https://www.gnu.org/software/automake/manual/automake.html#Hello-World

2.4 A Small Hello World

In this section, we recreate the `amhello-1.0` package from scratch. The first subsection shows how to call the Autotools to instantiate the GNU Build System, while the second explains the meaning of the `configure.ac` and `Makefile.am` files read by the Autotools.

2.4.1 Creating `amhello-1.0.tar.gz`

Here is how we can recreate `amhello-1.0.tar.gz` from scratch. The package is simple enough so that we will only need to write 5 files. (You may copy them from the final `amhello-1.0.tar.gz` that is distributed with Automake if you do not want to write them.)

Create the following files in an empty directory.

- `src/main.c` is the source file for the `hello` program. We store it in the `src/` subdirectory, because later, when the package evolves, it will ease the addition of a `man/` directory for man pages, a `data/` directory for data files, etc.

```
~/amhello % cat src/main.c
#include <config.h>
#include <stdio.h>

int
main (void)
{
  puts ("Hello World!");
  puts ("This is " PACKAGE_STRING ".");
  return 0;
}
```

- `README` contains some very limited documentation for our little package.

```
~/amhello % cat README
This is a demonstration package for GNU Automake.
Type 'info Automake' to read the Automake manual.
```

- `Makefile.am` and `src/Makefile.am` contain Automake instructions for these two directories.

```
~/amhello % cat src/Makefile.am
bin_PROGRAMS = hello
hello_SOURCES = main.c
~/amhello % cat Makefile.am
SUBDIRS = src
dist_doc_DATA = README
```

- Finally, `configure.ac` contains Autoconf instructions to create the `configure` script.

```
~/amhello % cat configure.ac
AC_INIT([amhello], [1.0], [bug-automake@gnu.org])
AM_INIT_AUTOMAKE([-Wall -Werror foreign])
AC_PROG_CC
AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([
 Makefile
 src/Makefile
])
AC_OUTPUT
```

Once you have these five files, it is time to run the Autotools to instantiate the build system. Do this using the `autoreconf` command as follows:

```
~/amhello % autoreconf --install
configure.ac: installing './install-sh'
configure.ac: installing './missing'
configure.ac: installing './compile'
src/Makefile.am: installing './depcomp'
```

At this point the build system is complete.

In addition to the three scripts mentioned in its output, you can see that `autoreconf` created four other files: `configure`, `config.h.in`, `Makefile.in`, and `src/Makefile.in`. The latter three files are templates that will be adapted to the system by `configure` under the names `config.h`, `Makefile`, and `src/Makefile`. Let’s do this:

```
~/amhello % ./configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
checking for gcc... gcc
checking for C compiler default output file name... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking for style of include used by make... GNU
checking dependency style of gcc... gcc3
configure: creating ./config.status
config.status: creating Makefile
config.status: creating src/Makefile
config.status: creating config.h
config.status: executing depfiles commands
```

You can see `Makefile`, `src/Makefile`, and `config.h` being created at the end after `configure` has probed the system. It is now possible to run all the targets we wish (see Standard Targets). For instance:

```
~/amhello % make
…
~/amhello % src/hello
Hello World!
This is amhello 1.0.
~/amhello % make distcheck
…
=============================================
amhello-1.0 archives ready for distribution:
amhello-1.0.tar.gz
=============================================
```

Note that running `autoreconf` is only needed initially when the GNU Build System does not exist. When you later change some instructions in a `Makefile.am` or `configure.ac`, the relevant part of the build system will be regenerated automatically when you execute `make`.
`autoreconf` is a script that calls `autoconf`, `automake`, and a bunch of other commands in the right order. If you are beginning with these tools, it is not important to figure out in which order all of these tools should be invoked and why. However, because Autoconf and Automake have separate manuals, the important point to understand is that `autoconf` is in charge of

creating configure from configure.ac, while automake is in charge of creating Makefile.ins from Makefile.ams and configure.ac. This should at least direct you to the right manual when seeking answers.

Next: amhello's Makefile.am Setup Explained, Previous: Creating amhello, Up: Hello World [Contents][Index]

2.4.2 amhello’s configure.ac Setup Explained

Let us begin with the contents of configure.ac.

```
AC_INIT([amhello], [1.0], [bug-automake@gnu.org])
AM_INIT_AUTOMAKE([-Wall -Werror foreign])
AC_PROG_CC
AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([
 Makefile
 src/Makefile
])
AC_OUTPUT
```

This file is read by both autoconf (to create configure) and automake (to create the various Makefile.ins). It contains a series of M4 macros that will be expanded as shell code to finally form the configure script. We will not elaborate on the syntax of this file, because the Autoconf manual has a whole section about it (see Writing configure.ac in *The Autoconf Manual*).

The macros prefixed with AC_ are Autoconf macros, documented in the Autoconf manual (see Autoconf Macro Index in *The Autoconf Manual*). The macros that start with AM_ are Automake macros, documented later in this manual (see Macro Index).

The first two lines of configure.ac initialize Autoconf and Automake. AC_INIT takes in as parameters the name of the package, its version number, and a contact address for bug-reports about the package (this address is output at the end of ./configure --help, for instance). When adapting this setup to your own package, by all means please do not blindly copy Automake’s address: use the mailing list of your package, or your own mail address.

The argument to AM_INIT_AUTOMAKE is a list of options for automake (see Options). -Wall and -Werror ask automake to turn on all warnings and report them as errors. We are speaking of **Automake** warnings here, such as dubious instructions in Makefile.am. This has absolutely nothing to do with how the compiler will be called, even though it may support options with similar names. Using -Wall -Werror is a safe setting when starting to work on a package: you do not want to miss any issues. Later you may decide to relax things a bit.

The foreignoption tells Automake that this package will not follow the GNU Standards. GNU packages should always distribute additional files such as ChangeLog, AUTHORS, etc. We do not want automake to complain about these missing files in our small example.
The AC_PROG_CC line causes the configure script to search for a C compiler and define the variable CC with its name. The src/Makefile.in file generated by Automake uses the variable CCto build hello, so when configure creates src/Makefile from src/Makefile.in, it will define CC with the value it has found. If Automake is asked to create a Makefile.in that uses CC but configure.ac does not define it, it will suggest you add a call to AC_PROG_CC.
The AC_CONFIG_HEADERS([config.h]) invocation causes the configure script to create a config.h file gathering '#define's defined by other macros in configure.ac. In our case, the AC_INIT macro already defined a few of them. Here is an excerpt of config.h after configure has run:

```
…

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

#define PACKAGE_BUGREPORT "bug-automake@gnu.org"



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

#define PACKAGE_STRING "amhello 1.0"

…
```

As you probably noticed, src/main.c includes config.h so it can use PACKAGE_STRING. In a real-world project, config.h can grow really big, with one '#define' per feature probed on the system.
The AC_CONFIG_FILES macro declares the list of files that configure should create from their *.in templates. Automake also scans this list to find the Makefile.am files it must process. (This is important to remember: when adding a new directory to your project, you should add its Makefile to this list, otherwise Automake will never process the new Makefile.am you wrote in that directory.)
Finally, the AC_OUTPUT line is a closing command that actually produces the part of the script in charge of creating the files registered with AC_CONFIG_HEADERS and AC_CONFIG_FILES.
When starting a new project, we suggest you start with such a simple configure.ac, and gradually add the other tests it requires. The command autoscan can also suggest a few of the tests your package may need (see Using autoscan to Create configure.ac in *The Autoconf Manual*).

Previous: amhello's configure.ac Setup Explained, Up: Hello World   [Contents][Index]

2.4.3 amhello's Makefile.am Setup Explained

We now turn to src/Makefile.am. This file contains Automake instructions to build and install hello.

```
bin_PROGRAMS = hello
hello_SOURCES = main.c
```

A Makefile.am has the same syntax as an ordinary Makefile. When automake processes a Makefile.am it copies the entire file into the output Makefile.in (that will be later turned into Makefile by configure) but will react to certain variable definitions by generating some build rules and other variables. Often Makefile.ams contain only a list of variable definitions as above, but they can also contain other variable and rule definitions that automake will pass along without interpretation.

Variables that end with _PROGRAMS are special variables that list programs that the resulting Makefile should build. In Automake speak, this _PROGRAMS suffix is called a *primary*; Automake recognizes other primaries such as _SCRIPTS, _DATA, _LIBRARIES, etc. corresponding to different types of files.

The 'bin' part of the bin_PROGRAMS tells automake that the resulting programs should be installed in *bindir*. Recall that the GNU Build System uses a set of variables to denote destination directories and allow users to customize these locations (see Standard Directory Variables). Any such directory variable can be put in front of a primary (omitting the dir suffix) to tell automake where to install the listed files.

Programs need to be built from source files, so for each program *prog* listed in a _PROGRAMS variable, automake will look for another variable named *prog*_SOURCES listing its source files. There may be more than one source file: they will all be compiled and linked together.

Automake also knows that source files need to be distributed when creating a tarball (unlike built programs). So a side-effect of this hello_SOURCES declaration is that main.c will be part of the tarball created by make dist.

Finally here are some explanations regarding the top-level Makefile.am.

```
SUBDIRS = src
dist_doc_DATA = README
```

SUBDIRS is a special variable listing all directories that make should recurse into before processing the current directory. So this line is responsible for make building src/hello even though we run it from the top-level. This line also causes make install to install src/hello before installing README (not that this order matters).

The line `dist_doc_DATA = README` causes `README` to be distributed and installed in *docdir*. Files listed with the `_DATA` primary are not automatically part of the tarball built with `make dist`, so we add the `dist_` prefix so they get distributed. However, for `README` it would not have been necessary: `automake` automatically distributes any `README` file it encounters (the list of other files automatically distributed is presented by `automake --help`). The only important effect of this second line is therefore to install `README` during `make install`.

One thing not covered in this example is accessing the installation directory values (see Standard Directory Variables) from your program code, that is, converting them into defined macros. For this, see Defining Directories in *The Autoconf Manual*.