Building a simple message from scratch . . . 36

5.4 Building a simple message using `vmime::messageBuilder` . . . 37

5.5 Building a message with an attachment using `vmime::messageBuilder` . . . 38

5.6 Building an HTML message with an embedded image using the `vmime::messageBuilder` . . . 39
5.7 Testing if a body part is an attachment . . . 40
5.8 Extracting all attachments from a message . . . 41
5.9 Adding an attachment to an existing message . . . 41
6.1 Setting user credentials using session properties . . . 48
6.2 A simple interactive authenticator . . . 48
6.3 A simple SASL authenticator . . . 49
6.4 Using a transport service . . . 51
6.5 Connecting to a store service . . . 52
6.6 Opening a folder from its path . . . 53
6.7 Fetching information about multiple messages . . . 54
6.8 Using fetchAttributes object to fetch specific header fields of a message . . . 54
6.9 Extracting messages . . . 55
6.10 Extracting a specific MIME part of a message . . . 55
6.11 Deleting messages . . . 55
6.12 Implementing a simple timeout handler . . . 57
6.13 Reading a X.509 certificate from a file . . . 61
6.14 Using the default certificate verifier . . . 61
6.15 A custom certificate verifier . . . 62
6.16 Setting TLS cipher suite preferences . . . 63
6.17 Setting TLS cipher suite preferences using predefined modes . . . 64

List of Figures

4.1 Diagram for address-related classes 21

4.2 Overall structure of MIME messages 22

6.1 Overall structure of the messaging module 43

List of Tables

2.1 CMake build options 11

4.1 Standard fields and their types 23

4.2 Standard parameterized fields 25

6.1 Properties common to all protocols 45

6.2 Protocol-specific options 46

Appendix A

The GNU General Public License

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if

you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of

works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other

than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do

not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a

copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent

the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the

additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may

otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest

to attach them to the start of each source file to most effectively
state the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year> <name of author>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

  If the program does terminal interaction, make it output a short
notice like this when it starts in an interactive mode:

    <program> Copyright (C) <year> <name of author>
    This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate
parts of the General Public License. Of course, your program's commands
might be different; for a GUI interface, you would use an "about box".

  You should also get your employer (if you work as a programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<http://www.gnu.org/licenses/>.

  The GNU General Public License does not permit incorporating your program
into proprietary programs. If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library. If this is what you want to do, use the GNU Lesser General
Public License instead of this License. But first, please read
<http://www.gnu.org/philosophy/why-not-lgpl.html>.