The DUNE Buildsystem HOWTO

Christian Engwer* Felix Albrecht†

March 1 2009

*Interdisziplinäres Zentrum für Wissenschaftliches Rechnen, Universität Heidelberg,
Im Neuenheimer Feld 368, D-69120 Heidelberg, Germany

†Institut für Numerische und Angewandte Mathematik, Westfälische Wilhelms-Universität Münster,
Einsteinstr. 62, D-48149 Münster, Germany

`http://www.dune-project.org/`

Contents

1 Getting started

TODO: How do I build the grid howto?

2 Creating your own DUNE module

This section tells you how to begin working with DUNE without explaining any further details. For a closer look on duneproject, see section 6.

Once you have downloaded all the DUNE modules you are interested in, you probably wonder "How do I start working with DUNE?" It is quite easy. Let us assume you have a terminal open and are inside a directory containing some DUNE modules. Let us say

```
ls -l
```

produces something like:

```
dune-common/
dune-grid/
config.opts
```

There is no difference between a DUNE module you have downloaded from the web and modules you created yourself. dunecontrol takes care of configuring your project and creating the correct Makefiles (so you can easily link and use all the other DUNE modules). It can be done by calling

```
./dune-common/bin/duneproject
```

Note: In case you are using the unstable version DUNE you should be aware that the buildsystem may change, just like the source code. Therefore it might be that duneproject is not up to date with the latest changes.

After calling duneproject, you have to provide a name for your project (without whitespace), e.g., dune-foo. The prefix dune- is considered good practice, but it is not mandatory. You are then asked to provide a list of all modules the new project should depend on (this will be something like dune-common dune-grid, etc.). At last, you should provide the version of your project (e.g., 0.1) and your email address. duneproject now creates your new project which is a folder with the name of your project, containing some files needed in order to work with DUNE. In our example,

```
ls -l dune-foo/
```

should produce something like

```
configure.ac
dune.module
dune-foo.cc
Makefile.am
README
```

You can now call dunecontrol for your new project, as you would for any other DUNE module. If you have a config.opts file configured to your needs (see e.g. the "Installation Notes" on www.dune-project.org), a simple call of

```
./dune-common/bin/dunecontrol --module=dune-foo --opts=config.opts all
```

should call dune-autogen, configure and make for your project and all modules your project depends on first.

Remark 2.1 *Always call* `dunecontrol` *from the directory containing* `dune-common`*.*

You can now simply run

```
./dune-foo/dune-foo
```

which should produce something like

```
Hello World! This is dune-foo.
This is a sequential program.
```

If you want your DUNE module to be useable by other people your design should follow a certain structure. A good way to indicate that your module is set up like the other DUNE modules is by naming it with the prefix `dune-`. Since your module should be concerned with a certain topic, you should give it a meaningful name (e.g. `dune-grid` is about grids).

Our next step is to create the subfolders `doc/`, `foo/` and `src/` in `dune-foo/`.
`foo/` will contain any headers that are of interest to other users (like the subfolder `common/` in `dune-common`, `grid/` in `dune-grid`, etc.). Other users will have to include those files if they want to work with them. Let's say we already have some interface implementation in a file `bar.hh`. We put this one into the subfolder `foo/`.
It is then convenient to collect whatever documentation exists about those header files in `doc/` and since there should at least exist some doxygen documentation we create the subdirectory `doc/doxygen/` (see "Coding Style" in the setion "Developing Dune" on `www.dune-project.org` for details). We will need a `Makefile.am` in `doc/` and `doc/doxygen/` so we copy the `Makefile.am` from `dune-foo/Makefile.am` to `dune-foo/doc/Makefile.am`. Since we also need some other files for `doxygen` to work we just copy the following files from e.g. `dune-grid/doc/doxygen/` to `dune-foo/doc/doxygen/`:

```
Doxydep
Doxyfile
doxy-footer.html
doxy-header.wml
dune-doxy.css
mainpage
Makefile.am
modules
```

The `src/` subdirectory will contain the sources of your implementation (usually at least one `.cc` file with a `main` method). Since we already have such a file (`dune_foo.cc`), we move it to `src/`. We also need a `Makefile.am` in `src/` so we copy our `Makefile.am` from `dune-foo/Makefile.am` to `dune-foo/src/Makefile.am`.
Of course we will have to edit those copies later on. But let's take a look at the structure of our project now.

```
dune-foo/
-> configure.ac
-> doc/
   -> doxygen/
      -> Doxydep
      -> Doxyfile
      -> doxy-footer.html
      -> doxy-header.wml
      -> dune-doxy.css
      -> mainpage
      -> Makefile.am
      -> modules
   -> Makefile.am
-> dune.module
```

```
-> foo/
   -> bar.hh
-> Makefile.am
-> README
-> src/
   -> dune_foo.cc
```

Now all that's left to do is to edit those `Makefile.am`s, the `configure.ac` and some `doxygen` files. First we open the file `dune-foo/Makefile.am`. Ignoring comments, you should edit the file as follows:

```
1  EXTRA_DIST=dune.module
2
3  DIST_SUBDIRS = doc src
4
5  if BUILD_DOCS
6  SUBDIRS = doc src
7  else
8  SUBDIRS = src
9  endif
10
11 AUTOMAKE_OPTIONS = foreign 1.5
12
13 DISTCHECK_CONFIGURE_FLAGS = --with-dune=$(DUNEROOT) CXX="$(CXX)" CC="$(CC)"
14
15 include $(top_srcdir)/am/global-rules
```

Lines $5-9$ state that there are two relevant subdirectories (in `doc/` and `src/`) if your module is being configured without the `--disable-documentation` flag or just in `src/` otherwise. Line 13 might look different on your machine, and you should use the line from the original `Makefile.am` in that case.
We can leave `dune-foo/src/Makefile.am` nearly as it is, only the line

```
EXTRA_DIST=dune.module
```

should be removed.

Now we have to tell `dune-autogen` something about the structure of our project. This can easily be done by opening `dune-foo/configure.ac` and editing just two lines.
Since we moved `dune_foo.cc` into `src/` we have to tell `configure` where it is now. Therefore we change the line

```
AC_CONFIG_SRCDIR([dune_foo.cc])
```

to

```
AC_CONFIG_SRCDIR([src/dune_foo.cc])
```

Also we have to tell `configure` about all the `Makefile`s we need created. We do this by editing the line

```
AC_CONFIG_FILES([Makefile])
```

For our example, we replace the above line with the following:

```
AC_CONFIG_FILES([Makefile
    src/Makefile
    doc/Makefile
    doc/doxygen/Makefile])
```

Now your module is nearly ready to being configured by `dunecontrol`. Only the `doxygen` part is missing (in fact, configuring it with the `--disable-documentation` flag would work from this point

on).

To configure doxygen we have to edit two Makefile.ams and some doxygen files. The first Makefile.am to edit is of course dune-foo/doc/Makefile.am. Ignoring comments again, the file should look like:

```
1  SUBDIRS = doxygen
2
3  all: $(PAGES)
4
5  CURDIR=doc
6
7  BASEDIR=..
8
9  docdir=$(datadir)/doc/dune-foo
10
11 include $(top_srcdir)/am/webstuff
12
13 CLEANFILES = $(PAGES)
14
15 if ! BUILD_DOCS
16 dist-hook:
17     echo "# No documentation included in distribution! " > $(distdir)/$(DOCUMENTATION_TAG_FILE)
18 endif
19
20 include $(top_srcdir)/am/global-rules
```

Now we can take a look at Makefile.am in dune-foo/doc/doxygen/. Since this one was copied from dune-grid it should suffice to change all occurrences of grid into foo. In DUNE release 1.2 this should be just the lines

```
doxygendir = $(datadir)/doc/dune-grid/doxygen
```

and

```
EXTRAINSTALL="$(DOXYGENINSTALL)" CURDIR="$(CURDIR)/dune-grid-html" install ; \
```

which have to be changed to

```
doxygendir = $(datadir)/doc/dune-foo/doxygen
```

and

```
EXTRAINSTALL="$(DOXYGENINSTALL)" CURDIR="$(CURDIR)/dune-foo-html" install ; \
```

respectively.
Now dunecontrol is ready to take care of building the documentation. But since we copied some doxygen stuff from dune-grid there are some doxygen files left we have to take care of. These are Doxyfile, mainpage and modules. You can edit the latter two as you like. They will form the main page and the module page of the html documentation of doxygen.
We are basically left to change all occurrences of grid in the Doxyfile as well as some settings. If you open the file dune-foo/doc/doxygen/Doxyfile the first thing you should see is a line like

```
PROJECT_NAME = dune-grid
```

which should obviously be changed into

```
PROJECT_NAME = dune-foo
```

to state the name of your module.

The next thing we're interested in are the lines

```
INPUT = mainpage \
        modules \
        ../../grid/modules \
        ../../grid
```

which should of course look like

```
INPUT = mainpage \
        modules \
        ../../foo
```

Then there are some settings of `dune-grid` left which we probably don't want to have. So we just comment all the following lines by just adding a `#` at the beginning of every line.

```
EXCLUDE = ../../grid/onedgrid \
          ../../grid/uggrid \
          ../../grid/test

EXAMPLE_PATH = ../../grid/io/file/dgfparser/test

IMAGE_PATH = ../../grid/sgrid \
             ../../grid/yaspgrid \
             ../../grid/common \
             ../../grid/io/file/dgfparser/test \
             ../appl/refelements \
             ../refinement
```

Now we are done editing files and ready to configure the module without the option `--disable-documentation` (provided you have `doxygen` and the necessary tools installed on your system).
After running

```
./dune-common/bin/dunecontrol --module=foo --opts=config.opts all
```

with a `config.opts` that enables documentation you should now find a `html doxygen` documentation in `dune-foo/doc/doxygen/html/index.html`.

If you take a look at the `DUNE` core modules you will find a symlink

```
dune -> .
```

in each main folder. Its purpose is to allow inclusion directives like

```
#include <dune/foo/bar.hh>
```

If you want header files from your module to be includable from other modules you should have this link. You can set it manually by goint to your module directory `dune-foo/` and typing

```
ln -s . dune
```

You can also let `configure` do this for you automatically if you include this line

```
DUNE_SYMLINK
```

in your `configure.ac` file.

Remark 2.2 *This mechanism is under discussion and may change in a future version of DUNE.*

3 The Structure of DUNE

DUNE consists of several independent modules:

- dune-common
- dune-grid
- dune-istl
- dune-grid-howto
- dune-grid-dev-howto

Single modules can depend on other modules and so the DUNE modules form a dependency graph. The build system has to track and resolve these inter-module dependencies.

The build system is structured as follows:

- Each module is built using the GNU AutoTools.
- Each module has a set of modules it depends on, these modules have to be built before building the module itself.
- Each module has a file dune.module which holds dependencies and other information regarding the module.
- The modules can be built in the appropriate order using the dunecontrol script (shipped with dune-common)

The reasons to use the GNU AutoTools for DUNE were the following

- We need platform independent build.
- Enabling or disabling of certain features depending on features present on the system.
- Creations of libraries on all platforms.
- Easy creation of portable but flexible Makefiles.

The reasons to add the dunecontrol script and the dune.module description files were

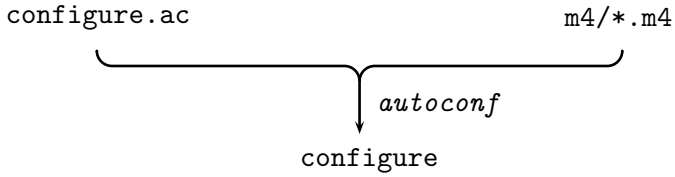
- One tool to setup all modules (the AutoTools can only work on one module).
- Automatic dependency tracking.
- Automatic collection of command-line parameters (configure needs special command-line parameters for all modules it uses)

4 Building Single Modules Using the GNU AutoTools

Software is generally developed to be used on multiple platforms. Since each of these platforms has different compilers, different header files, there is a need to write makefiles and build scripts that work on a variety of platforms. The Free Software Foundation (FSF), faced with this problem, devised a set of tools to generate makefiles and build scripts that work on a variety of platforms. These are the GNU AutoTools. If you have downloaded and built any GNU software from source, you are familiar with the `configure` script. The `configure` script runs a series of tests to get information about your machine.

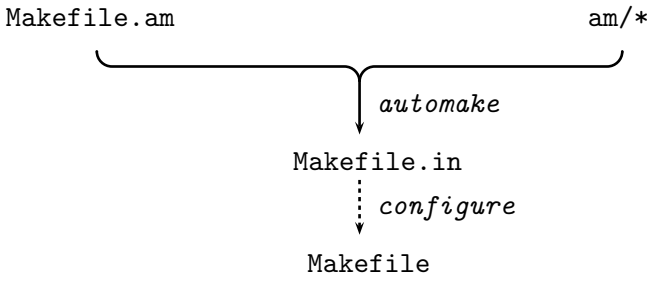
The autotools simplify the generation of portable Makefiles and configure scripts.

autoconf
`autoconf` is used to create the `configure` script. `configure` is created from `configure.ac`, using a set of `m4` files.



How to write a `configure.ac` for `DUNE` is described in Sec. 4.2.

automake
`automake` is used to create the `Makefile.in` files (needed for `configure`) from `Makefile.am` files, using a set of include files located in a directory called `am`. These include files provide additional features not provided by the standard `automake` (see Sec. 4.1.2). The `am` directory is in the `dune-common` module and each module intending to use one of these includes has to have a symlink `am` that points to `dune-common/am`. This link is usually created by `dune-autogen` (see Sec. 4.3).



Information on writing a `Makefile.am` is described in 4.1

libtool
`libtool` is a wrapper around the compiler and linker. It offers a generic interface for creating static and shared libraries, regardless of the platform it is running on.

`libtool` hides all the platform specific aspects of library creation and library usage. When linking a library or an executable you (or `automake`) can call the compiler via `libtool`. `libtool` will then take care of

- platform specific command-line parameters for the linker,
- library dependencies.

configure
`configure` will run the set of tests specified in your `configure.ac`. Using the results of these tests configure can check that all necessary features (libraries, programs, etc.) are present and can activate and deactivate certain features of the module depending on what is available on your system.

For example `configure` in `dune-grid` will search for the ALUGrid library and enable or disable `Dune::ALU3dGrid`. This is done by writing a preprocessor macro `#define HAVE_ALUGRID` in the `config.h` header file. A header file can then use an `#ifdef` statement to disable parts of the code that do not work without a certain feature. This can be used in the applications as well as in the headers of a `DUNE` module.

The `config.h` file is created by `configure` from a `config.h.in` file, which is automatically created from the list of tests used in the `configure.ac`.

4.1 Makefile.am

4.1.1 Overview

Let's start off with a simple program *hello* built from `hello.c`. As `automake` is designed to build and install a package it needs to know

- what programs it should build,
- where to put them when installing,
- which sources to use.

The core of a `Makefile.am` thus looks like this:

```
noinst_PROGRAMS = hello
hello_SOURCES = hello.c
```

This would build *hello* but not install it when `make install` is called. Using `bin_PROGRAMS` instead of `noinst_PROGRAMS` would install the *hello*-binary into a *`prefix`*`/bin` directory.

Building more programs with several source files works like this

```
noinst_PROGRAMS = hello bye

hello_SOURCES = common.c common.h hello.c
bye_SOURCES = common.c common.h bye.c parser.y lexer.l
```

`automake` has more integrated rules than the standard make, the example above would automatically use yacc/lex to create `parser.c/lexer.c` and build them into the *bye* binary.

Make-Variables may be defined and used as usual:

```
noinst_PROGRAMS = hello bye

COMMON = common.c common.h

hello_SOURCES = $(COMMON) hello.c
bye_SOURCES = $(COMMON) bye.c parser.y lexer.l
```

Even normal make-rules may be used in a `Makefile.am`.

Using flags

Compiler/linker/preprocessor-flags can be set either globally:

```
noinst_PROGRAMS = hello bye

AM_CPPFLAGS = -DDEBUG

hello_SOURCES = hello.c
bye_SOURCES = bye.c
```

or locally:

```
noinst_PROGRAMS = hello bye

hello_SOURCES = hello.c
hello_CPPFLAGS = -DHELLO

bye_SOURCES = bye.c
bye_CPPFLAGS = -DBYE
```

The local setting overrides the global one, thus

```
hello_CPPFLAGS = $(AM_CPPFLAGS) -Dmyflags
```

may be a good idea.

It is even possible to compile the same sources with different flags:

```
noinst_PROGRAMS = hello bye

hello_SOURCES = generic-greeting.c
hello_CPPFLAGS = -DHELLO

bye_SOURCES = generic-greeting.c
bye_CPPFLAGS = -DBYE
```

Perhaps you're wondering why the above examples used `AM_CPPFLAGS` instead of the normal `CPPFLAGS`? The reason for this is that the variables `CFLAGS`, `CPPFLAGS`, `CXXFLAGS` etc. are considered *user variables* which may be set on the commandline:

```
make CXXFLAGS="-O2000"
```

This would override any settings in Makefile.am which might be necessary to build. Thus, if the variables should be set even if the user wishes to modify the values, you should use the `AM_*` version.

The real compile-command always uses both `AM_`*`VAR`* and *`VAR`*. Options that autoconf finds are stored in the user variables (so that they may be overridden)

Commonly used variables are:

- `AM_CPPFLAGS`: flags for the C-Preprocessor. This includes preprocessor defines like `-DNDEBUG` and include pathes like `-I/usr/local/package/include`
- `AM_CFLAGS`, `AM_CXXFLAGS`: flags for the compiler (-g, -O, ...). One difference between these and the `CPPFLAGS` is that the linker will get `CFLAGS`/`CXXFLAGS` and `LDFLAGS` but not `CPPFLAGS`
- `AM_LDFLAGS` options for the linker
- `LDADD`: libraries to link to a binary
- `LIBADD`: libraries to add to a library

- `SOURCES`: list of source-files (may include headers as well)

Conditional builds

Some parts of `DUNE` only make sense if certain addon-packages were found. autoconf therefore defines *conditionals* which automake can use:

```
if OPENGL
  PROGS = hello glhello
else
  PROGS = hello
endif

hello_SOURCES = hello.c

glhello_SOURCES = glhello.c hello.c
```

This will only build the *glhello* program if OpenGL was found. An important feature of these conditionals is that they work with any make program, even those without a native *if* construct like GNU-make.

Default targets

An automake-generated Makefile does not only know the usual *all*, *clean* and *install* targets but also

- **tags** travel recursively through the directories and create TAGS-files which can be used in many editors to quickly find where symbols/functions are defined (use emacs-format)
- **ctags** the same as "tags" but uses the vi-format for the tags-files
- **dist** create a distribution tarball
- **distcheck** create a tarball and do a test-build if it really works

4.1.2 Building Documentation

If you want to build documentation you might need additional make rules. `DUNE` offers a set of predefined rules to create certain kinds of documentation. Therefor you have to include the appropriate rules from the `am/` directory. These rules are stored in the `dune-common/am/` directory. If you want to use these any of these rules in your `DUNE` module or application you will have to create a symbolic link to `dune-common/am/`. The creation of this link should be done by the `dune-autogen` script.

html pages

Webpages are created from wml sources, using the program `wml` (`http://thewml.org/`).
`$(top_srcdir)/am/webstuff` contains the necessary rules.

Listing 1 (File Makefile.am)

```
# $Id: Makefile.am 5388 2008-12-03 09:51:16Z sander $

# also build these sub directories
SUBDIRS = devel doxygen layout buildsystem

# only build html pages, if documentation is enabled
if BUILD_DOCS
# only build html when wml is available
if WML
 PAGES = view-concept.html installation-notes.html
endif
endif

# automatically create these web pages
all: $(PAGES)

# setting like in dune-web
CURDIR=doc
# position of the web base directory,
# relative to $(CURDIR)
BASEDIR=..
EXTRAINSTALL=example.opts

# install the html pages
docdir=$(datadir)/doc/dune-common
doc_DATA = $(PAGES) example.opts

EXTRA_DIST = $(PAGES) example.opts

if ! BUILD_DOCS
# add tag to notify that dist has been build without documentation
dist-hook:
    echo "# No documentation included in distribution! " > $(distdir)/$(DOCUMENTATION_TAG_FILE)
endif


# include rules for wml -> html transformation
include $(top_srcdir)/am/webstuff

# remove html pages on ``make clean''
SVNCLEANFILES = $(PAGES)
clean-local:
    if test -e $(top_srcdir)/doc/doxygen/Doxydep; then rm -rf $(SVNCLEANFILES); fi

# include further rules needed by Dune
include $(top_srcdir)/am/global-rules
```

LaTeXdocuments

In order to compile LaTeXdocuments you can include `$(top_srcdir)/am/latex`. This way you get rules for creation of DVI files, PS files and PDF files.

SVG graphics

SVG graphics can be converted to png, in order to include them into the web page. This conversion can be done using inkscape (`http://www.inkscape.org/`). `$(top_srcdir)/am/inkscape.am` offers the necessary rules.

4.2 configure.ac

`configure.ac` is a normal text file that contains several `autoconf` macros. These macros are evaluated my the `m4` macro processor and transformed into a shell script.

Listing 2 (File dune-common/configure.ac)

```
#! /bin/bash
# $Id: configure.ac 5403 2009-01-21 10:11:40Z sander $
# Process this file with autoconf to produce a configure script.

DUNE_AC_INIT # gets module version from dune.module file
AM_INIT_AUTOMAKE
AC_CONFIG_SRCDIR([common/stdstreams.cc])
AM_CONFIG_HEADER([config.h])

# create symlink dune -> $top_srcdir
DUNE_SYMLINK
# add configure flags needed to create log files for dune-autobuild
DUNE_AUTOBUILD_FLAGS
# check all dune dependecies and prerequisits
DUNE_CHECK_ALL

# preset variable to path such that #include <dune/...> works
AC_SUBST([DUNE_COMMON_ROOT], '$(top_builddir)')
AC_SUBST([AM_CPPFLAGS], '-I$(top_srcdir)')
AC_SUBST([LOCAL_LIBS], '$(top_builddir)/common/libcommon.la')

# write output
AC_CONFIG_FILES([Makefile
     lib/Makefile
     bin/Makefile
     common/Makefile
     common/test/Makefile
     common/exprtmpl/Makefile
     doc/Makefile
     doc/devel/Makefile
     doc/layout/Makefile
     doc/doxygen/Makefile
     doc/doxygen/Doxyfile
     doc/buildsystem/Makefile
     m4/Makefile
     am/Makefile
     bin/wmlwrap
     bin/check-log-store
     dune-common.pc])
AC_OUTPUT

# make scripts executable
chmod +x bin/wmlwrap
chmod +x bin/check-log-store

# print results
DUNE_SUMMARY_ALL
```

We offer a set of macros that can be used in your `configure.ac`:

- `DUNE_CHECK_ALL` runs all checks usually needed by a *DUNE module*. It checks for all dependencies and suggestions and for their prerequisites. In order to make the dependencies known to `configure` `dune-autogen` calls `dunecontrol m4create` and write a file `dependencies.m4`.

- DUNE_SYMLINK creates symlink $(top_srcdir)/dune → $(top_srcdir). The programming guidelines (7) require that the include statements be like #include <dune/...>. If your module has a directory structure $(top_srcdir)/foo, you will need such a link. However, you are encouraged to store the files directly in a directory structure $(top_srcdir)/dune/foo in order to avoid any inconvenience when copying the files. This will also eliminate the necessity for DUNE_SYMLINK.
- DUNE_AUTOBUILD_FLAGS adds configure flags needed to create log files for dune-autobuild. If you want to add your module to the dune-autobuild system, you have to call this macro.
- DUNE_SUMMARY_ALL prints information on the results of all major checks run by DUNE_CHECK_ALL.

DUNE_CHECK_ALL and DUNE_CHECK_ALL_M define certain variables that can be used in the configure script or in the Makefile.am:

- DUNE *MODULE* _CPPFLAGS
- DUNE *MODULE* _LDFLAGS
- DUNE *MODULE* _LIBS
- DUNE *MODULE* ROOT

The last step to a complete configure.ac is that you tell autoconf which files should be generated by configure. Therefor you add an AC_CONFIG_FILES([*WhiteSpaceSeparatedListOfFiles*]) statement to your configure.ac. The list of files should be the list of files that are to be generated, not the input – i.e. you would write

```
AC_CONFIG_FILES([Makefile doc/Makefile])
```

end not

```
AC_CONFIG_FILES([Makefile.in doc/Makefile.in])
```

After you told autoconf which files to create you have to actually trigger their creation with command AC_OUTPUT

4.3 dune-autogen

The dune-autogen script is used to bring the freshly checked out module into that state that you expect from a module received via the tarball. That means it runs all necessary steps so that you can call configure to setup your module. In the case of DUNE this means that dune-autogen runs

- libtoolize (prepare the module for libtool)
- dunecontrol m4create (create an m4 file containing the dependencies of this module)
- aclocal (collect all autoconf macros needed for this module)
- autoheader (create the config.h.in)
- automake (create the Makefile.in)
- autoconf (create configure)

If needed it will also create the symbolic link to the dune-common/am/ directory (see 4.1.2).

4.4 m4 files

m4 files contain macros which are then composed into configure and are run during execution of configure.

private m4 macros
You can add new tests to configure by providing additional macro files in the directory module/m4/.

dependencies.m4
$(top_srcdir)/dependencies.m4 hold all information about the dependencies and suggestions of this module. It is an automatically generated file. It is generated by dunecontrol m4create.

For each dependencies of your module *MODULE*_CHECKS and *MODULE*_CHECK_MODULE is called. Last *MODULE*_CHECKS is called for your module, in order to check all prerequisites for your module.

What you just read implies that you have to provide the two macros *MODULE*_CHECKS and *MODULE*_CHECK_MODULE for your module. These should be written to a m4/*.m4 file.

Here follows an example for the module dune-foo:

```
AC_DEFUN([DUNE_FOO_CHECKS])
AC_DEFUN([DUNE_FOO_CHECK_MODULE],[
  DUNE_CHECK_MODULES([dune-foo],        dnl module name
                     [foo/foo.hh],      dnl header file
                     [Dune::FooFnkt])   dnl symbol in libdunefoo
])
```

The first one calls all checks required to make use of dune-foo. The dependency checks are not to be included, they are run automatically. The second macro tells how to check for your module. In case you are only writing an application and don't want to make this module available to other modules, you can just leave it empty. If you have to provide some way to find your module. The easiest is to use the DUNE_CHECK_MODULES macro, which is defined in dune-common/m4/dune.m4.

5 Building Sets of Modules Using dunecontrol

dunecontrol helps you building the different DUNE modules in the appropriate order. Each module has a dune.module file which contains information on the module needed by dunecontrol.

dunecontrol searches for dune.module files recursively from where you are executing the program. For each DUNE module found it will execute a dunecontrol command. All commands offered by dunecontrol have a default implementation. This default implementation can be overwritten and extended in the dune.module file.

The commands you are interested in right now are

- autogen runs dune-autogen for each module. A list of directories containing dune.module files and the parameters given on the commandline are passed as parameters to dune-autogen.
- configure runs configure for each module. --with-dune*module* parameters are created for a set of known DUNE modules.
- make runs make for each module.
- all runs dune-autogen, configure and make for each module.

In order to build DUNE the first time you will need the `all` command. In pseudo code `all` does the following:

```
foreach ($module in $Modules) {
  foreach (command in {autogen,configure,make) {
    run $command in $module
  }
}
```

This differs from calling

```
dunecontrol autogen
dunecontrol configure
dunecontrol make
```

as it ensures that i.e. `dune-common` is fully built before `configure` is executed in `dune-grid`. Otherwise `configure` in `dune-grid` would complain that `libcommon.la` from `dune-common` is missing.

Further more you can add parameters to the commands; these parameters get passed on to the program being executed. Assuming you want to call `make clean` in all DUNE modules you can execute

```
dunecontrol make clean
```

opts files

You can also let `dunecontrol` read the command parameters from a file. For each command you can specify parameters. The parameters are stored in a variable called *COMMAND* `_FLAGS` with *COMMAND* written in capital letters.

Listing 3 (File example.opts)

```
# use these options for configure if no options a provided on the cmdline
AUTOGEN_FLAGS="--ac=2.50␣--am=-1.8"
CONFIGURE_FLAGS="CXX=g++-3.4␣--prefix='/tmp/Hu␣Hu'"
MAKE_FLAGS=install
```

When you specify an opts file and command line parameters

```
dunecontrol --opts=some.opts configure --with-foo=bar
```

`dunecontrol` will ignore the parameters specified in the opts file and you will get a warning.

environment variables

You can further control the behavior of `dunecontrol` by certain environment variables.

- `DUNE_CONTROL_PATH` specifies the paths, where `dunecontrol` is searching for modules. All entries have to be colon separated and should point to either a directory (which is search recursively for `dune.module` files) or a directly `dune.module` file.
- `DUNE_OPTS_FILE` specifies the opts file that should be read by `dunecontrol`. This variable will be overwritten by the `--opts=` option.
- `MAKE` tells `dunecontrol` which command to invoke for 'make'. This can be useful for example, if you want to use *gmake* as a make drop-in.
- `GREP` tells `dunecontrol` which command to invoke for 'grep'.

dune.module

The `dune.module` file is split into two parts. First we have the parameter section where you specify parameters describing the module. Then we have the command section where you can overload the default implementation of a command called via `dunecontrol`.

Listing 4 (File dune.module)

```
# parameters for dune control
Module: dune_grid
Depends: dune_common
Suggests: UG Alberta Alu3d

# overload the run_configure command
run_configure () {
  # lets extend the parameter list $CMD_FLAGS
  if test "x$HAVE_UG" == "xyes"; then
    CMD_FLAGS="$CMD_FLAGS \"--with-ug=$PATH_UG\""
  fi
  if test "x$HAVE_Alberta" == "xyes"; then
    CMD_FLAGS="$CMD_FLAGS \"--with-alberta=$PATH_Alberta\""
  fi
  if test "x$HAVE_Alu3d" == "xyes"; then
    CMD_FLAGS="$CMD_FLAGS \"--with-alugrid=$PATH_Alu3d\""
  fi
  # call the default implementation
  run_default_configure
}
```

The parameter section will be parsed by `dunecontrol` will effect i.e. the order in which the modules are built. The parameters and their values are separated by colon. Possible parameters are

- `Module` (*required*) is the name of the module. The name is of the form `[a-zA-Z0-9_]+`.
- `Depends` (*required*) takes a space separated list of required modules. This module is not functional without these other modules.
- `Suggests` (*optional*) takes a space separated list of optional modules. This module is functional without these other modules, but can offer further functionality if one or more of the suggested modules are found.

The command section lets you overload the default implementation provided by `dunecontrol`. For each command `dunecontrol` call the function `run_`*`command`*. The parameters from the commandline or the opts file are store in the variable `$CMD_FLAGS`. If you just want to create additional parameters you can add these to `$CMD_FLAGS` and then call the default implementation of the command via `run_default_`*`command`*.

6 Creating a new DUNE project

From a buildsystem point of view there is no difference between a DUNE application and a DUNE module.

DUNE modules are packages that offer a certain functionality that can be used by DUNE applications. Therefore DUNE modules offer libraries and/or header files. A DUNE module needs to comply with certain rules (see 7).

Creating a new DUNE project has been covered in detail in 2 using duneproject to take work off of the user. This is also the recommended way to start a new project. If for whatever reasons you do not wish to use duneproject here is the bare minimum you have to provide in order to create a new project:

- a dune.module file
  Usually you will only need to specify the parameters Module and Depends.
- *Note:* an dune-autogen script is *not* needed any more!
- a basic m4 file
  You need to provide two macros *MODULE*_CHECKS and *MODULE*_CHECK_MODULE.
- a configure.ac file
  Have look at the configure.ac in dune-grid for example. The most important part is the call to DUNE_CHECK_ALL which runs all checks needed for a DUNE module, plus the checks for the dependencies.

7 Dune module guidelines

A DUNE module should comply with the following rules:

- Documentation is located under doc/ and gets web-installed under BASEDIR/doc/.
- automake includes are located in dune-common. To use them, you will have to make a symbolic link to dune-common/am/ (see 4.1.2). The symlink creation should be handled by the dune-autogen (see 4.3).
- The am/ directory does not get included in the tarball.
- Header files that can be used by other DUNE modules should be accessible via #include <dune/foo/bar.hh>. In order to work with a freshly checkout version of your module you will usually need to create a local symbolic link dune -> *module-directory/*. This link gets created by the DUNE_SYMLINK command in your configure.ac. When running make install all header files should be installed into *prefix*/include/dune/.

8 Further documentation

automake & Makefile.am
http://www.gnu.org/software/automake/manual/
The automake manual describes in detail how to write and maintain a Makefile.am and the usage of automake.

autoconf & configure.ac
http://www.gnu.org/software/autoconf/manual/
The autoconf manual covers the usage of autoconf and how to write configure.ac files (sometimes they are called configure.in).

Autoconf Macro Archive
`http://autoconf-archive.cryp.to/`
The Autoconf Macro Archive provides macros that can be integrated in your `configure.ac` in order to search for certain software. These macros are useful to many software writers using the autoconf tool, but too specific to be included into autoconf itself.

libtool
`http://www.gnu.org/software/libtool/manual.html`
The `libtool` manual offers further information on the usage of `libtool` package and gives a good overview of the different problems/aspects of creating portable libraries.

autobook
`http://sources.redhat.com/autobook/`
The autobook is a complete book describing the GNU toolchain (`autoconf`, `automake` and `libtool`). It contains many recipes on how to use the autotools. The book is available as an online version.

dune-project
`http://www.dune-project.org/`
The official homepage of `DUNE`.