MagiCBuild

Configuration and build system

Version 0.1

User's Guide

Marko Grönroos (magi@iki.fi)

August 5th 2003

About this document

This document provides information about the MagiCBuild configuration and build system.

Copyright and License

User's Guide

MagiCBuild version 0.1

Special copyright chapter

The copyright is waived for the chapter titled in this document as "Building a package" to allow the use of the documentation in the installation instructions for software that uses MagiCBuild build system.

No material from other parts of this document many be transferred to that chapter without a prior written permission from the copyright owner, as the material would need to have its copyright waived. If the permission is not granted for such transferral, the modified chapter will follow the copyright and license of the transferred material.

Table of Contents

Chapter 1 Introduction

MagiCBuild is a build system that provides high-level functionality for compiling a software product from source code. It is based on GNU Make and consists of a framework of makefiles.

The most important feature of MagiCBuild is that all output and intermediate files are written to an output directory tree that is totally separate from source code tree. This helps keep the source tree completely clean from trash such as object files and executables. After compilation, the executable binaries, libraries, headers, shared files, and documentation can be "installed" from the output directory to actual installation directory hierarchy.

MagiCBuild also contains a configuration tool, "`configure`", that can be used to automatically detect relevant features of the platform such as availability and version numbers of required software packages, output and installation directories, and machine architecture. The configuration tool is a shell script, which makes extending its functionality very easy. The basic usage of the configuration tool is compatible with GNU Autoconf, making its use easy for open source software developers.

An example of using the configuration and build system is provided in the `test` subdirectory of the MagiCBuild distribution package.

Features

- Compiling C++ applications
- Compilation output tree separate from source tree
- Building reference documentation with Doxygen
- Installing and uninstalling
- Cleaning up the output directory
- Creation of source code distribution packages
- Configuration tool with GNU Autoconf compatibility

Limitations

- No installation to system directories

- Only GNU/Linux platform currently supported
- Only C++ compilation currently supported with gcc

For a more complete list of limitations, see the chapter *Known bugs and limitations*.

1.1. System requirements

MagiCBuild has the following system requirements:

- GNU/Linux operating system
- GNU Make 3.79.1 or newer
- Doxygen (optional)

Platforms

The following Linux distributions have been tested:

Distribution
Red Hat Linux 9
Mandrake 9.1
Debian 2.2 + upgrades

1.2. Licensing

MagiCBuild is licensed under the *GNU Lesser General Public License* (LGPL). The GNU Lesser General Public License is given in file `docs/COPYING.LIB`.

This documentation is licensed under the *GNU Free Documentation License*, as presented in the Chapter 7.

Chapter 2 Installing MagiCBuild

This chapter describes the installation procedure of MagiCBuild package.

Note: *There is currently no way to "install" MagiCBuild to standard system directories, but you must use it from the distribution directory.*

2.1. Unpacking the distribution package

If you wish to make the distribution package available system-wide, you need to login in as root and change to the directory under which you wish to unpack the distribution package.

```
# cd /opt
```

The source code is provided as a `tar` package compressed with BZip2 (`bz2`). If you have GNU Tar, you can unpack the package with the following shell command:

```
# tar jxf magicbuild-0.1beta1.tar.bz2
```

This will unpack the package in an appropriate subdirectory under the current directory.

2.2. Installation to user project

This section describes how to install MagiCBuild build system to your software project. The installation copies all of its relevant files to the project directory.

MagiCBuild is used from the distribution directory that was contained in the distribution package.

To install MagiCBuild files in your own programming project, change to the *root directory of your own project* (not the MagiCBuild directory) and run the install-magicbuild script from the MagiCBuild distribution directory. For example:

```
$ cd mysources
$ /opt/magicbuild-0.1beta1/build/install-magicbuild
```

This will create a "`build`" subdirectory and copy all the MagiCBuild files there. The configuration script will be copied to the current directory, as will be a template top-level `Makefile`.

```
mkdir: created directory `build'
`../configure' -> `configure'
`../build/makefile.template' -> `Makefile.template'
`../build/makefile.template' -> `build/makefile.template'
`../build/install-magicbuild' -> `build/install-magicbuild'
`../build/magicdef.mk' -> `build/magicdef.mk'
`../build/magiccmp.mk' -> `build/magiccmp.mk'
`../build/magicdist.mk' -> `build/magicdist.mk'
`../build/magictop.mk' -> `build/magictop.mk'
`../build/magicver.mk' -> `build/magicver.mk'
`../build/toDox.pl' -> `build/toDox.pl'
```

A file is not copied if there already exists a newer file with same name.

Chapter 3 Building a package

This chapter provides instructions for configuring, building, and actually installing a software package managed by MagiCBuild build system.

The installation instructions for the software package would probably be very much like the instructions in this chapter.

Note: *You are granted a permission to use any material in this (and only this) chapter for the installation instructions of your software. We hereby waive the copyrights for the contents of this chapter.*

3.1. Opening the source package

The source code is normally provided as a GNU Tar package compressed with `bz2`. You can unpack it with the following shell command:

```
tar jxf mysoftware-0.1beta1.tar.bz2
```

This will unpack the source code into an appropriate subdirectory under the current directory.

3.2. Configuring

To configure the source code for compilation, change to the source directory and run the `configure` script as follows:

```
cd mysoftware-0.1beta1
./configure
```

Optionally, if you wish to later install the package (headers and library) to some other than the default directory, you need to set the installation path with the `--prefix` attribute:

```
./configure --prefix=/opt/mysoftware
```

The default path for *root* user is `/usr/local`, and for other users their home directory.

No other configuration flags are currently supported.

3.3. Compiling

Include dependencies have to be determined before actual compiling, with the following command:

```
make deps
```

This may produce some errors, which are usually not relevant. Making dependencies is important if you intend to recompile the sources after making changes to them.

The package is compiled with the following simple command:

```
make
```

3.3.1. Compilation output

The output binaries as well as any intermediate files of the compilation will be located in an output directory tree separate from the source tree.

The build framework does the compilation output in separate directory, determined by the configuration script. The default output directory is located in:

`/tmp/$USER/build/<`*`architecture`*`>/release`

where `$USER` is the user name and *architecture* is the operating system and processor architecture, for example, `Linux-i686`.

For example, binaries are found under the `bin` subdirectory:

```
cd /tmp/$USER/build/Linux-i686/release/bin
./some_binary
```

Yoy can clean the output with the following command in the top-level source directory:

```
make clean
```

You do not normally need to clean the output.

3.4. Installing

After compiling, you can be install the package under the configured installation directory (see above) by issuing the following command in the source directory:

```
make install
```

This will copy the output library binaries and header files to appropriate subdirectories under the installation directory.

Directory	***Description***
`<instdir>/lib`	Libraries
`<instdir>/bin`	Binaries
`<instdir>/include`	Header files

3.5. Uninstalling

You can remove the installation by giving the following command in the source directory:

```
make uninstall
```

This removes the installed files and directories only if the installation path has not been changed with `configure` script after installing.

Chapter 4 Using the configuration system

This chapter gives a detailed description of the "`configure`" configuration system included with MagiCBuild and instructions for using it in your own software projects.

Notice that the configuration system is currently really minimal and provides only basic functionality.

4.1. Overview

The configuration system is defined entirely in the `configure` script copied to the root directory of the source tree. Application-specific requirements are defined in file `build/conf-reqs.sh`.

The configuration system follows a commonly used scheme: detect the features and write them to a file as variable definitions. Some other configuration systems write to a shell script that is run by the user to get the values as environment variables. Our system avoids this sort of contamination of the environment and writes the variables to a makefile include file, `build/config.mk`. Actually, as the makefile contains only lines such as "`export SRC=/path...`", you can also use the makefile as a shell script in Bash (Bourne-Again Shell).

Some features are always checked. These include operating system, processor architecture, and C++ compiler.

An example of using the configuration system is provided in the `test` subdirectory of the MagiCBuild distribution package.

4.2. Requirements file

Application-specific requirements and any additional checking and configuration code are placed in file `build/conf-reqs.sh`.

The following requirement checks are currently defined:

Call	***Description***
`check_for_qt`	Qt C++ library by Trolltech

Call	*Description*
`check_for_libjpeg`	JPEG library

Note: it is not currently possible to check the version numbers of the required components.

4.3. Extending requirement checks

It is possible to define custom checks for the configuration system, although this feature is still somewhat immature.

The custom checks are written to the user-defined `conf-reqs.sh` file (or another file included from that file) in the `build` subdirectory of the top-level project directory (`$SRCDIR`), just like the check provided by the configuration framework. The configuration framework does not, and does not need to, provide any assistance for the actual checks. It provides a callback definition for writing the custom configuration to the configuration file.

4.3.1. Writing a custom check

The user-defined file `build/conf-reqs.sh` can contain any custom checks and actions.

Below is a simple definition that checks whether Python is installed in the system and determines (at the same time) path to its binary executable.

```
function check_python_path () {
    # Print beginning of check message without a newline
    echo -n "checking for python... "

    # Find out where Python binary is located
    PYTHONPATH=`which python`

    # Handle situation where it is not found
    if [ ! $PYTHONPATH ] ; then
          echo "not found"
          exit 1
    fi

    # Display rest of the check message
    echo "$PYTHONPATH"
}
```

It is not really necessary to write the custom checks as functions, but writing them as such is a good practice and helps for possible modularization if there are a lot of checks.

The custom check can now be called, in `build/conf-reqs.sh`, just like checks defined in the framework:

```
check_python_path
```

Adding libraries

A configuration function can add extra libraries as well as any necessary include file and library paths with the following three functions:

```
add_include_dir "$SOMEDIR/include"
add_library_dir "$SOMEDIR/lib"
add_library     "somelib"
```

The above three definitions would add the following flags to C++ compiler:

```
-I $SOMEDIR/include -L $SOMEDIR/lib -lsomelib
```

(The `$SOMEDIR` variable in this example would be expanded in the configuration script.)

4.3.2. Writing custom configuration

After all checks have been done, the configuration system writes a configuration file `config.mk`. Custom configuration is written through a callback function, which appends lines to configuration file. The name of the configuration file, including path, is stored in the `$MKCONFIG` variable.

```
# Write custom configuration to configuration file
function my_custom_config () {
    echo "Writing custom config..."
    echo "export PYTHONPATH=$PYTHONPATH" >> $MKCONFIG
}
```

The name of the callback function must be passed to the configuration framework using the WRITE_CUSTOM_CONFIG variable.

```
# Inform the framework about this customization callback
WRITE_CUSTOM_CONFIG=my_custom_config
```

Chapter 5 Using the build system

This chapter describes how to use the makefile framework of the build system by creating an appropriate top-level and module makefiles.

5.1. Overview

The build system is used by defining makefiles that include the makefiles of the build system. The build framework requires various parameters defined by the configuration system. Using these parameters, it defines a number of other variables and, most importantly, makefile rules to build various targets.

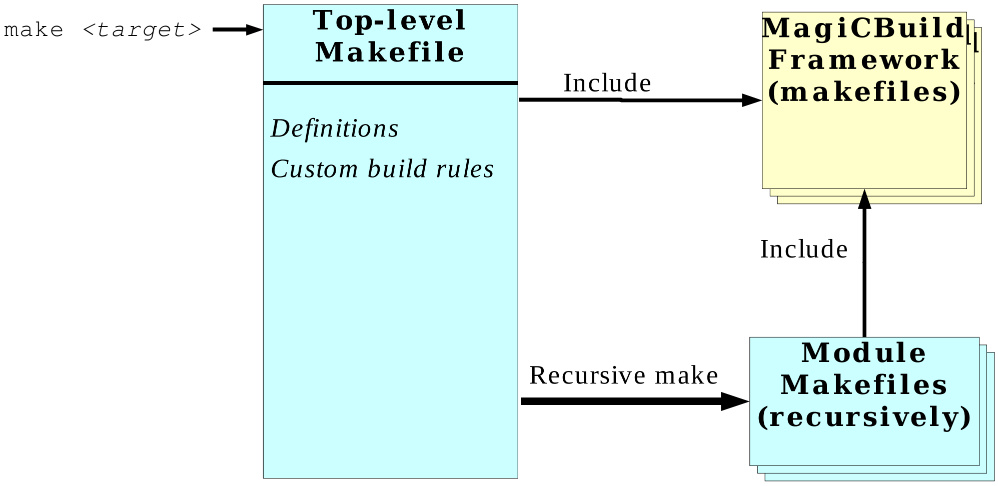

Figure 1: General makefile hierarchy

The various parameters and definitions required by the build system, as well as the structure of the top-level and module specific makefiles, are described in the subsequent sections.

An example of using the build system is provided in the `test` subdirectory of the MagiCBuild distribution package.

5.2. Build targets

MagiCBuild framework defines the following targets for Make:

Target	*Description*
`all`	Default target. Compiles everything recursively from source tree to output tree.
`deps`	Makes dependencies. You have to make dependencies if you intend to recompile something later and wish to ensure that all files are recompiled if the files they depend on have changed.
`clean`	Cleans output tree by deleting all intermediary and output files and directories. If the directories contain files from other projects, the directories are not removed.
`rebuild`	Cleans output tree and recompiles everything using the default target. Same as "`clean all`".
`dox`	Compiles reference documentation for C or C++ projects using Doxygen documentation generator.
`install`	Installs files from output directory tree to an installation directory hierarchy. For example, if the `$INSTALLDIR` (as defined with `--prefix` option for the `configure` script) is `/usr/local`, binaries will be copied to `/usr/local/bin`, libraries to `/usr/local/lib`, and shared files to `/usr/local/share`.
`uninstall`	Removes files that were installed using "`install`" target. The files are removed according to the `$INSTALLDIR` variable, as defined with `--prefix` option for the `configure` script. If the variable has changed after installing, the installed files will not be removed correctly.
`dist`	Builds a source code distribution package, which is a BZip2 compressed Tar archive.

It is possible to add custom targets and extend some of the predefined targets, as described in Section 5.5.2 *Custom targets* below.

5.3. Parameter variables

The build system requires the following parameters, which are usually defined by the configuration system. The build system attempts to read them from the `build/config.mk` configuration file, but they can also be given from the environment.

Variable	*Default*	*Description*
`SRCDIR`	`.` or `..`	Topmost source directory, under which the `build` subdirectory containing MagiCBuild files is located.
`BUILDDIR`	`/tmp/$USER/build`	Output directory where final executable files, object files, temporary files, and documentation files are written.
`PLATFORM`	`linux`	Operating system.
`ARCH`	`i386`	Processor architecture.

Variable	*Default*	*Description*
`INSTALLDIR`	`/usr/local` or `$HOME`	Base directory for installation. The default directory is `/usr/local` for system administrator (root) and `$HOME` for other users.
`BUILDTYPE`	`release`	Build type.
`CXX`	`g++`	C++ compiler to use.
`CXX_PATH`		GCC installation directory. The path is used to determine dependencies correctly. If it is not defined or is incorrect, generation of dependencies with "`make deps`" will generate many ugly (but mostly harmless) warnings.

Specialized requirement checks can add more parameters. For example, `check_for_qt` configuration requirement adds variable `QTDIR`, which points to the directory of the detected Qt installation.

Requirement checks that find out libraries can add elements to the following parameters:

Variable name	*Description*
`EXTRA_INCLUDE_DIRS`	Extra include directories for C++. The directories are listed as "`-I<`*`directory`*`>`" flags for the compiler.
`EXTRA_LIB_DIRS`	Extra library inclusion directories for C++. The directories are listed as "`-L<`*`directory`*`>`" flags for the linker.
`EXTRA_LIBS`	Extra libraries for C++. The libraries are listed as "`-l<`*`libname`*`>`" flags for the linker.

User-defined feature checks done in `build/conf-reqs.sh` can also add their own parameters.

5.4. Defined variables

MagiCBuild framework defines a set of variables for input and output directory paths. The definitions are done in `build/magicdef.mk`. and are based on the parameter variables.

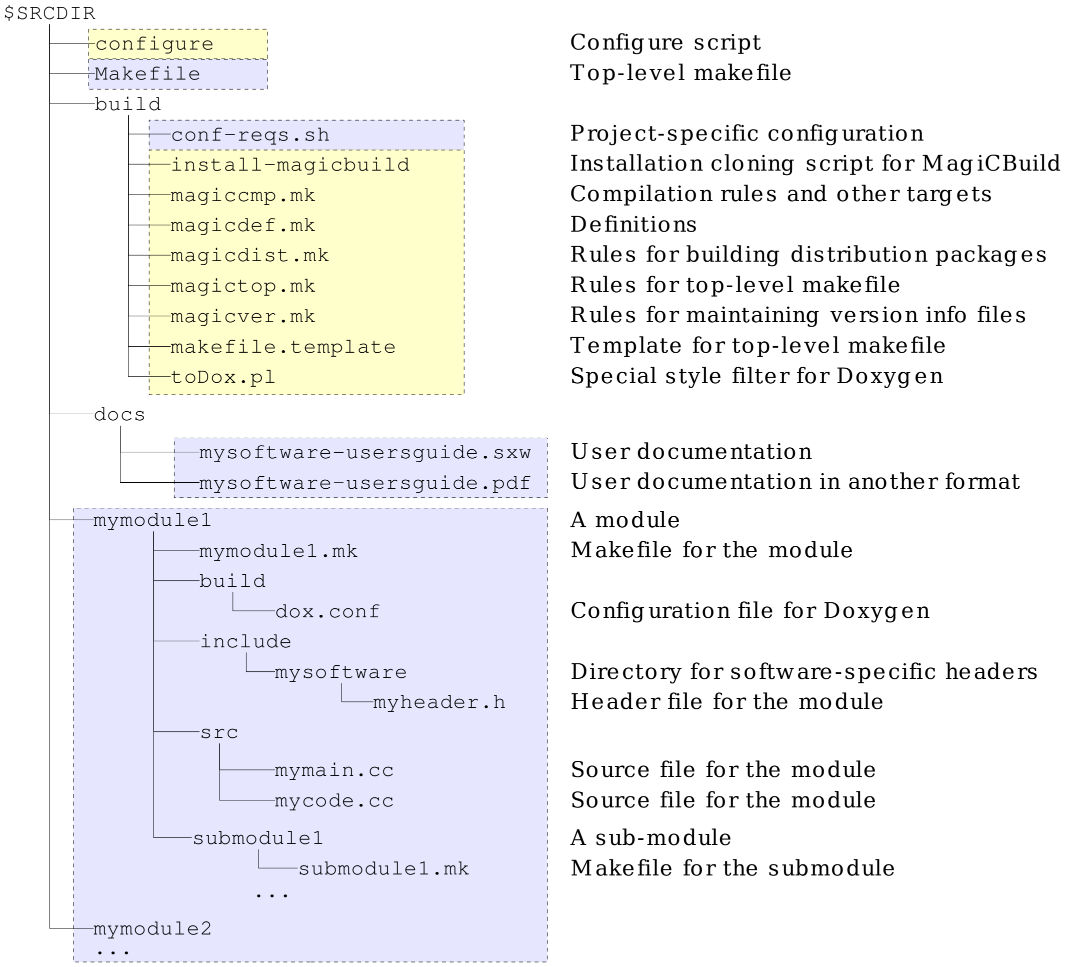
$SRCDIR
configure
Configure script
Makefile
Top-level makefile
build
conf-reqs.sh
Project-specific configuration
install-magicbuild
Installation cloning script for MagiCBuild
magiccmp.mk
Compilation rules and other targets
magicdef.mk
Definitions
magicdist.mk
Rules for building distribution packages
magictop.mk
Rules for top-level makefile
magicver.mk
Rules for maintaining version info files
makefile.template
Template for top-level makefile
toDox.pl
Special style filter for Doxygen
docs
mysoftware-usersguide.sxw
User documentation
mysoftware-usersguide.pdf
User documentation in another format
mymodule1
A module
mymodule1.mk
Makefile for the module
build
dox.conf
Configuration file for Doxygen
include
mysoftware
Directory for software-specific headers
myheader.h
Header file for the module
src
mymain.cc
Source file for the module
mycode.cc
Source file for the module
submodule1
A sub-module
submodule1.mk
Makefile for the submodule
...
mymodule2
...

Figure 2: Source directory hierarchy of a typical project

5.4.1. Source directories

The following variables define directories in the source tree.

Top-level source directories

The top-level source directories have "`grp-`" prefix.

Variable	*Default*	*Description*
`grpincdir`	`$SRCDIR/include`	Top-level include directory.
`grpbuilddir`	`$SRCDIR/build`	Top-level build directory that contains MagiCBuild files and also application-specific configuration and build files.
`grpdocdir`	`$SRCDIR/docs`	Top-level documentation directory.

Module level source directories

The module-level source directories have "`mod-`" prefix.

Variable	*Default*	*Description*
`moddir`	`$SRCDIR/$modpath`	Full path to module directory.
`modsrcdir`	`$moddir/src`	C and C++ source code
`modincdir`	`$moddir/include`	C and C++ headers. The header files are written to this common directory, or if `headersubdir` variable is defined in source modules, to a module or software specific subdirectory defined with the variable.
`modcfgdir`	`$moddir/config`	Configuration files to be installed
`moddatadir`	`$moddir/data`	Run-time data files to be installed
`modbuilddir`	`$moddir/build`	Additional build files

5.4.2. Output directories

All intermediate, temporary, and output files are written to a common output directory tree, which is totally separate from the source tree.

```
$outputdir
        ├───bin      ($bindir)                           Directory for compiled binaries
        │    └───myprogram                               A binary executable
        ├───lib      ($libdir)                           Directory for compiled libraries
        │    └───libmylibrary.a                          A library
        ├───include  ($incdir)                           Directory for compiled libraries
        │      └─mysoftware                              Header files for a specific software
        │            └───myheader.h                      A header file
        ├───dist     ($distdir)                          Source distribution packages
        │    └───mysoftware-0.1.1beta1.src.tar.bz2       A source distribution package
        ├───deps                                         Dependency files ("make deps")
        │    └───myprojectmymodule1-deps.mk              Dependencies for a module
        └───obj      ($objdir)                           Object files
             └───mymodule1                               Object files for a specific module
                        ├───mymain.o                     An object file
                        └───mysource.o                   An object file
```

Figure 3: Output directory hierarchy for a simple project

Common output directories

These directories contain files common to all projects.

Variable	*Default*	*Description*
`archdir`	`$BUILDDIR/$PLATFORM-$ARCH`	Operating system and processor architecture specific directory.
`outputdir`	`$archdir/$BUILDTYPE`	Output root directory
`libdir`	`$outputdir/lib`	Libraries
`bindir`	`$outputdir/bin`	Executable binaries
`objdir`	`$outputdir/obj`	Object files. Object files for each module and sub-module (application or library) are contained in respective subdirectories.

Variable	*Default*	*Description*
`mocdir`	`$outputdir/moc`	Meta-object compiler output files for Qt projects.
`incdir`	`$outputdir/include`	Shared headers
`sharedir`	`$outputdir/share`	Shared files to be installed, such as configuration files and data files.
`docdir`	`$outputdir/doc`	Documentation files to be installed. Documentation files for each module (application or library) are contained in respective subdirectories.
`tmpdir`	`$outputdir/tmp`	Temporary files
`distdir`	`$outputdir/dist`	Distribution packages created with "`make dist`".

Module specific output directories

The following variables specify module-specific output directories. The files are placed in module-specific directories to avoid name clashes.

Variable	*Default*	*Description*
`objmoddir`	`$objdir/$modpath`	Object files for a module
`docmoddir`	`$docdir/$modpath`	Documentation files for a module
`mocmoddir`	`$mocdir/$modpath`	Meta-object compiler output files for a module
`sharemoddir`	`$sharedir/apps/$modname`	Shared directory for a project or module. This is consistent with the Linux file hierarchy standard.
`cfgmoddir`	`$sharemoddir/config`	Configuration files for a project or module
`datamoddir`	`$sharemoddir/data`	Data files for a project or module

5.5. Recursive makefiles

All makefiles can be recursive to compile their sub-modules. The recursion is done by listing the submodules in the `makemodules` variable. The submodules are built after the `magictop.mk` or `magiccmp.mk` is called from a top-level or module-level makefile, respectively. These two types of recursive makefiles are detailed in sections 5.6 and 5.7 below.

The common variables defined for both types of recursive makefiles are:

Variable name	*Description*
`makemodules`	Space-separated list of modules (or sub-module) to be built recursively.
`extra_targets`	Extra make targets to be built for the default target.
`extra_dist_targets`	Extra make targets to be built for the `dist` target for making distribution packages. This is often used to compile document formats.

The general structure of recursive makefiles is illustrated in Figure 4 below.

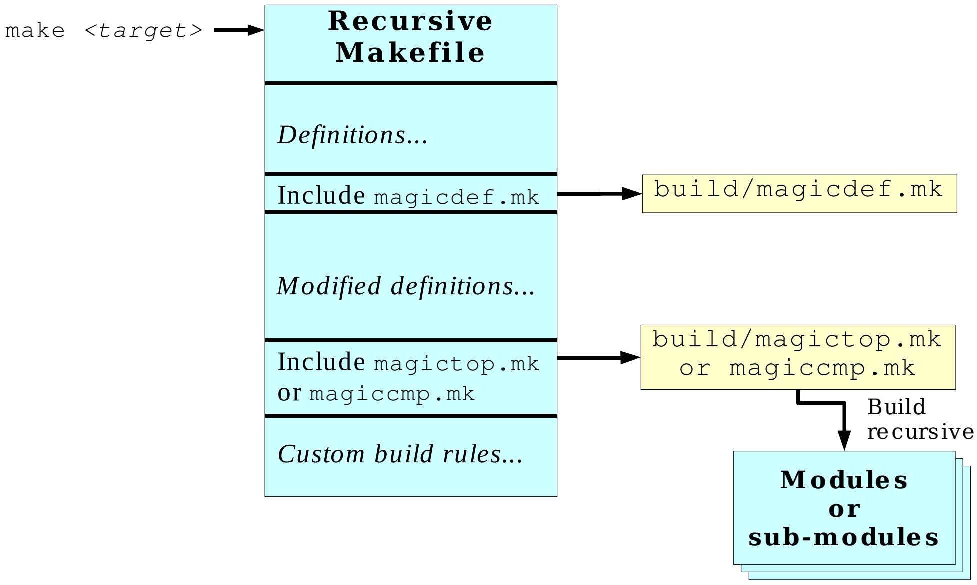

Figure 4: General structure of recursive makefiles

For example, the following top-level makefile builds two modules recursively:

```
...
include $(SRCDIR)/build/magicdef.mk

# Modules to build recursively
makemodules = module1 module2

#############################################################
# Include build rules
#############################################################
include $(SRCDIR)/build/magictop.mk
...
```

5.5.1. Definitions for building distribution packages

The following definitions are needed to build source distribution packages. The definitions list any "extra" files to be included in the package. Also all the MagiCBuild files must be listed (this may change in a future version).

Variable	***Description***
`distdirs`	Directories containing material to be included in the distribution package. In top-level makefiles, also the "`build`" directory containing MagiCBuild files *must* be included. In module-level makefiles, the `build` directory must be included only if it contains custom configuration files.

Variable	***Description***
`distfiles`	Files to be included in the distribution package. Full relative path to the files must be given, relative to the directory of the makefile. Intop-level makefile, also the "`configure`" script must be included in the list, as well as the "`Makefile`". In module-level makefiles, the makefile itself must be included. This list also typically contains a `README.TXT` file and any other documentation files placed in the `docs` subdirectory.
`buildfiles`	Files in the `build` subdirectory to be included in the distribution package. The list *must* include all MagiCBuild files for the top-level makefile. The user-defined `conf-reqs.sh` should also be included in this list, if it exists. In module-level makefiles, the list typically contains `dox.conf` configuration file for Doxygen documentation generator.

For example:

```
... basic definitions ...
##############################################################
# Distribution files and directories
##############################################################

# Extra directories to include in distribution package
distdirs   =         build docs

# Files to include in distribution package
distfiles  =         README.TXT configure Makefile \
                     docs/magicbuild-usersguide.sxw \

# Files in 'build' subdirectory, including custom ones
buildfiles =         conf-reqs.sh \
                     magicdef.mk magiccmp.mk magicdist.mk \
                     magictop.mk magicver.mk \
                     install-magicbuild toDox.pl \
                     makefile.template
...
```

5.5.2. Custom targets

Although the build system defines internally most necessary targets, it is often necessary to define custom targets, for example, if you need to compile software written in programming language not supported by MagiCBuild.

For example, the following top-level makefile builds two modules recursively:

```
... basic definitions ...
include $(SRCDIR)/build/magicdef.mk

# Any extra targets (see below)
extra_targets = hello

# Extra targets for building distribution package (see below)
extra_dist_targets = my_dist_target
```

```
#############################################################
# Include build rules
#############################################################
include $(SRCDIR)/build/magictop.mk

#############################################################
# Extra targets
#############################################################
hello:
        echo "Hello, world!"

# Do extra stuff for building distribution package
my_dist_target: $(distdir)/my_doc_file.pdf

$(distdir)/my_doc_file.pdf: $(docdir)/my_doc_file.ps
          pstopdf $< $@
```

The custom targets are called from `magictop.mk` (for top-level makefiles) or `magiccmp.mk` (for module-level makefiles).

5.6. Top-level makefile

The top-level makefile is a recursive makefile that is located at the top-level directory of the source directory tree. It is used to build all top-level modules of the software.

The top-level makefile is always named as "`Makefile`", to allow automatic recognition by the GNU Make without need to specify a file name. When MagiCBuild is installed to a software project with `install-magicbuild` program, the program creates a top-level makefile template as `Makefile.template`. You just need to rename this template as "`Makefile`" and modify it for your software.

Module-level makefiles have the following structure:

1. Define project parameters
2. Call `magicdef.mk`
3. Define recursive compilation parameters
4. Call `magictop.mk`
5. Define custom targets

The structure of a top-level makefile is illustrated in the Figure 5 below.

See Section 5.5 *Recursive makefiles* above for general information about recursive makefiles. Also the custom build rules are explained in that section.

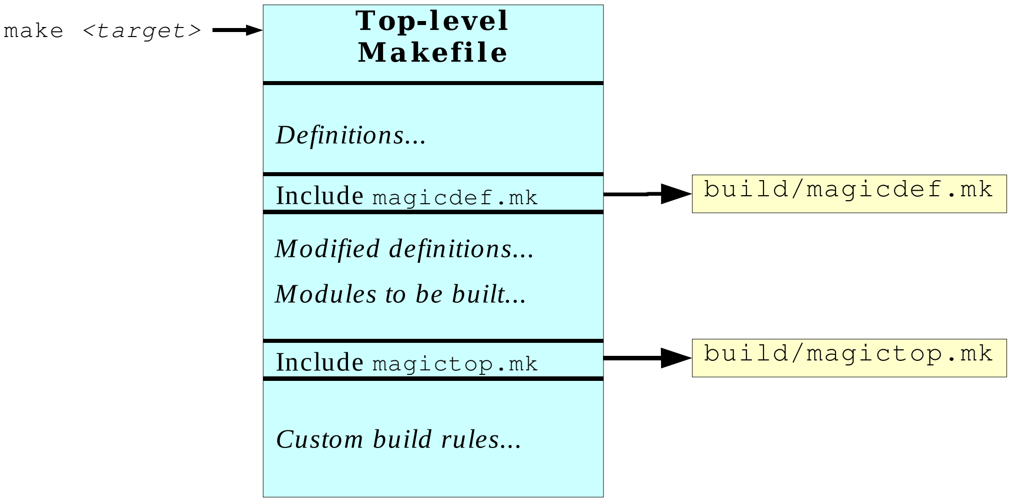

Figure 5: General structure of a top-level makefile

Basic definitions

The following definitions are necessary for the top-level makefile

```
##############################################################
# Define default root directory of the source tree
##############################################################
export SRCDIR ?= .

##############################################################
# Package and version info
##############################################################
export packagename = mypackage
export vermajor    = 0
export verminor    = 1
export verbuild    = 1
export versuffix   = beta
```

The package name and version numbers are used for building distribution packages.

Invoking build framework

After all the definitions above have been done, you can call the build framework. First, call `magicdef.mk` to make all the necessary definitions, and then `magictop.mk` to make the recursive build.

```
...
##############################################################
# Include build framework
##############################################################
include $(SRCDIR)/build/magicdef.mk

# Modules to build recursively
makemodules = module1 module2

##############################################################
```

```
# Include build rules
##############################################################
include $(SRCDIR)/build/magictop.mk
```

You can add your custom targets after this, as described in section.

5.7. Module-level makefiles

Module-level makefiles are very much like the top-level makefiles, except that they do not have the overall package definitions.

The name of a makefile of a module matches the module name. For example, if the module is located in subdirectory "`mymodule`", the makefile must be named "`mymodule.mk`". The binary executable or library compiled from the module will also have the same name by default.

Module-level makefiles have the following structure:

6. Define module parameters
7. Call `magicdef.mk`
8. Define compilation parameters
9. Call `magiccmp.mk`
10. Define custom targets

These are illustrated in the Figure 6 below.

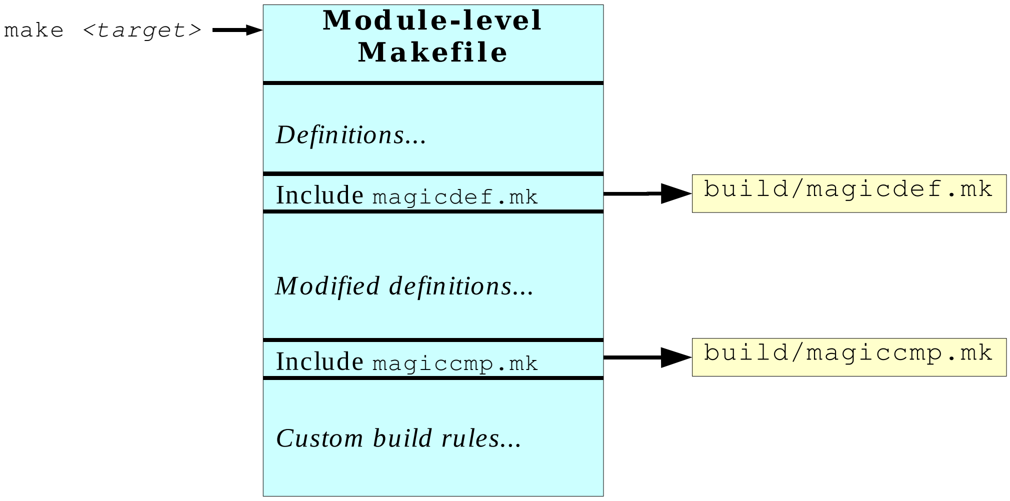

Figure 6: General structure of module-level makefiles.

The module parameters and compilation parameters are defined in subsections below.

See Section 5.5 *Recursive makefiles* above for general information about recursive makefiles. Also the custom build rules are explained in that section.

5.7.1. Module parameters

The following parameters can be defined in a module-level makefile.

Variable	*Default*	*Description*
`modname`	`-`	Name of the module (MANDATORY)
`modpath`	`$modname`	Path to the module directory, relative to the top-level source directory. This must be defined if the module is a second or lower level sub-module.
`modtarget`	`$modname`	Target basename for binaries and libraries.
`compile_library`	`0`	Is the module a library?
`makedox`	`0`	Should Doxygen documentation be generated? This requires having a `dox.conf` configuration file in the `build` subdirectory of the module.
`modversionfile`		Version info file for the module
`modversionheader`		Version C or C++ header file for the module

For example:

```
#############################################################
# Module parameters
#############################################################
modname   = mymodule
modpath   = somemodule/submodule/anotherlevel/mymodule
modtarget = mybinary

#############################################################
# Include build framework definitions
#############################################################
include $(SRCDIR)/build/magicdef.mk
```

In the above example, the "`mymodule`" is a 4th level submodule, so its path has to be defined with `modpath`.

5.7.2. Compilation parameters for C++

Compilation parameters are defined usually after calling `magicdef.mk`. The following definitions can be made:

Variable	*Description*
`sources`	List of source files in `src` subdirectory. The directory name should not be included in the names.
`headers`	List of header files in `include` subdirectory. The directory name should not be included in the names, unless the headers are contained in a subdirectory.
`libdeps`	List of library dependecies. The libraries are listed with their short name, for example, "`dl`" refers to the library `libdl`. The library names will be passed to the linker as '`-l<`*`name`*`>`", for example, "`-ldl`".

Variable	*Description*
`headersubdir`	Subdirectory of the `include` subdirectory containing the header files, making it possible to include the headers with "`#include <headersubdir/myheader.h>`". If the `headersubdir` is not given, all the headers will be written to a common include directory (such as `/usr/include`), which may result in file name clashes.

For example:

```
...
#############################################################
# Compilation parameters
#############################################################

# Source files in "src" subdirectory
sources = mymain.cc mysource.cc

# Header files in "include/mysoftware" subdirectory
headers = myheader.h

# Software-specific include directory
headersubdir = mysoftware

#############################################################
# Include build rules
#############################################################
include $(SRCDIR)/build/magiccmp.mk
```

Chapter 6 Known problems and limitations

6.1. Problems

MagiCBuild has the following known problems:

- The build system prints some shell execution errors with "`make`" command.
- The cleanup with "`make clean`" is incomplete in some cases.

6.2. Limitations

MagiCBuild has the following general limitations:

- Only C++ language supported
- MagiCBuild and its installation script can not be installed system-wide (to `/usr/bin`, etc.)

Configuration system

The configuration system, as defined in the `configure` script, has the following limitations:

- Options for the `configure` script are very limited. Especially the options for following features are missing:
  - Extra libraries and library and include file search paths
  - Enabling and disabling software components
  - Enabling debug build
  - Adding prefixes and suffixes to compiled binaries
- Configuration file (`config.mk`) is written to the source tree, not to the output tree, which would be nicer. As it is written to source tree, it is not possible to maintain multiple configurations simultaneously, for example, for debug and release builds. It would be possible to write it to the output tree, but the build system would not know where it was written. This could be solved by making the user define some environment variables, but we want to avoid that.

- Requiring a specific or minimum version number of required software is currently not supported for any of the predefined requirement checks
- C++ features are not checked
- A C++ header file should be generated to offer definitions to programs

Build system

- Dependencies are not determined internally, but they must be determined by running "`make deps`".
- All header files are currently copied to output directory and installed, regardless if they are private (such as those for compiling binary modules) or actually intended to be shared (such as those that provide an API to libraries).

Chapter 7 GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by thecopyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if thecopyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from theircopyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of theGNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address newproblems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Alphabetical Index