

How To Create a GNU Autoconf / Automake Based Configure Script for Your Application

Under Construction

by Prof.Dr. Walter Roth
University of Applied Science Suedwestfalen, Germany

Table of Contents

| | | |
|-------|--|---|
| 1 | Typography..... | 2 |
| 2 | Why Should You Use GNU AutoXXX Tools?..... | 2 |
| 3 | How Does it Work?..... | 3 |
| 4 | What do You Need?..... | 3 |
| 5 | Building a configure Script for „Hello World“ Step by Step | 4 |
| 5.1 | General Comments | 4 |
| 5.2 | Preparing the Sources..... | 4 |
| 5.3 | Creating a makefile.am..... | 4 |
| 5.4 | Creating the configure.ac File..... | 5 |
| 5.5 | Creating config.h.in..... | 5 |
| 5.6 | Creating the aclocal.m4 File..... | 5 |
| 5.7 | Creating configure..... | 5 |
| 5.8 | Creating makefile.in..... | 6 |
| 5.9 | Testing the package..... | 6 |
| 6 | More Complex Applications..... | 7 |
| 6.1 | Applications with subdirectories..... | 7 |
| 6.2 | Applications with libraries..... | 8 |
| 6.2.1 | Static Libraries..... | 8 |
| 6.2.2 | Shared Libraries..... | 8 |
| 7 | Literature..... | 9 |

1 General Information on This Document

1.1 Typography

Because some fonts are difficult to read, the following table gives images of the characters, that are important for commands. This document was written in Times New Roman 12 for the normal text and Courier 11 for the command lines and sources. Note that the double minus may be drawn as a single longer line.

| <i>Character Name</i> | <i>Character Image</i> | |
|-----------------------|------------------------|----------------|
| | <i>Times</i> | <i>Courier</i> |
| minus | - | - |
| double minus | -- | -- |
| underscore | — | — |
| dot | . | . |
| | ~ | ~ |

1.2 Name Definitions

The application is the program that you are developing.

The target system is the computer, into which your program shall be installed.

The development system is the computer where the development takes place.

2 Why Should You Use GNU AutoXXX Tools?

Using Autoconf and Automake is the only (reasonable) way to create Makefiles for your application, that work on every system for which the GNU tools exist. GNU tools are available for all Unixes, Windows (Cygwin), MacOS.

This How To is meant to be a straight forward cooking recipe that leads you to your first configure script without reading hundreds of pages of documentation. However, for more complex applications, there is no way around reading. You find the stuff for reading at:

<http://www.gnu.org/software/autoconf>

<http://www.gnu.org/software/automake>

3 How Does it Work?

The GNU Autoconf system consists out of a couple of programs, that finally create makefiles for your application. For each subdirectory of your distribution, a makefile of it's own is created. Because the makefiles are specific for the user's machine (the target system), on which your program shall run, they have to be created on that machine itself. All the information on the target system is available at this machine. The makefiles are created by a lengthy script program called "configure", which you have to supply together with your sources. On the target system, configure builds the makefiles according to the results of a lot of tests it runs and the rules that you have put into a set of makefile.am files. The makefiles support a lot of different targets, the first of them is called "all". Make all creates the binaries of your program. The "install" target installs those binaries, and the "uninstall" target uninstalls. Thus the user can get a working sample of your program by entering the top directory of your application and typing these three well-known commands:

```
./configure
make
make install
```

The second line is equivalent to make all, because all is the first target. Of course, the make utility and a compiler must be available on the target system. However, this is checked by configure. If configure does not find all the programs it needs, it stops with an error message.

On your system however, the configure script itself has to be created, which is a much more complex task.

4 What do You Need?

First of all, you need your sources. Make shure, that there is a top directory of your application and that all files, that are needed for the application are in this directory and its subdirectories, and that the application compiles without errors.

Unless you have not already done so, it's now the time to add a few files, that provide a minimum of documentation for your application. The following files **must** exist in your top application directory:

| <i>Filename</i> | <i>Contents</i> |
|-----------------|--|
| INSTALL | Installation instructions. You may copy a standard INSTALL file from an Automake based application and use that as a start. Then add your application specific informations. |
| README | What the user should know about the application. It is a good idea to start this file with a short description of the application's purpose. |

| <i>Filename</i> | <i>Contents</i> |
|-----------------|---|
| AUTHORS | The list of the authors, that contributed to the application. |
| NEWS | The latest news about the application. |
| ChangeLog | Change history of the application |

All of these files need not contain a single line of text, therefore for a first run you may just create a set of empty files. However, this mini – documentation is very important for users. Therefore you should put some work into thoroughly writing it. The README file is the most important part of this documentation. Make shure that it is as informative as possible.

Secondly, of course you need the GNU tools. Fortunately, the GNU tools are included in all modern Linux distributions. You should verify, that they have been installed. (On a SuSE 9.0 system, install the „extended developer“ selection). To check for the tools, just type

```
which automake
```

this should create a response like:

```
/usr/bin/automake
```

If you get an empty line as a response, the GNU tools are most likely not yet installed.

5 Building a configure Script for „Hello World“ Step by Step

5.1 General Comments

Commands in the following text are in *Courier* font. All commands may be executed under your user account. There is no need to become root. The name of the example application is „myapplication“ and its version is 1.0. The example application has only one directory named myapplication, and only a single source code file named main.c. You will have to edit your sources, your documentation and the files configure.ac and makefile.am. All the other stuff is created automatically.

5.2 Preparing the Sources

cd into the top directory of your application.

```
cd myapplication
```

Create the empty documentation files as specified above and keep in mind to fill in senseful text later.

```
touch INSTALL README AUTHORS NEWS ChangeLog
```

5.3 Creating a makefile.am

5.3.1 Building the Application

The makefile.am contains the informations about the application, that are needed by the configure script to create the final Makefile. You have to create a makefile.am, that contains the targets, sources and subdirectories of your application. This is the makefile.am for the example application, which has got no subdirectories:

```
##Process this file with automake to create Makefile.in
bin_PROGRAMS = myapplication
```

```
myapplication_SOURCES = main.c
```

The first line is a default comment, that should be used for every makefile.am. The second line lists the files, that are the binary programs to be built and to be installed into the (bin) directory for binary programs of the target computer. In our case, it's only myapplication and this file will be installed in /usr/bin on a SuSe 9.0 system, which is the default for binary programs. Refer to the Automake documentation, chapter 7, if you want to build more than one program or your programs have to be installed into other directories.

The third line lists all the sources, that belong to the application. The first word is composed from the application's name and `_SOURCES`.

5.3.2 Installing Files

The Makefile.am does not only manage the build process of your application, it also defines the target directories for all files that need to be installed into the target system. The standard installation directories have already been predefined by automake. The most important are:

- `prefix`: the top of the installation tree, standard is /usr/local (for KDE: /opt/kde)
- `bindir`: the directory for the binary programs
- `libexecdir`: the directory for the libraries belonging to the program

There are more predefined directories, please refer to “The GNU Coding Standards”, Directory variables(5). Files that go into one of these directories are listed like follows:

```
bin_PROGRAMS = myapplication
```

The file myapplication goes to `bindir`. The Directory identifier is always the name of the variable without the “dir”.

If you need some files to be installed in a non-standard directory, you may define it yourself. The name must end with `dir`. For example:

```
htmldir = $(prefix)/html
```

This is the /html directory directly under the prefix. In your Makefile.am you define the files to be installed there as follows:

```
html_DATA = usermanual.html
```

In addition to the standard directories, automake defines directories for the individual package: `pkglibdir`, `pkgincludedir` and `pkgdatadir` may be used, if you want to keep your files in clearly recognizable subdirectories under the standard `lib`, `include` or `data` directories. All of these installation directories will be created and named like the package itself.

Some files will not be needed for every kind of final installation. For instance, images (icons), that will be compiled into the executable, are not needed any longer, once the executable is compiled.

However, they are not real source files. Because they have to be included into the distribution, you list these files as `EXTRA_dist` files. For each primary variable (e.g. directory) an `EXTRA_` variable is recognized by automake. Put those files that are not always needed into the `EXTRA_` lists.

5.4 Creating the configure.ac File

Notice: `configure.ac` (ac for autoconf) was called `configure.in` in earlier versions of automake and autoconf. There is a tool called `autoscan`, that can create a default startup version of this file by scanning the contents of your sources. Therefore enter your application top directory and create a `configure.scan` file as a starter for `configure.ac` using `autoscan`.

```
autoscan
```

The resulting `configure.scan` file may look like this:

```

#                                     -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.

AC_PREREQ(2.59)
AC_INIT(FULL-PACKAGE-NAME, VERSION, BUG-REPORT-ADDRESS)
AC_CONFIG_SRCDIR([config.h.in])
AC_CONFIG_HEADER([config.h])

# Checks for programs.
AC_PROG_CXX
AC_PROG_CC
AC_PROG_CPP
AC_PROG_INSTALL
AC_PROG_LN_S
AC_PROG_MAKE_SET
AC_PROG_RANLIB

# Checks for libraries.

# Checks for header files.

# Checks for typedefs, structures, and compiler characteristics.
AC_HEADER_STDBOOL
AC_C_CONST

# Checks for library functions.

AC_CONFIG_FILES([Makefile
                 src/Makefile])
AC_OUTPUT

```

Edit the file `configure.scan`. You need to edit the line:

```
AC_INIT(FULL-PACKAGE-NAME, VERSION, BUG-REPORT-ADDRESS)
```

Replace `FULL-PACKAGE-NAME` with your application's name and `VERSION` with its version number. `BUG-REPORT-ADDRESS` is optional and should be the email address, to which bugs are to be reported. The example line now looks like this:

```
AC_INIT(myapplication, 1.0)
```

Directly after this line, add a new line, that invokes automake support.

```
AM_INIT_AUTOMAKE(@PACKAGE_NAME@, @PACKAGE_VERSION@)
```

This line uses the variables `PACKAGE_NAME` and `PACKAGE_VERSION` that have been defined by the `AC_INIT` macro in the line before and passes them to automake. The `@`'s mark the included string as a variable identifier. Without the `@`'s the strings would be passed literally, resulting in a package named `PACKAGE_NAME_PACKAGE_VERSION`.

The `AC_CONFIG_SRCDIR` makes configure check for the file `config.h.in` in the `src` directory. The `AC_CONFIG_HEADER` macro indicates, that you want to use a configuration header file.

The next macros check for the various programs required to build the application. Add here every program you need on the target system.

In the typedefs etc. group you may check for special properties of the build system on the target machine. The `AC_HEADER_STDBOOL` macro, that autoscan added for the example application, checks for the availability of `stdbool.h` and whether a C99 conformal `bool` type exists. The `AC_C_CONST` macro checks for a working `const` mechanism, which is needed in the example program.

Finally, in `AC_CONFIG_FILES`, you specify all the makefiles, that you want configure to generate and make configure create the outputfiles with `AC_OUTPUT`, which is normally the last line of a `configure.ac` file.

Now save the edited file as `configure.ac`.

5.5 Creating `config.h.in`

Run `autoheader` to create the file `config.h.in` from your `configure.ac` file. If you want special `#defines` to be included in your `config.h` file, you have to define them in `configure.ac`. See the documentation of `autoconf`, `AC_CONFIG_HEADER`.

```
autoheader
```

This will create the `config.h` file, which consists out of defines that describe the code of the application. The following text is a cut-out of the example program's `config.h.in`.

```
/* config.h.in.  Generated from configure.in by autoheader.  */

* Define to 1 if stdbool.h conforms to C99. */
#undef HAVE_STDBOOL_H

/* Define to 1 if the system has the type `_Bool'. */
#undef HAVE__BOOL

/* Name of package */
#undef PACKAGE

/* Define to the address where bug reports for this package should be
sent. */
#undef PACKAGE_BUGREPORT

/* Define to the full name of this package. */
#undef PACKAGE_NAME

/* Define to the full name and version of this package. */
#undef PACKAGE_STRING

/* Define to the one symbol short name of this package. */
#undef PACKAGE_TARNAME

/* Define to the version of this package. */
#undef PACKAGE_VERSION

/* Define to 1 if you have the ANSI C header files. */
#undef STDC_HEADERS

/* Version number of package */
#undef VERSION

/* Define to empty if `const' does not conform to ANSI C. */
#undef const
```

5.6 Creating the `aclocal.m4` File

Fortunately there is a program, that does the job. Unless you have not written your own macros for autoconf, all you need to do is to call:

```
aclocal
```

This will create `aclocal.m4`. This file contains the macros for autoconf, that are available on your machine and are referred to in your `configure.ac` file or are needed as sub-macros. The file contains the full source code of the macros and therefore it is quite long. If not all macros could be found on your machine, try to get them from the autoconf macro archive at www.gnu.org/software/ac-archive. There are lots of macros available for most of the problems you might have. For instance, there is a `BNV_HVE_QT` macro that checks for the Qt library and a `MDL_HAVE_OPENGL` macro that checks for OpenGL.

If you can not find a macro, that covers your needs, you will have to write one of your own. The "Goat Book" (1), tells you how to do it.

5.7 Creating `configure`

Autoconf is now able to create a `configure` script from your `configure.ac` and `aclocal.m4` files. All you need to do is to call:

```
autoconf
```

This will produce a `configure` script, which may be up to 20000 lines long for a GUI application.

5.8 Creating `makefile.in`

The `makefile.in` contains the information from `makefile.am` plus a lot more, that can be added automatically. The `Makefile.in` is needed by the `configure` script to create the final `Makefile`. Fortunately again, there is a program that does the job. `Automake` will do it for you. However, several other files, which have to be included in the distribution package of your application are still missing in your application's top directory. `Automake` will copy those files from the GNU tools installation and then create `makefile.in` when you call:

```
automake -a
```

or more readable:

```
automake -add-missing
```

5.9 Testing the package

The `configure` script is now ready to create a `makefile` on every system supported by GNU. `Configure` accepts a variety of arguments on the command line. Run `./configure --help` to get an overview. The most important of the possible arguments are probably the `--enable-FEATURE`, where `feature` is one of many available features. For program development it is essential to switch on debugging with `--enable-debug=full`. For the program user, the `--prefix` parameter and the `--with-LIBRARY-dir` parameter control the various installation directories. For a first try, run `configure` without any arguments. This will produce a set of `makefiles` for a release version of your program with the installation top directory, the so called prefix, set to `/usr/local`. Simply type:

```
./configure
```

You will see a lot of checking... messages and hopefully configure terminates with a set of lines like this:

```
configure: creating ./config.status
config.status: creating makefile
config.status: creating config.h
config.status: config.h is unchanged
config.status: executing depfiles commands
```

Next, test the new makefile by calling:

```
make
```

This should compile the application without errors. At this point, DO NOT call `make install` as root, otherwise you will install your program to the default prefix, which most probably is the wrong place. The application can be installed to the given (default) prefix by calling:

```
make install
```

As long as you are not root, you will see a lot of error messages, that complain about not being able to write the files. Starting `make install` as a simple user may be quite useful to find out, where your application wants to install its files.

Moreover, your Makefile will support all standard targets (commands) like `clean`, `dist`, `uninstall` and so on.

In order to get a development version of your program, you have to run `configure` again, to produce a set of makefiles that support debugging. Run

```
./configure --enable-debug=full
```

Then run

```
make
```

again. This time a program is compiled, that contains debugging information. Therefore the executable file is a lot bigger than the one compiled previously. You may now run your program under the control of a debugger.

6 More Complex Applications

6.1 Applications with subdirectories

You need a `makefile.am` in the top directory (myapplication) and in every subdirectory (`src`, `doc`, `img`), that belongs to the distribution. Subdirectories like `myapplication/CVS`, which are not part of the distribution, have to be skipped. The direct subdirectories of the respective directory have to be listed in the `makefile.am` as follows:

```
SUBDIRS = subdir1 subdir2 subdir3
```

You may **not** specify sub-subdirectories in a `SUBDIRS` line. The `makefile.am` in each subdirectory specifies only the **direct** subdirectories of this directory.

For the example with the top directory `myapplication` and the subdirectories `CVS` for the administration of `CVS`, `src` for sources, `doc` for documentation and `img` for images, the top level `makefile.am` in the `myapplication` directory looks like this:

```
##Process this file with automake to create Makefile.in
SUBDIRS = src doc img
```

It is a good idea, to use a separate directory called `admin` for most of the many files used and created by the `autoXXX` tools themselves. This will keep the top directory more readable. You achieve this by adding

```
AC_CONFIG_AUX_DIR(admin)
```

directly after the AC_INIT Macro. Now you have to create the admin directory manually, so that it is ready for use by automake.

```
mkdir admin
```

Source files are listed as usual in the myapplication_SOURCES list. However, this time the sources are in the src directory, Therefore they are listed in myapplication/src/makefile.am. The bin_PROGRAMS list of binaries is also specified in this file. This is the file myapplication/src/makefile.am:

```
##Process this file with automake to create Makefile.in
bin_PROGRAMS = myapplication
myapplication_SOURCES = main.c
```

All non-source files, that you want to have included into the distribution, have to be listed as EXTRA_DIST files. If the doc directory contains a doc file named index.html, you have to add that file to the EXTRA_DIST list of the makefile.am in myapplication/doc directory:

```
##Process this file with automake to create Makefile.in
EXTRA_DIST = index.html page1.html
```

Finally, this is the file in the myapplication/img directory:

```
##Process this file with automake to create Makefile.in
EXTRA_DIST = image1.png image2.bmp
```

Run autoscan and edit the resulting configure.scan file as described in section 3.3. Verify, that the AC_CONFIG_FILES macro at the end of the file lists a makefile for every subdirectory, that has to be included in your distribution. Run from the top directory of the application (myapplication):

```
aclocal
autoconf
autoheader
automake -a
```

Verify that automake created a makefile.in for every makefile.am.

6.2 Applications with libraries

6.2.1 Static Libraries

Static libraries are built much like applications, however the target is specified as _LIBRARIES variable. The library mylib would be specified as:

```
lib_LIBRARIES = mylib.a
```

if it is to be installed into the global lib directory (default: /usr/lib),

```
pkglib_LIBRARIES = mylib.a
```

if it is to be installed into the application's lib directory (default: /usr/myapplication/lib) or even

```
noinst_LIBRARIES = mylib.a
```

if it is only used during the build process and needs not to be installed.

An example makefile.am might look like this:

6.2.2 Shared Libraries

Building shared libraries, however is „a relatively complex matter“. You should refer to the automake and libtool documentation for this. However, here is how it works for simple cases: Use the `_LTLIBRARIES` macro for the libraries to be built with libtool. The names of the libraries should begin with `lib` and end with `.la` (e.g. `libmylib.la`). In the corresponding `_SOURCES` makro the dot (`.`) before `la` has to be replaced with an underscore (`_`). For a library named `mylib` that is to be installed in `libdir`, the macros would be:

```
lib_LTLIBRARIES = libmylib.la
libmylib_la_SOURCES = mylib.c
```

In the `configure.ac` file add the macro `AC_PROG_LIBTOOL` right after the `AC_PROG_CC` makro. This makes autoconf add libtool support to the configure script.

Before calling automake, call `libtoolize`. This will add a few files that are required by automake.

7 Literature

1. Gary Vaughan, Ben Elliston, Tom Tromey, Ian Taylor: “GNU Autoconf, Automake and Libtool”, New Riders Publishing, 2000, also available online at www.gnu.org
2. GNU Automake
<http://www.gnu.org/software/automake/manual/automake.html>
3. GNU Autoconf
<http://www.gnu.org/software/autoconf/manual/autoconf-2.57/autoconf.html>
4. Libtool
5. The GNU Coding Standards
<http://www.gnu.org/prep/standards/standards.html>