GNU Automake

For version 1.4, 10 January 1999

David MacKenzie and Tom Tromey

Copyright © 1995, 96 Free Software Foundation, Inc.

This is the first edition of the GNU Automake documentation,
and is consistent with GNU Automake 1.4.

Published by the Free Software Foundation
59 Temple Place - Suite 330,
Boston, MA 02111-1307 USA

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the Free Software Foundation.

1 Introduction

Automake is a tool for automatically generating 'Makefile.in's from files called 'Makefile.am'. Each 'Makefile.am' is basically a series of make macro definitions (with rules being thrown in occasionally). The generated 'Makefile.in's are compliant with the GNU Makefile standards.

The GNU Makefile Standards Document (see section "Makefile Conventions" in *The GNU Coding Standards*) is long, complicated, and subject to change. The goal of Automake is to remove the burden of Makefile maintenance from the back of the individual GNU maintainer (and put it on the back of the Automake maintainer).

The typical Automake input file is simply a series of macro definitions. Each such file is processed to create a 'Makefile.in'. There should generally be one 'Makefile.am' per directory of a project.

Automake does constrain a project in certain ways; for instance it assumes that the project uses Autoconf (see section "Introduction" in *The Autoconf Manual*), and enforces certain restrictions on the 'configure.in' contents.

Automake requires perl in order to generate the 'Makefile.in's. However, the distributions created by Automake are fully GNU standards-compliant, and do not require perl in order to be built.

Mail suggestions and bug reports for Automake to bug-automake@gnu.org.

2 General ideas

The following sections cover a few basic ideas that will help you understand how Automake works.

2.1 General Operation

Automake works by reading a 'Makefile.am' and generating a 'Makefile.in'. Certain macros and targets defined in the 'Makefile.am' instruct Automake to generate more specialized code; for instance, a 'bin_PROGRAMS' macro definition will cause targets for compiling and linking programs to be generated.

The macro definitions and targets in the 'Makefile.am' are copied verbatim into the generated file. This allows you to add arbitrary code into the generated 'Makefile.in'. For instance the Automake distribution includes a non-standard cvs-dist target, which the Automake maintainer uses to make distributions from his source control system.

Note that GNU make extensions are not recognized by Automake. Using such extensions in a 'Makefile.am' will lead to errors or confusing behavior.

Automake tries to group comments with adjoining targets and macro definitions in an intelligent way.

A target defined in 'Makefile.am' generally overrides any such target of a similar name that would be automatically generated by automake. Although this is a supported feature, it is generally best to avoid making use of it, as sometimes the generated rules are very particular.

Similarly, a macro defined in 'Makefile.am' will override any definition of the macro that automake would ordinarily create. This feature is more often useful than the ability to override a target definition. Be warned that many of the macros generated by automake are considered to be for internal use only, and their names might change in future releases.

When examining a macro definition, Automake will recursively examine macros referenced in the definition. For example, if Automake is looking at the content of foo_SOURCES in this snippet

```
xs = a.c b.c
foo_SOURCES = c.c $(xs)
```

it would use the files 'a.c', 'b.c', and 'c.c' as the contents of foo_SOURCES.

Automake also allows a form of comment which is *not* copied into the output; all lines beginning with '##' are completely ignored by Automake.

It is customary to make the first line of 'Makefile.am' read:

```
## Process this file with automake to produce Makefile.in
```

2.2 Depth

automake supports three kinds of directory hierarchy: 'flat', 'shallow', and 'deep'.

A *flat* package is one in which all the files are in a single directory. The 'Makefile.am' for such a package by definition lacks a SUBDIRS macro. An example of such a package is termutils.

A *deep* package is one in which all the source lies in subdirectories; the top level directory contains mainly configuration information. GNU cpio is a good example of such a package, as is GNU tar. The top level 'Makefile.am' for a deep package will contain a SUBDIRS macro, but no other macros to define objects which are built.

A *shallow* package is one in which the primary source resides in the top-level directory, while various parts (typically libraries) reside in subdirectories. Automake is one such package (as is GNU make, which does not currently use automake).

2.3 Strictness

While Automake is intended to be used by maintainers of GNU packages, it does make some effort to accommodate those who wish to use it, but do not want to use all the GNU conventions.

To this end, Automake supports three levels of *strictness*—the strictness indicating how stringently Automake should check standards conformance.

The valid strictness levels are:

'foreign' Automake will check for only those things which are absolutely required for proper operations. For instance, whereas GNU standards dictate the existence of a 'NEWS' file, it will not be required in this mode. The name comes from the fact that Automake is intended to be used for GNU programs; these relaxed rules are not the standard mode of operation.

'gnu' Automake will check—as much as possible—for compliance to the GNU standards for packages. This is the default.

'`gnits`' Automake will check for compliance to the as-yet-unwritten *Gnits standards*. These are based on the GNU standards, but are even more detailed. Unless you are a Gnits standards contributor, it is recommended that you avoid this option until such time as the Gnits standard is actually published.

For more information on the precise implications of the strictness level, see Chapter 19 [Gnits], page 36.

2.4 The Uniform Naming Scheme

Automake macros (from here on referred to as *variables*) generally follow a *uniform naming scheme* that makes it easy to decide how programs (and other derived objects) are built, and how they are installed. This scheme also supports `configure` time determination of what should be built.

At `make` time, certain variables are used to determine which objects are to be built. These variables are called *primary variables*. For instance, the primary variable `PROGRAMS` holds a list of programs which are to be compiled and linked.

A different set of variables is used to decide where the built objects should be installed. These variables are named after the primary variables, but have a prefix indicating which standard directory should be used as the installation directory. The standard directory names are given in the GNU standards (see section "Directory Variables" in *The GNU Coding Standards*). Automake extends this list with `pkglibdir`, `pkgincludedir`, and `pkgdatadir`; these are the same as the non-'`pkg`' versions, but with '`@PACKAGE@`' appended. For instance, `pkglibdir` is defined as `$(datadir)/@PACKAGE@`.

For each primary, there is one additional variable named by prepending '`EXTRA_`' to the primary name. This variable is used to list objects which may or may not be built, depending on what `configure` decides. This variable is required because Automake must statically know the entire list of objects that may be built in order to generate a '`Makefile.in`' that will work in all cases.

For instance, `cpio` decides at configure time which programs are built. Some of the programs are installed in `bindir`, and some are installed in `sbindir`:

```
EXTRA_PROGRAMS = mt rmt
bin_PROGRAMS = cpio pax
sbin_PROGRAMS = @PROGRAMS@
```

Defining a primary variable without a prefix (e.g. `PROGRAMS`) is an error.

Note that the common '`dir`' suffix is left off when constructing the variable names; thus one writes '`bin_PROGRAMS`' and not '`bindir_PROGRAMS`'.

Not every sort of object can be installed in every directory. Automake will flag those attempts it finds in error. Automake will also diagnose obvious misspellings in directory names.

Sometimes the standard directories—even as augmented by Automake— are not enough. In particular it is sometimes useful, for clarity, to install objects in a subdirectory of some predefined directory. To this end, Automake allows you to extend the list of possible installation directories. A given prefix (e.g. '`zar`') is valid if a variable of the same name with '`dir`' appended is defined (e.g. `zardir`).

For instance, until HTML support is part of Automake, you could use this to install raw HTML documentation:

```
htmldir = $(prefix)/html
html_DATA = automake.html
```

The special prefix '`noinst`' indicates that the objects in question should not be installed at all.

The special prefix '`check`' indicates that the objects in question should not be built until the `make check` command is run.

Possible primary names are '`PROGRAMS`', '`LIBRARIES`', '`LISP`', '`SCRIPTS`', '`DATA`', '`HEADERS`', '`MANS`', and '`TEXINFOS`'.

2.5 How derived variables are named

Sometimes a Makefile variable name is derived from some text the user supplies. For instance, program names are rewritten into Makefile macro names. Automake canonicalizes this text, so that it does not have to follow Makefile macro naming rules. All characters in the name except for letters, numbers, and the underscore are turned into underscores when making macro references. For example, if your program is named `sniff-glue`, the derived variable name would be `sniff_glue_SOURCES`, not `sniff-glue_SOURCES`.

3 Some example packages

3.1 A simple example, start to finish

Let's suppose you just finished writing `zardoz`, a program to make your head float from vortex to vortex. You've been using Autoconf to provide a portability framework, but your '`Makefile.in`'s have been ad-hoc. You want to make them bulletproof, so you turn to Automake.

The first step is to update your '`configure.in`' to include the commands that `automake` needs. The simplest way to do this is to add an `AM_INIT_AUTOMAKE` call just after `AC_INIT`:

```
AM_INIT_AUTOMAKE(zardoz, 1.0)
```

Since your program doesn't have any complicating factors (e.g., it doesn't use `gettext`, it doesn't want to build a shared library), you're done with this part. That was easy!

Now you must regenerate '`configure`'. But to do that, you'll need to tell `autoconf` how to find the new macro you've used. The easiest way to do this is to use the `aclocal` program to generate your '`aclocal.m4`' for you. But wait... you already have an '`aclocal.m4`', because you had to write some hairy macros for your program. The `aclocal` program lets you put your own macros into '`acinclude.m4`', so simply rename and then run:

```
mv aclocal.m4 acinclude.m4
aclocal
autoconf
```

Now it is time to write your '`Makefile.am`' for `zardoz`. Since `zardoz` is a user program, you want to install it where the rest of the user programs go. Additionally, `zardoz` has

some Texinfo documentation. Your '`configure.in`' script uses `AC_REPLACE_FUNCS`, so you need to link against '`@LIBOBJS@`'. So here's what you'd write:

```
bin_PROGRAMS = zardoz
zardoz_SOURCES = main.c head.c float.c vortex9.c gun.c
zardoz_LDADD = @LIBOBJS@

info_TEXINFOS = zardoz.texi
```

Now you can run `automake --add-missing` to generate your '`Makefile.in`' and grab any auxiliary files you might need, and you're done!

3.2 A classic program

GNU hello (`ftp://prep.ai.mit.edu/pub/gnu/hello-1.3.tar.gz`) is renowned for its classic simplicity and versatility. This section shows how Automake could be used with the GNU Hello package. The examples below are from the latest beta version of GNU Hello, but with all of the maintainer-only code stripped out, as well as all copyright comments.

Of course, GNU Hello is somewhat more featureful than your traditional two-liner. GNU Hello is internationalized, does option processing, and has a manual and a test suite. GNU Hello is a deep package.

Here is the '`configure.in`' from GNU Hello:

```
dnl Process this file with autoconf to produce a configure script.
AC_INIT(src/hello.c)
AM_INIT_AUTOMAKE(hello, 1.3.11)
AM_CONFIG_HEADER(config.h)

dnl Set of available languages.
ALL_LINGUAS="de fr es ko nl no pl pt sl sv"

dnl Checks for programs.
AC_PROG_CC
AC_ISC_POSIX

dnl Checks for libraries.

dnl Checks for header files.
AC_STDC_HEADERS
AC_HAVE_HEADERS(string.h fcntl.h sys/file.h sys/param.h)

dnl Checks for library functions.
AC_FUNC_ALLOCA

dnl Check for st_blksize in struct stat
AC_ST_BLKSIZE

dnl internationalization macros
AM_GNU_GETTEXT
AC_OUTPUT([Makefile doc/Makefile intl/Makefile po/Makefile.in \
```

```
                src/Makefile tests/Makefile tests/hello],
      [chmod +x tests/hello])
```

The 'AM_' macros are provided by Automake (or the Gettext library); the rest are standard Autoconf macros.

The top-level 'Makefile.am':

```
EXTRA_DIST = BUGS ChangeLog.0
SUBDIRS = doc intl po src tests
```

As you can see, all the work here is really done in subdirectories.

The 'po' and 'intl' directories are automatically generated using gettextize; they will not be discussed here.

In 'doc/Makefile.am' we see:

```
info_TEXINFOS = hello.texi
hello_TEXINFOS = gpl.texi
```

This is sufficient to build, install, and distribute the GNU Hello manual.

Here is 'tests/Makefile.am':

```
TESTS = hello
EXTRA_DIST = hello.in testdata
```

The script 'hello' is generated by configure, and is the only test case. make check will run this test.

Last we have 'src/Makefile.am', where all the real work is done:

```
bin_PROGRAMS = hello
hello_SOURCES = hello.c version.c getopt.c getopt1.c getopt.h system.h
hello_LDADD = @INTLLIBS@ @ALLOCA@
localedir = $(datadir)/locale
INCLUDES = -I../intl -DLOCALEDIR=\"$(localedir)\"
```

3.3 Building etags and ctags

Here is another, trickier example. It shows how to generate two programs (ctags and etags) from the same source file ('etags.c'). The difficult part is that each compilation of 'etags.c' requires different cpp flags.

```
bin_PROGRAMS = etags ctags
ctags_SOURCES =
ctags_LDADD = ctags.o

etags.o: etags.c
        $(COMPILE) -DETAGS_REGEXPS -c etags.c

ctags.o: etags.c
        $(COMPILE) -DCTAGS -o ctags.o -c etags.c
```

Note that ctags_SOURCES is defined to be empty—that way no implicit value is substituted. The implicit value, however, is used to generate etags from 'etags.o'.

ctags_LDADD is used to get 'ctags.o' into the link line. ctags_DEPENDENCIES is generated by Automake.

The above rules won't work if your compiler doesn't accept both '-c' and '-o'. The simplest fix for this is to introduce a bogus dependency (to avoid problems with a parallel make):

```
etags.o: etags.c ctags.o
        $(COMPILE) -DETAGS_REGEXPS -c etags.c

ctags.o: etags.c
        $(COMPILE) -DCTAGS -c etags.c && mv etags.o ctags.o
```

Also, these explicit rules do not work if the de-ANSI-fication feature is used (see Section 7.10 [ANSI], page 25). Supporting de-ANSI-fication requires a little more work:

```
etags._o: etags._c ctags.o
        $(COMPILE) -DETAGS_REGEXPS -c etags.c

ctags._o: etags._c
        $(COMPILE) -DCTAGS -c etags.c && mv etags._o ctags.o
```

4 Creating a 'Makefile.in'

To create all the 'Makefile.in's for a package, run the automake program in the top level directory, with no arguments. automake will automatically find each appropriate 'Makefile.am' (by scanning 'configure.in'; see Chapter 5 [configure], page 9) and generate the corresponding 'Makefile.in'. Note that automake has a rather simplistic view of what constitutes a package; it assumes that a package has only one 'configure.in', at the top. If your package has multiple 'configure.in's, then you must run automake in each directory holding a 'configure.in'.

You can optionally give automake an argument; '.am' is appended to the argument and the result is used as the name of the input file. This feature is generally only used to automatically rebuild an out-of-date 'Makefile.in'. Note that automake must always be run from the topmost directory of a project, even if being used to regenerate the 'Makefile.in' in some subdirectory. This is necessary because automake must scan 'configure.in', and because automake uses the knowledge that a 'Makefile.in' is in a subdirectory to change its behavior in some cases.

automake accepts the following options:

'-a'
'--add-missing'
: Automake requires certain common files to exist in certain situations; for instance 'config.guess' is required if 'configure.in' runs AC_CANONICAL_HOST. Automake is distributed with several of these files; this option will cause the missing ones to be automatically added to the package, whenever possible. In general if Automake tells you a file is missing, try using this option. By default Automake tries to make a symbolic link pointing to its own copy of the missing file; this can be changed with --copy.

'--amdir=*dir*'
: Look for Automake data files in directory *dir* instead of in the installation directory. This is typically used for debugging.

'--build-dir=*dir*'
: Tell Automake where the build directory is. This option is used when including dependencies into a 'Makefile.in' generated by make dist; it should not be used otherwise.

'-c'

'--copy'
: When used with --add-missing, causes installed files to be copied. The default is to make a symbolic link.

'--cygnus'
: Causes the generated 'Makefile.in's to follow Cygnus rules, instead of GNU or Gnits rules. For more information, see Chapter 20 [Cygnus], page 36.

'--foreign'
: Set the global strictness to 'foreign'. For more information, see Section 2.3 [Strictness], page 2.

'--gnits'
: Set the global strictness to 'gnits'. For more information, see Chapter 19 [Gnits], page 36.

'--gnu'
: Set the global strictness to 'gnu'. For more information, see Chapter 19 [Gnits], page 36. This is the default strictness.

'--help'
: Print a summary of the command line options and exit.

'-i'

'--include-deps'
: Include all automatically generated dependency information (see Section 7.11 [Dependencies], page 25) in the generated 'Makefile.in'. This is generally done when making a distribution; see Chapter 13 [Dist], page 31.

'--generate-deps'
: Generate a file concatenating all automatically generated dependency information (see Section 7.11 [Dependencies], page 25) into one file, '.dep_segment'. This is generally done when making a distribution; see Chapter 13 [Dist], page 31. It is useful when maintaining a 'SMakefile' or makefiles for other platforms ('Makefile.DOS', etc.) It can only be used in conjunction with '--include-deps', '--srcdir-name', and '--build-dir'. Note that if this option is given, no other processing is done.

'--no-force'
: Ordinarily automake creates all 'Makefile.in's mentioned in 'configure.in'. This option causes it to only update those 'Makefile.in's which are out of date with respect to one of their dependents.

'-o *dir*'

'--output-dir=*dir*'
: Put the generated 'Makefile.in' in the directory *dir*. Ordinarily each 'Makefile.in' is created in the directory of the corresponding 'Makefile.am'. This option is used when making distributions.

'--srcdir-name=*dir*'
: Tell Automake the name of the source directory associated with the current build. This option is used when including dependencies into a 'Makefile.in' generated by make dist; it should not be used otherwise.

'-v'
'--verbose'
: Cause Automake to print information about which files are being read or created.

'--version'
: Print the version number of Automake and exit.

5 Scanning 'configure.in'

Automake scans the package's 'configure.in' to determine certain information about the package. Some autoconf macros are required and some variables must be defined in 'configure.in'. Automake will also use information from 'configure.in' to further tailor its output.

Automake also supplies some Autoconf macros to make the maintenance easier. These macros can automatically be put into your 'aclocal.m4' using the aclocal program.

5.1 Configuration requirements

The simplest way to meet the basic Automake requirements is to use the macro AM_INIT_AUTOMAKE (see Section 5.4 [Macros], page 13). But if you prefer, you can do the required steps by hand:

- Define the variables PACKAGE and VERSION with AC_SUBST. PACKAGE should be the name of the package as it appears when bundled for distribution. For instance, Automake defines PACKAGE to be 'automake'. VERSION should be the version number of the release that is being developed. We recommend that you make 'configure.in' the only place in your package where the version number is defined; this makes releases simpler.

  Automake doesn't do any interpretation of PACKAGE or VERSION, except in 'Gnits' mode (see Chapter 19 [Gnits], page 36).
- Use the macro AC_ARG_PROGRAM if a program or script is installed. See section "Transforming Program Names When Installing" in *The Autoconf*.
- Use AC_PROG_MAKE_SET if the package is not flat. See section "Creating Output Files" in *The Autoconf Manual*.
- Use AM_SANITY_CHECK to make sure the build environment is sane.
- Call AC_PROG_INSTALL (see section "Particular Program Checks" in *The Autoconf Manual*).
- Use AM_MISSING_PROG to see whether the programs aclocal, autoconf, automake, autoheader, and makeinfo are in the build environment. Here is how this is done:

```
missing_dir=`cd $ac_aux_dir && pwd`
AM_MISSING_PROG(ACLOCAL, aclocal, $missing_dir)
```

```
AM_MISSING_PROG(AUTOCONF, autoconf, $missing_dir)
AM_MISSING_PROG(AUTOMAKE, automake, $missing_dir)
AM_MISSING_PROG(AUTOHEADER, autoheader, $missing_dir)
AM_MISSING_PROG(MAKEINFO, makeinfo, $missing_dir)
```

Here are the other macros which Automake requires but which are not run by AM_INIT_AUTOMAKE:

AC_OUTPUT
: Automake uses this to determine which files to create (see section "Creating Output Files" in *The Autoconf Manual*). Listed files named Makefile are treated as 'Makefile's. Other listed files are treated differently. Currently the only difference is that a 'Makefile' is removed by make distclean, while other files are removed by make clean.

5.2 Other things Automake recognizes

Automake will also recognize the use of certain macros and tailor the generated 'Makefile.in' appropriately. Currently recognized macros and their effects are:

AC_CONFIG_HEADER
: Automake requires the use of AM_CONFIG_HEADER, which is similar to AC_CONFIG_HEADER (see section "Configuration Header Files" in *The Autoconf Manual*), but does some useful Automake-specific work.

AC_CONFIG_AUX_DIR
: Automake will look for various helper scripts, such as 'mkinstalldirs', in the directory named in this macro invocation. If not seen, the scripts are looked for in their 'standard' locations (either the top source directory, or in the source directory corresponding to the current 'Makefile.am', whichever is appropriate). See section "Finding 'configure' Input" in *The Autoconf Manual*. FIXME: give complete list of things looked for in this directory

AC_PATH_XTRA
: Automake will insert definitions for the variables defined by AC_PATH_XTRA into each 'Makefile.in' that builds a C program or library. See section "System Services" in *The Autoconf Manual*.

AC_CANONICAL_HOST
AC_CHECK_TOOL
: Automake will ensure that 'config.guess' and 'config.sub' exist. Also, the 'Makefile' variables 'host_alias' and 'host_triplet' are introduced. See both section "Getting the Canonical System Type" in *The Autoconf Manual*, and section "Generic Program Checks" in *The Autoconf Manual*.

AC_CANONICAL_SYSTEM
: This is similar to AC_CANONICAL_HOST, but also defines the 'Makefile' variables 'build_alias' and 'target_alias'. See section "Getting the Canonical System Type" in *The Autoconf Manual*.

`AC_FUNC_ALLOCA`
`AC_FUNC_GETLOADAVG`
`AC_FUNC_MEMCMP`
`AC_STRUCT_ST_BLOCKS`
`AC_FUNC_FNMATCH`
`AM_FUNC_STRTOD`
`AC_REPLACE_FUNCS`
`AC_REPLACE_GNU_GETOPT`
`AM_WITH_REGEX`

Automake will ensure that the appropriate dependencies are generated for the objects corresponding to these macros. Also, Automake will verify that the appropriate source files are part of the distribution. Note that Automake does not come with any of the C sources required to use these macros, so `automake -a` will not install the sources. See Section 7.2 [A Library], page 17, for more information. Also, see section "Particular Function Checks" in *The Autoconf Manual*.

`LIBOBJS` Automake will detect statements which put '`.o`' files into `LIBOBJS`, and will treat these additional files as if they were discovered via `AC_REPLACE_FUNCS`. See section "Generic Function Checks" in *The Autoconf Manual*.

`AC_PROG_RANLIB`

This is required if any libraries are built in the package. See section "Particular Program Checks" in *The Autoconf Manual*.

`AC_PROG_CXX`

This is required if any C++ source is included. See section "Particular Program Checks" in *The Autoconf Manual*.

`AC_PROG_F77`

This is required if any Fortran 77 source is included. This macro is distributed with Autoconf version 2.13 and later. See section "Particular Program Checks" in *The Autoconf Manual*.

`AC_F77_LIBRARY_LDFLAGS`

This is required for programs and shared libraries that are a mixture of languages that include Fortran 77 (see Section 7.8.3 [Mixing Fortran 77 With C and C++], page 22). See Section 5.4 [Autoconf macros supplied with Automake], page 13.

`AM_PROG_LIBTOOL`

Automake will turn on processing for `libtool` (see section "Introduction" in *The Libtool Manual*).

`AC_PROG_YACC`

If a Yacc source file is seen, then you must either use this macro or define the variable '`YACC`' in '`configure.in`'. The former is preferred (see section "Particular Program Checks" in *The Autoconf Manual*).

`AC_DECL_YYTEXT`

This macro is required if there is Lex source in the package. See section "Particular Program Checks" in *The Autoconf Manual*.

`AC_PROG_LEX`
: If a Lex source file is seen, then this macro must be used. See section "Particular Program Checks" in *The Autoconf Manual*.

`ALL_LINGUAS`
: If Automake sees that this variable is set in 'configure.in', it will check the 'po' directory to ensure that all the named '.po' files exist, and that all the '.po' files that exist are named.

`AM_C_PROTOTYPES`
: This is required when using automatic de-ANSI-fication; see Section 7.10 [ANSI], page 25.

`AM_GNU_GETTEXT`
: This macro is required for packages which use GNU gettext (see Section 9.2 [gettext], page 28). It is distributed with gettext. If Automake sees this macro it ensures that the package meets some of gettext's requirements.

`AM_MAINTAINER_MODE`
: This macro adds a '--enable-maintainer-mode' option to configure. If this is used, automake will cause 'maintainer-only' rules to be turned off by default in the generated 'Makefile.in's. This macro is disallowed in 'Gnits' mode (see Chapter 19 [Gnits], page 36). This macro defines the 'MAINTAINER_MODE' conditional, which you can use in your own 'Makefile.am'.

`AC_SUBST`
`AC_CHECK_TOOL`
`AC_CHECK_PROG`
`AC_CHECK_PROGS`
`AC_PATH_PROG`
`AC_PATH_PROGS`
: For each of these macros, the first argument is automatically defined as a variable in each generated 'Makefile.in'. See section "Setting Output Variables" in *The Autoconf Manual*, and section "Generic Program Checks" in *The Autoconf Manual*.

5.3 Auto-generating aclocal.m4

Automake includes a number of Autoconf macros which can be used in your package; some of them are actually required by Automake in certain situations. These macros must be defined in your 'aclocal.m4'; otherwise they will not be seen by autoconf.

The aclocal program will automatically generate 'aclocal.m4' files based on the contents of 'configure.in'. This provides a convenient way to get Automake-provided macros, without having to search around. Also, the aclocal mechanism is extensible for use by other packages.

At startup, aclocal scans all the '.m4' files it can find, looking for macro definitions. Then it scans 'configure.in'. Any mention of one of the macros found in the first step causes that macro, and any macros it in turn requires, to be put into 'aclocal.m4'.

The contents of 'acinclude.m4', if it exists, are also automatically included in 'aclocal.m4'. This is useful for incorporating local macros into 'configure'.

aclocal accepts the following options:

--acdir=*dir*
: Look for the macro files in *dir* instead of the installation directory. This is typically used for debugging.

--help
: Print a summary of the command line options and exit.

-I *dir*
: Add the directory *dir* to the list of directories searched for '.m4' files.

--output=*file*
: Cause the output to be put into *file* instead of 'aclocal.m4'.

--print-ac-dir
: Prints the name of the directory which aclocal will search to find the '.m4' files. When this option is given, normal processing is suppressed. This option can be used by a package to determine where to install a macro file.

--verbose
: Print the names of the files it examines.

--version
: Print the version number of Automake and exit.

5.4 Autoconf macros supplied with Automake

AM_CONFIG_HEADER
: Automake will generate rules to automatically regenerate the config header. If you do use this macro, you must create the file 'stamp-h.in' in your source directory. It can be empty.

AM_ENABLE_MULTILIB
: This is used when a "multilib" library is being built. A *multilib* library is one that is built multiple times, once per target flag combination. This is only useful when the library is intended to be cross-compiled. The first optional argument is the name of the 'Makefile' being generated; it defaults to 'Makefile'. The second option argument is used to find the top source directory; it defaults to the empty string (generally this should not be used unless you are familiar with the internals).

AM_FUNC_STRTOD
: If the strtod function is not available, or does not work correctly (like the one on SunOS 5.4), add 'strtod.o' to output variable LIBOBJS.

AM_FUNC_ERROR_AT_LINE
: If the function error_at_line is not found, then add 'error.o' to LIBOBJS.

AM_FUNC_MKTIME
: Check for a working mktime function. If not found, add 'mktime.o' to 'LIBOBJS'.

AM_FUNC_OBSTACK
: Check for the GNU obstacks code; if not found, add 'obstack.o' to 'LIBOBJS'.

AM_C_PROTOTYPES
: Check to see if function prototypes are understood by the compiler. If so, define 'PROTOTYPES' and set the output variables 'U' and 'ANSI2KNR' to the empty string. Otherwise, set 'U' to '_' and 'ANSI2KNR' to './ansi2knr'. Automake uses these values to implement automatic de-ANSI-fication.

AM_HEADER_TIOCGWINSZ_NEEDS_SYS_IOCTL
: If the use of TIOCGWINSZ requires '<sys/ioctl.h>', then define GWINSZ_IN_SYS_IOCTL. Otherwise TIOCGWINSZ can be found in '<termios.h>'.

AM_INIT_AUTOMAKE
: Runs many macros that most 'configure.in''s need. This macro has two required arguments, the package and the version number. By default this macro AC_DEFINE's 'PACKAGE' and 'VERSION'. This can be avoided by passing in a non-empty third argument.

AM_PATH_LISPDIR
: Searches for the program emacs, and, if found, sets the output variable lispdir to the full path to Emacs' site-lisp directory.

AM_PROG_CC_STDC
: If the C compiler in not in ANSI C mode by default, try to add an option to output variable CC to make it so. This macro tries various options that select ANSI C on some system or another. It considers the compiler to be in ANSI C mode if it handles function prototypes correctly.

  If you use this macro, you should check after calling it whether the C compiler has been set to accept ANSI C; if not, the shell variable am_cv_prog_cc_stdc is set to 'no'. If you wrote your source code in ANSI C, you can make an un-ANSIfied copy of it by using the ansi2knr option (see Section 7.10 [ANSI], page 25).

AM_PROG_LEX
: Like AC_PROG_LEX with AC_DECL_YYTEXT (see section "Particular Program Checks" in *The Autoconf Manual*), but uses the missing script on systems that do not have lex. 'HP-UX 10' is one such system.

AM_SANITY_CHECK
: This checks to make sure that a file created in the build directory is newer than a file in the source directory. This can fail on systems where the clock is set incorrectly. This macro is automatically run from AM_INIT_AUTOMAKE.

AM_SYS_POSIX_TERMIOS
: Check to see if POSIX termios headers and functions are available on the system. If so, set the shell variable am_cv_sys_posix_termios to 'yes'. If not, set the variable to 'no'.

AM_TYPE_PTRDIFF_T
: Define 'HAVE_PTRDIFF_T' if the type 'ptrdiff_t' is defined in '<stddef.h>'.

AM_WITH_DMALLOC
: Add support for the dmalloc (ftp://ftp.letters.com/src/dmalloc/dmalloc.tar.gz) package. If the user configures with '--with-dmalloc', then define WITH_DMALLOC and add '-ldmalloc' to LIBS.

AM_WITH_REGEX
: Adds '--with-regex' to the configure command line. If specified (the default), then the 'regex' regular expression library is used, 'regex.o' is put into 'LIBOBJS', and 'WITH_REGEX' is defined.. If '--without-regex' is given, then the 'rx' regular expression library is used, and 'rx.o' is put into 'LIBOBJS'.

5.5 Writing your own aclocal macros

The aclocal program doesn't have any built-in knowledge of any macros, so it is easy to extend it with your own macros.

This is mostly used for libraries which want to supply their own Autoconf macros for use by other programs. For instance the gettext library supplies a macro AM_GNU_GETTEXT which should be used by any package using gettext. When the library is installed, it installs this macro so that aclocal will find it.

A file of macros should be a series of AC_DEFUN's. The aclocal programs also understands AC_REQUIRE, so it is safe to put each macro in a separate file. See section "Prerequisite Macros" in *The Autoconf Manual*, and section "Macro Definitions" in *The Autoconf Manual*.

A macro file's name should end in '.m4'. Such files should be installed in '$(datadir)/aclocal'.

6 The top-level 'Makefile.am'

In non-flat packages, the top level 'Makefile.am' must tell Automake which subdirectories are to be built. This is done via the SUBDIRS variable.

The SUBDIRS macro holds a list of subdirectories in which building of various sorts can occur. Many targets (e.g. all) in the generated 'Makefile' will run both locally and in all specified subdirectories. Note that the directories listed in SUBDIRS are not required to contain 'Makefile.am's; only 'Makefile's (after configuration). This allows inclusion of libraries from packages which do not use Automake (such as gettext). The directories mentioned in SUBDIRS must be direct children of the current directory. For instance, you cannot put 'src/subdir' into SUBDIRS.

In a deep package, the top-level 'Makefile.am' is often very short. For instance, here is the 'Makefile.am' from the GNU Hello distribution:

```
EXTRA_DIST = BUGS ChangeLog.0 README-alpha
SUBDIRS = doc intl po src tests
```

It is possible to override the SUBDIRS variable if, like in the case of GNU Inetutils, you want to only build a subset of the entire package. In your 'Makefile.am' include:

```
SUBDIRS = @SUBDIRS@
```

Then in your 'configure.in' you can specify:

```
SUBDIRS = "src doc lib po"
AC_SUBST(SUBDIRS)
```

The upshot of this is that Automake is tricked into building the package to take the subdirs, but doesn't actually bind that list until `configure` is run.

Although the `SUBDIRS` macro can contain configure substitutions (e.g. '`@DIRS@`'); Automake itself does not actually examine the contents of this variable.

If `SUBDIRS` is defined, then your '`configure.in`' must include `AC_PROG_MAKE_SET`.

The use of `SUBDIRS` is not restricted to just the top-level '`Makefile.am`'. Automake can be used to construct packages of arbitrary depth.

By default, Automake generates '`Makefiles`' which work depth-first ('`postfix`'). However, it is possible to change this ordering. You can do this by putting '`.`' into `SUBDIRS`. For instance, putting '`.`' first will cause a '`prefix`' ordering of directories.

7 Building Programs and Libraries

A large part of Automake's functionality is dedicated to making it easy to build programs and libraries.

7.1 Building a program

In a directory containing source that gets built into a program (as opposed to a library), the '`PROGRAMS`' primary is used. Programs can be installed in `bindir`, `sbindir`, `libexecdir`, `pkglibdir`, or not at all ('`noinst`').

For instance:

```
bin_PROGRAMS = hello
```

In this simple case, the resulting '`Makefile.in`' will contain code to generate a program named `hello`. The variable `hello_SOURCES` is used to specify which source files get built into an executable:

```
hello_SOURCES = hello.c version.c getopt.c getopt1.c getopt.h system.h
```

This causes each mentioned '`.c`' file to be compiled into the corresponding '`.o`'. Then all are linked to produce '`hello`'.

If '*prog*`_SOURCES`' is needed, but not specified, then it defaults to the single file '`prog.c`'.

Multiple programs can be built in a single directory. Multiple programs can share a single source file, which must be listed in each '`_SOURCES`' definition.

Header files listed in a '`_SOURCES`' definition will be included in the distribution but otherwise ignored. In case it isn't obvious, you should not include the header file generated by '`configure`' in an '`_SOURCES`' variable; this file should not be distributed. Lex ('`.l`') and Yacc ('`.y`') files can also be listed; see Section 7.6 [Yacc and Lex], page 19.

Automake must know all the source files that could possibly go into a program, even if not all the files are built in every circumstance. Any files which are only conditionally built should be listed in the appropriate '`EXTRA_`' variable. For instance, if '`hello-linux.c`' were conditionally included in `hello`, the '`Makefile.am`' would contain:

```
EXTRA_hello_SOURCES = hello-linux.c
```

Similarly, sometimes it is useful to determine the programs that are to be built at configure time. For instance, GNU `cpio` only builds `mt` and `rmt` under special circumstances.

In this case, you must notify Automake of all the programs that can possibly be built, but at the same time cause the generated '`Makefile.in`' to use the programs specified by `configure`. This is done by having `configure` substitute values into each '`_PROGRAMS`' definition, while listing all optionally built programs in `EXTRA_PROGRAMS`.

If you need to link against libraries that are not found by `configure`, you can use `LDADD` to do so. This variable actually can be used to add any options to the linker command line.

Sometimes, multiple programs are built in one directory but do not share the same link-time requirements. In this case, you can use the '*prog*`_LDADD`' variable (where *prog* is the name of the program as it appears in some '`_PROGRAMS`' variable, and usually written in lowercase) to override the global `LDADD`. If this variable exists for a given program, then that program is not linked using `LDADD`.

For instance, in GNU cpio, `pax`, `cpio` and `mt` are linked against the library '`libcpio.a`'. However, `rmt` is built in the same directory, and has no such link requirement. Also, `mt` and `rmt` are only built on certain architectures. Here is what cpio's '`src/Makefile.am`' looks like (abridged):

```
bin_PROGRAMS = cpio pax @MT@
libexec_PROGRAMS = @RMT@
EXTRA_PROGRAMS = mt rmt

LDADD = ../lib/libcpio.a @INTLLIBS@
rmt_LDADD =

cpio_SOURCES = ...
pax_SOURCES = ...
mt_SOURCES = ...
rmt_SOURCES = ...
```

'*prog*`_LDADD`' is inappropriate for passing program-specific linker flags (except for '`-l`' and '`-L`'). So, use the '*prog*`_LDFLAGS`' variable for this purpose.

It is also occasionally useful to have a program depend on some other target which is not actually part of that program. This can be done using the '*prog*`_DEPENDENCIES`' variable. Each program depends on the contents of such a variable, but no further interpretation is done.

If '*prog*`_DEPENDENCIES`' is not supplied, it is computed by Automake. The automatically-assigned value is the contents of '*prog*`_LDADD`', with most configure substitutions, '`-l`', and '`-L`' options removed. The configure substitutions that are left in are only '`@LIBOBJS@`' and '`@ALLOCA@`'; these are left because it is known that they will not cause an invalid value for '*prog*`_DEPENDENCIES`' to be generated.

7.2 Building a library

Building a library is much like building a program. In this case, the name of the primary is '`LIBRARIES`'. Libraries can be installed in `libdir` or `pkglibdir`.

See Section 7.4 [A Shared Library], page 18, for information on how to build shared libraries using Libtool and the 'LTLIBRARIES' primary.

Each '_LIBRARIES' variable is a list of the libraries to be built. For instance to create a library named 'libcpio.a', but not install it, you would write:

```
noinst_LIBRARIES = libcpio.a
```

The sources that go into a library are determined exactly as they are for programs, via the '_SOURCES' variables. Note that the library name is canonicalized (see Section 2.5 [Canonicalization], page 4), so the '_SOURCES' variable corresponding to 'liblob.a' is 'liblob_a_SOURCES', not 'liblob.a_SOURCES'.

Extra objects can be added to a library using the '*library*_LIBADD' variable. This should be used for objects determined by configure. Again from cpio:

```
libcpio_a_LIBADD = @LIBOBJS@ @ALLOCA@
```

7.3 Special handling for LIBOBJS and ALLOCA

Automake explicitly recognizes the use of @LIBOBJS@ and @ALLOCA@, and uses this information, plus the list of LIBOBJS files derived from 'configure.in' to automatically include the appropriate source files in the distribution (see Chapter 13 [Dist], page 31). These source files are also automatically handled in the dependency-tracking scheme; see See Section 7.11 [Dependencies], page 25.

@LIBOBJS@ and @ALLOCA@ are specially recognized in any '_LDADD' or '_LIBADD' variable.

7.4 Building a Shared Library

Building shared libraries is a relatively complex matter. For this reason, GNU Libtool (see section "Introduction" in *The Libtool Manual*) was created to help build shared libraries in a platform-independent way.

Automake uses Libtool to build libraries declared with the 'LTLIBRARIES' primary. Each '_LTLIBRARIES' variable is a list of shared libraries to build. For instance, to create a library named 'libgettext.a' and its corresponding shared libraries, and install them in 'libdir', write:

```
lib_LTLIBRARIES = libgettext.la
```

Note that shared libraries *must* be installed, so check_LTLIBRARIES is not allowed. However, noinst_LTLIBRARIES is allowed. This feature should be used for libtool "convenience libraries".

For each library, the '*library*_LIBADD' variable contains the names of extra libtool objects ('.lo' files) to add to the shared library. The '*library*_LDFLAGS' variable contains any additional libtool flags, such as '-version-info' or '-static'.

Where an ordinary library might include @LIBOBJS@, a libtool library must use @LTLIBOBJS@. This is required because the object files that libtool operates on do not necessarily end in '.o'. The libtool manual contains more details on this topic.

For libraries installed in some directory, Automake will automatically supply the appropriate '-rpath' option. However, for libraries determined at configure time (and thus

mentioned in EXTRA_LTLIBRARIES), Automake does not know the eventual installation directory; for such libraries you must add the '-rpath' option to the appropriate '_LDFLAGS' variable by hand.

See section "The Libtool Manual" in *The Libtool Manual*, for more information.

7.5 Variables used when building a program

Occasionally it is useful to know which 'Makefile' variables Automake uses for compilations; for instance you might need to do your own compilation in some special cases.

Some variables are inherited from Autoconf; these are CC, CFLAGS, CPPFLAGS, DEFS, LDFLAGS, and LIBS.

There are some additional variables which Automake itself defines:

INCLUDES A list of '-I' options. This can be set in your 'Makefile.am' if you have special directories you want to look in. Automake already provides some '-I' options automatically. In particular it generates '-I$(srcdir)' and a '-I' pointing to the directory holding 'config.h' (if you've used AC_CONFIG_HEADER or AM_CONFIG_HEADER).

INCLUDES can actually be used for other cpp options besides '-I'. For instance, it is sometimes used to pass arbitrary '-D' options to the compiler.

COMPILE This is the command used to actually compile a C source file. The filename is appended to form the complete command line.

LINK This is the command used to actually link a C program.

7.6 Yacc and Lex support

Automake has somewhat idiosyncratic support for Yacc and Lex.

Automake assumes that the '.c' file generated by yacc (or lex) should be named using the basename of the input file. That is, for a yacc source file 'foo.y', Automake will cause the intermediate file to be named 'foo.c' (as opposed to 'y.tab.c', which is more traditional).

The extension of a yacc source file is used to determine the extension of the resulting 'C' or 'C++' file. Files with the extension '.y' will be turned into '.c' files; likewise, '.yy' will become '.cc'; '.y++', 'c++'; and '.yxx', '.cxx'.

Likewise, lex source files can be used to generate 'C' or 'C++'; the extensions '.l', '.ll', '.l++', and '.lxx' are recognized.

You should never explicitly mention the intermediate ('C' or 'C++') file in any 'SOURCES' variable; only list the source file.

The intermediate files generated by yacc (or lex) will be included in any distribution that is made. That way the user doesn't need to have yacc or lex.

If a yacc source file is seen, then your 'configure.in' must define the variable 'YACC'. This is most easily done by invoking the macro 'AC_PROG_YACC' (see section "Particular Program Checks" in *The Autoconf Manual*).

Similarly, if a `lex` source file is seen, then your '`configure.in`' must define the variable '`LEX`'. You can use '`AC_PROG_LEX`' to do this (see section "Particular Program Checks" in *The Autoconf Manual*). Automake's `lex` support also requires that you use the '`AC_DECL_YYTEXT`' macro—automake needs to know the value of '`LEX_OUTPUT_ROOT`'. This is all handled for you if you use the `AM_PROG_LEX` macro (see Section 5.4 [Macros], page 13).

Automake makes it possible to include multiple `yacc` (or `lex`) source files in a single program. Automake uses a small program called `ylwrap` to run `yacc` (or `lex`) in a subdirectory. This is necessary because yacc's output filename is fixed, and a parallel make could conceivably invoke more than one instance of `yacc` simultaneously. The `ylwrap` program is distributed with Automake. It should appear in the directory specified by '`AC_CONFIG_AUX_DIR`' (see section "Finding 'configure' Input" in *The Autoconf Manual*), or the current directory if that macro is not used in '`configure.in`'.

For `yacc`, simply managing locking is insufficient. The output of `yacc` always uses the same symbol names internally, so it isn't possible to link two `yacc` parsers into the same executable.

We recommend using the following renaming hack used in `gdb`:

```
#define yymaxdepth c_maxdepth
#define yyparse c_parse
#define yylex c_lex
#define yyerror c_error
#define yylval c_lval
#define yychar c_char
#define yydebug c_debug
#define yypact c_pact
#define yyr1 c_r1
#define yyr2 c_r2
#define yydef c_def
#define yychk c_chk
#define yypgo c_pgo
#define yyact c_act
#define yyexca c_exca
#define yyerrflag c_errflag
#define yynerrs c_nerrs
#define yyps c_ps
#define yypv c_pv
#define yys c_s
#define yy_yys c_yys
#define yystate c_state
#define yytmp c_tmp
#define yyv c_v
#define yy_yyv c_yyv
#define yyval c_val
#define yylloc c_lloc
#define yyreds c_reds
#define yytoks c_toks
#define yylhs c_yylhs
#define yylen c_yylen
#define yydefred c_yydefred
```

```
#define yydgoto c_yydgoto
#define yysindex c_yysindex
#define yyrindex c_yyrindex
#define yygindex c_yygindex
#define yytable  c_yytable
#define yycheck  c_yycheck
#define yyname   c_yyname
#define yyrule   c_yyrule
```

For each define, replace the 'c_' prefix with whatever you like. These defines work for `bison`, `byacc`, and traditional `yacc`s. If you find a parser generator that uses a symbol not covered here, please report the new name so it can be added to the list.

7.7 C++ Support

Automake includes full support for C++.

Any package including C++ code must define the output variable '`CXX`' in '`configure.in`'; the simplest way to do this is to use the `AC_PROG_CXX` macro (see section "Particular Program Checks" in *The Autoconf Manual*).

A few additional variables are defined when a C++ source file is seen:

`CXX` The name of the C++ compiler.

`CXXFLAGS` Any flags to pass to the C++ compiler.

`CXXCOMPILE`
: The command used to actually compile a C++ source file. The file name is appended to form the complete command line.

`CXXLINK` The command used to actually link a C++ program.

7.8 Fortran 77 Support

Automake includes full support for Fortran 77.

Any package including Fortran 77 code must define the output variable '`F77`' in '`configure.in`'; the simplest way to do this is to use the `AC_PROG_F77` macro (see section "Particular Program Checks" in *The Autoconf Manual*). See Section 7.8.4 [Fortran 77 and Autoconf], page 24.

A few additional variables are defined when a Fortran 77 source file is seen:

`F77` The name of the Fortran 77 compiler.

`FFLAGS` Any flags to pass to the Fortran 77 compiler.

`RFLAGS` Any flags to pass to the Ratfor compiler.

`F77COMPILE`
: The command used to actually compile a Fortran 77 source file. The file name is appended to form the complete command line.

`FLINK` The command used to actually link a pure Fortran 77 program or shared library.

Automake can handle preprocessing Fortran 77 and Ratfor source files in addition to compiling them[1]. Automake also contains some support for creating programs and shared libraries that are a mixture of Fortran 77 and other languages (see Section 7.8.3 [Mixing Fortran 77 With C and C++], page 22).

These issues are covered in the following sections.

7.8.1 Preprocessing Fortran 77

'`N.f`' is made automatically from '`N.F`' or '`N.r`'. This rule runs just the preprocessor to convert a preprocessable Fortran 77 or Ratfor source file into a strict Fortran 77 source file. The precise command used is as follows:

'`.F`' `$(F77) -F $(DEFS) $(INCLUDES) $(AM_CPPFLAGS) $(CPPFLAGS) $(AM_FFLAGS) $(FFLAGS)`

'`.r`' `$(F77) -F $(AM_FFLAGS) $(FFLAGS) $(AM_RFLAGS) $(RFLAGS)`

7.8.2 Compiling Fortran 77 Files

'`N.o`' is made automatically from '`N.f`', '`N.F`' or '`N.r`' by running the Fortran 77 compiler. The precise command used is as follows:

'`.f`' `$(F77) -c $(AM_FFLAGS) $(FFLAGS)`

'`.F`' `$(F77) -c $(DEFS) $(INCLUDES) $(AM_CPPFLAGS) $(CPPFLAGS) $(AM_FFLAGS) $(FFLAGS)`

'`.r`' `$(F77) -c $(AM_FFLAGS) $(FFLAGS) $(AM_RFLAGS) $(RFLAGS)`

7.8.3 Mixing Fortran 77 With C and C++

Automake currently provides *limited* support for creating programs and shared libraries that are a mixture of Fortran 77 and C and/or C++. However, there are many other issues related to mixing Fortran 77 with other languages that are *not* (currently) handled by Automake, but that are handled by other packages[2].

---

[1] Much, if not most, of the information in the following sections pertaining to preprocessing Fortran 77 programs was taken almost verbatim from section "Catalogue of Rules" in *The GNU Make Manual*.

[2] For example, the cfortran package (`http://www-zeus.desy.de/~burow/cfortran/`) addresses all of these inter-language issues, and runs under nearly all Fortran 77, C and C++ compilers on nearly all platforms. However, `cfortran` is not yet Free Software, but it will be in the next major release.

Automake can help in two ways:

1. Automatic selection of the linker depending on which combinations of source code.
2. Automatic selection of the appropriate linker flags (e.g. '`-L`' and '`-l`') to pass to the automatically selected linker in order to link in the appropriate Fortran 77 intrinsic and run-time libraries.

   These extra Fortran 77 linker flags are supplied in the output variable `FLIBS` by the `AC_F77_LIBRARY_LDFLAGS` Autoconf macro supplied with newer versions of Autoconf (Autoconf version 2.13 and later). See section "Fortran 77 Compiler Characteristics" in *The Autoconf*.

If Automake detects that a program or shared library (as mentioned in some `_PROGRAMS` or `_LTLIBRARIES` primary) contains source code that is a mixture of Fortran 77 and C and/or C++, then it requires that the macro `AC_F77_LIBRARY_LDFLAGS` be called in '`configure.in`', and that either `$(FLIBS)` or `@FLIBS@` appear in the appropriate `_LDADD` (for programs) or `_LIBADD` (for shared libraries) variables. It is the responsibility of the person writing the '`Makefile.am`' to make sure that `$(FLIBS)` or `@FLIBS@` appears in the appropriate `_LDADD` or `_LIBADD` variable.

For example, consider the following '`Makefile.am`':

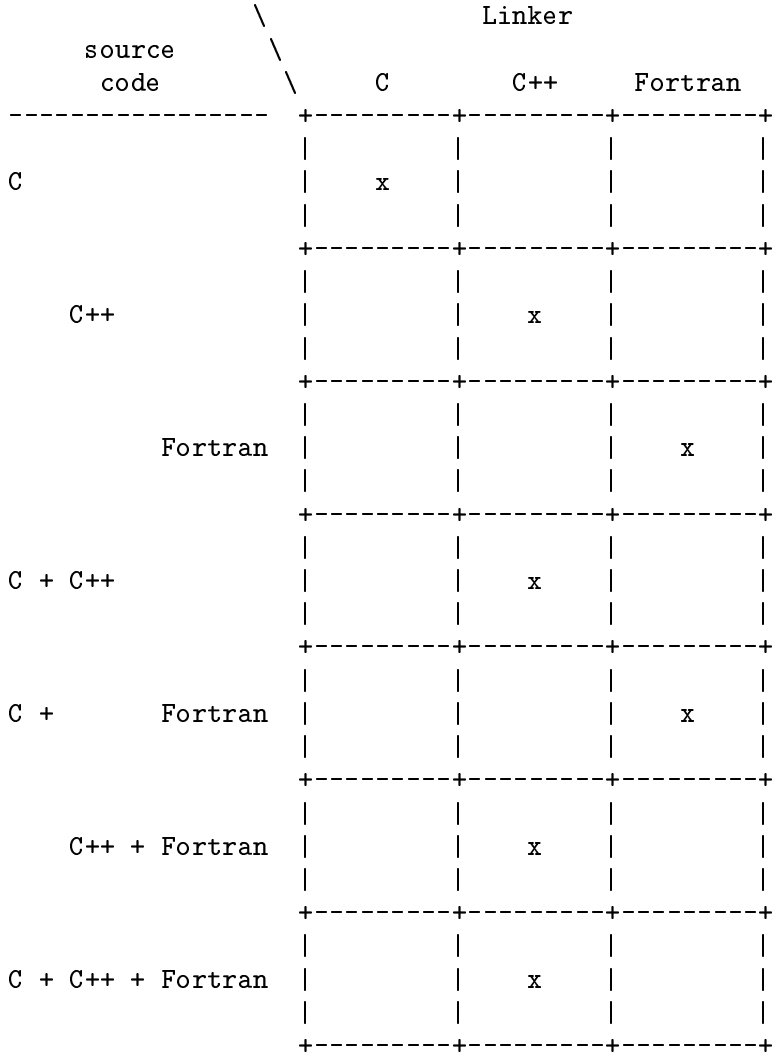
```
bin_PROGRAMS = foo
foo_SOURCES  = main.cc foo.f
foo_LDADD    = libfoo.la @FLIBS@

pkglib_LTLIBRARIES = libfoo.la
libfoo_la_SOURCES  = bar.f baz.c zardoz.cc
libfoo_la_LIBADD   = $(FLIBS)
```

In this case, Automake will insist that `AC_F77_LIBRARY_LDFLAGS` is mentioned in '`configure.in`'. Also, if `@FLIBS@` hadn't been mentioned in `foo_LDADD` and `libfoo_la_LIBADD`, then Automake would have issued a warning.

7.8.3.1 How the Linker is Chosen

The following diagram demonstrates under what conditions a particular linker is chosen by Automake.

For example, if Fortran 77, C and C++ source code were to be compiled into a program, then the C++ linker will be used. In this case, if the C or Fortran 77 linkers required any special libraries that weren't included by the C++ linker, then they must be manually added to an `_LDADD` or `_LIBADD` variable by the user writing the '`Makefile.am`'.



7.8.4 Fortran 77 and Autoconf

The current Automake support for Fortran 77 requires a recent enough version Autoconf that also includes support for Fortran 77. Full Fortran 77 support was added to Autoconf 2.13, so you will want to use that version of Autoconf or later.

7.9 Support for Other Languages

Automake currently only includes full support for C, C++ (see Section 7.7 [C++ Support], page 21)and Fortran 77 (see Section 7.8 [Fortran 77 Support], page 21). There is only rudimentary support for other languages, support for which will be improved based on user demand.

7.10 Automatic de-ANSI-fication

Although the GNU standards allow the use of ANSI C, this can have the effect of limiting portability of a package to some older compilers (notably SunOS).

Automake allows you to work around this problem on such machines by *de-ANSI-fying* each source file before the actual compilation takes place.

If the '`Makefile.am`' variable `AUTOMAKE_OPTIONS` (see Chapter 15 [Options], page 32) contains the option `ansi2knr` then code to handle de-ANSI-fication is inserted into the generated '`Makefile.in`'.

This causes each C source file in the directory to be treated as ANSI C. If an ANSI C compiler is available, it is used. If no ANSI C compiler is available, the `ansi2knr` program is used to convert the source files into K&R C, which is then compiled.

The `ansi2knr` program is simple-minded. It assumes the source code will be formatted in a particular way; see the `ansi2knr` man page for details.

Support for de-ANSI-fication requires the source files '`ansi2knr.c`' and '`ansi2knr.1`' to be in the same package as the ANSI C source; these files are distributed with Automake. Also, the package '`configure.in`' must call the macro `AM_C_PROTOTYPES` (see Section 5.4 [Macros], page 13).

Automake also handles finding the `ansi2knr` support files in some other directory in the current package. This is done by prepending the relative path to the appropriate directory to the `ansi2knr` option. For instance, suppose the package has ANSI C code in the '`src`' and '`lib`' subdirs. The files '`ansi2knr.c`' and '`ansi2knr.1`' appear in '`lib`'. Then this could appear in '`src/Makefile.am`':

```
AUTOMAKE_OPTIONS = ../lib/ansi2knr
```

If no directory prefix is given, the files are assumed to be in the current directory.

Files mentioned in `LIBOBJS` which need de-ANSI-fication will not be automatically handled. That's because `configure` will generate an object name like '`regex.o`', while `make` will be looking for '`regex_.o`' (when de-ANSI-fying). Eventually this problem will be fixed via `autoconf` magic, but for now you must put this code into your '`configure.in`', just before the `AC_OUTPUT` call:

```
# This is necessary so that .o files in LIBOBJS are also built via
# the ANSI2KNR-filtering rules.
LIBOBJS=`echo $LIBOBJS|sed 's/\.o /\$U.o /g;s/\.o$/\$U.o/'`
```

7.11 Automatic dependency tracking

As a developer it is often painful to continually update the '`Makefile.in`' whenever the include-file dependencies change in a project. Automake supplies a way to automatically track dependency changes, and distribute the dependencies in the generated '`Makefile.in`'.

Currently this support requires the use of GNU `make` and `gcc`. It might become possible in the future to supply a different dependency generating program, if there is enough demand. In the meantime, this mode is enabled by default if any C program or library is defined in the current directory, so you may get a '`Must be a separator`' error from non-GNU make.

When you decide to make a distribution, the `dist` target will re-run `automake` with '`--include-deps`' and other options. See Chapter 4 [Invoking Automake], page 7, and Chapter 15 [Options], page 32. This will cause the previously generated dependencies to be inserted into the generated '`Makefile.in`', and thus into the distribution. This step also turns off inclusion of the dependency generation code, so that those who download your distribution but don't use GNU `make` and `gcc` will not get errors.

When added to the '`Makefile.in`', the dependencies have all system-specific dependencies automatically removed. This can be done by listing the files in '`OMIT_DEPENDENCIES`'. For instance all references to system header files are removed by Automake. Sometimes it is useful to specify that a certain header file should be removed. For instance if your '`configure.in`' uses '`AM_WITH_REGEX`', then any dependency on '`rx.h`' or '`regex.h`' should be removed, because the correct one cannot be known until the user configures the package.

As it turns out, Automake is actually smart enough to handle the particular case of the regular expression header. It will also automatically omit '`libintl.h`' if '`AM_GNU_GETTEXT`' is used.

Automatic dependency tracking can be suppressed by putting `no-dependencies` in the variable `AUTOMAKE_OPTIONS`.

If you unpack a distribution made by `make dist`, and you want to turn on the dependency-tracking code again, simply re-run `automake`.

The actual dependency files are put under the build directory, in a subdirectory named '`.deps`'. These dependencies are machine specific. It is safe to delete them if you like; they will be automatically recreated during the next build.

8 Other Derived Objects

Automake can handle derived objects which are not C programs. Sometimes the support for actually building such objects must be explicitly supplied, but Automake will still automatically handle installation and distribution.

8.1 Executable Scripts

It is possible to define and install programs which are scripts. Such programs are listed using the '`SCRIPTS`' primary name. Automake doesn't define any dependencies for scripts; the '`Makefile.am`' should include the appropriate rules.

Automake does not assume that scripts are derived objects; such objects must be deleted by hand (see Chapter 12 [Clean], page 31).

The `automake` program itself is a Perl script that is generated at configure time from '`automake.in`'. Here is how this is handled:

```
bin_SCRIPTS = automake
```

Since `automake` appears in the `AC_OUTPUT` macro, a target for it is automatically generated.

Script objects can be installed in `bindir`, `sbindir`, `libexecdir`, or `pkgdatadir`.

8.2 Header files

Header files are specified by the '`HEADERS`' family of variables. Generally header files are not installed, so the `noinst_HEADERS` variable will be the most used.

All header files must be listed somewhere; missing ones will not appear in the distribution. Often it is clearest to list uninstalled headers with the rest of the sources for a program. See Section 7.1 [A Program], page 16. Headers listed in a '`_SOURCES`' variable need not be listed in any '`_HEADERS`' variable.

Headers can be installed in `includedir`, `oldincludedir`, or `pkgincludedir`.

8.3 Architecture-independent data files

Automake supports the installation of miscellaneous data files using the '`DATA`' family of variables.

Such data can be installed in the directories `datadir`, `sysconfdir`, `sharedstatedir`, `localstatedir`, or `pkgdatadir`.

By default, data files are *not* included in a distribution.

Here is how Automake installs its auxiliary data files:

```
pkgdata_DATA = clean-kr.am clean.am ...
```

8.4 Built sources

Occasionally a file which would otherwise be called '`source`' (e.g. a C '`.h`' file) is actually derived from some other file. Such files should be listed in the `BUILT_SOURCES` variable.

Built sources are also not compiled by default. You must explicitly mention them in some other '`_SOURCES`' variable for this to happen.

Note that, in some cases, `BUILT_SOURCES` will work in somewhat surprising ways. In order to get the built sources to work with automatic dependency tracking, the '`Makefile`' must depend on `$(BUILT_SOURCES)`. This can cause these sources to be rebuilt at what might seem like funny times.

9 Other GNU Tools

Since Automake is primarily intended to generate 'Makefile.in's for use in GNU programs, it tries hard to interoperate with other GNU tools.

9.1 Emacs Lisp

Automake provides some support for Emacs Lisp. The 'LISP' primary is used to hold a list of '.el' files. Possible prefixes for this primary are 'lisp_' and 'noinst_'. Note that if lisp_LISP is defined, then 'configure.in' must run AM_PATH_LISPDIR (see Section 5.4 [Macros], page 13).

By default Automake will byte-compile all Emacs Lisp source files using the Emacs found by AM_PATH_LISPDIR. If you wish to avoid byte-compiling, simply define the variable ELCFILES to be empty. Byte-compiled Emacs Lisp files are not portable among all versions of Emacs, so it makes sense to turn this off if you expect sites to have more than one version of Emacs installed. Furthermore, many packages don't actually benefit from byte-compilation. Still, we recommend that you leave it enabled by default. It is probably better for sites with strange setups to cope for themselves than to make the installation less nice for everybody else.

9.2 Gettext

If AM_GNU_GETTEXT is seen in 'configure.in', then Automake turns on support for GNU gettext, a message catalog system for internationalization (see section "GNU Gettext" in *GNU gettext utilities*).

The gettext support in Automake requires the addition of two subdirectories to the package, 'intl' and 'po'. Automake insures that these directories exist and are mentioned in SUBDIRS.

Furthermore, Automake checks that the definition of ALL_LINGUAS in 'configure.in' corresponds to all the valid '.po' files, and nothing more.

9.3 Guile

Automake provides some automatic support for writing Guile modules. Automake will turn on Guile support if the AM_INIT_GUILE_MODULE macro is used in 'configure.in'.

Right now Guile support just means that the AM_INIT_GUILE_MODULE macro is understood to mean:

- AM_INIT_AUTOMAKE is run.
- AC_CONFIG_AUX_DIR is run, with a path of '..'.

As the Guile module code matures, no doubt the Automake support will grow as well.

9.4 Libtool

Automake provides support for GNU Libtool (see section "Introduction" in *The Libtool Manual*) with the 'LTLIBRARIES' primary. See Section 7.4 [A Shared Library], page 18.

9.5 Java

Automake provides some minimal support for Java compilation with the 'JAVA' primary.

Any '.java' files listed in a '_JAVA' variable will be compiled with JAVAC at build time. By default, '.class' files are not included in the distribution.

Currently Automake enforces the restriction that only one '_JAVA' primary can be used in a given 'Makefile.am'. The reason for this restriction is that, in general, it isn't possible to know which '.class' files were generated from which '.java' files – so it would be impossible to know which files to install where.

10 Building documentation

Currently Automake provides support for Texinfo and man pages.

10.1 Texinfo

If the current directory contains Texinfo source, you must declare it with the 'TEXINFOS' primary. Generally Texinfo files are converted into info, and thus the info_TEXINFOS macro is most commonly used here. Note that any Texinfo source file must end in the '.texi' or '.texinfo' extension.

If the '.texi' file @includes 'version.texi', then that file will be automatically generated. The file 'version.texi' defines three Texinfo macros you can reference: EDITION, VERSION, and UPDATED. The first two hold the version number of your package (but are kept separate for clarity); the last is the date the primary file was last modified. The 'version.texi' support requires the mdate-sh program; this program is supplied with Automake and automatically included when automake is invoked with the --add-missing option.

Sometimes an info file actually depends on more than one '.texi' file. For instance, in GNU Hello, 'hello.texi' includes the file 'gpl.texi'. You can tell Automake about these dependencies using the *texi*_TEXINFOS variable. Here is how GNU Hello does it:

```
info_TEXINFOS = hello.texi
hello_TEXINFOS = gpl.texi
```

By default, Automake requires the file 'texinfo.tex' to appear in the same directory as the Texinfo source. However, if you used AC_CONFIG_AUX_DIR in 'configure.in' (see section "Finding 'configure' Input" in *The Autoconf Manual*), then 'texinfo.tex' is looked for there. Automake supplies 'texinfo.tex' if '--add-missing' is given.

If your package has Texinfo files in many directories, you can use the variable TEXINFO_TEX to tell Automake where to find the canonical 'texinfo.tex' for your package. The value of this variable should be the relative path from the current 'Makefile.am' to 'texinfo.tex':

```
TEXINFO_TEX = ../doc/texinfo.tex
```

The option 'no-texinfo.tex' can be used to eliminate the requirement for 'texinfo.tex'. Use of the variable TEXINFO_TEX is preferable, however, because that allows the dvi target to still work.

Automake generates an `install-info` target; some people apparently use this. By default, info pages are installed by '`make install`'. This can be prevented via the `no-installinfo` option.

10.2 Man pages

A package can also include man pages (but see the GNU standards on this matter, section "Man Pages" in *The GNU Coding Standards*.) Man pages are declared using the '`MANS`' primary. Generally the `man_MANS` macro is used. Man pages are automatically installed in the correct subdirectory of `mandir`, based on the file extension. They are not automatically included in the distribution.

By default, man pages are installed by '`make install`'. However, since the GNU project does not require man pages, many maintainers do not expend effort to keep the man pages up to date. In these cases, the `no-installman` option will prevent the man pages from being installed by default. The user can still explicitly install them via '`make install-man`'.

Here is how the documentation is handled in GNU `cpio` (which includes both Texinfo documentation and man pages):

```
info_TEXINFOS = cpio.texi
man_MANS = cpio.1 mt.1
EXTRA_DIST = $(man_MANS)
```

Texinfo source and info pages are all considered to be source for the purposes of making a distribution.

Man pages are not currently considered to be source, because it is not uncommon for man pages to be automatically generated. For the same reason, they are not automatically included in the distribution.

11 What Gets Installed

Naturally, Automake handles the details of actually installing your program once it has been built. All `PROGRAMS`, `SCRIPTS`, `LIBRARIES`, `LISP`, `DATA` and `HEADERS` are automatically installed in the appropriate places.

Automake also handles installing any specified info and man pages.

Automake generates separate `install-data` and `install-exec` targets, in case the installer is installing on multiple machines which share directory structure—these targets allow the machine-independent parts to be installed only once. The `install` target depends on both of these targets.

Automake also generates an `uninstall` target, an `installdirs` target, and an `install-strip` target.

It is possible to extend this mechanism by defining an `install-exec-local` or `install-data-local` target. If these targets exist, they will be run at '`make install`' time.

Variables using the standard directory prefixes '`data`', '`info`', '`man`', '`include`', '`oldinclude`', '`pkgdata`', or '`pkginclude`' (e.g. '`data_DATA`') are installed by '`install-data`'.

Variables using the standard directory prefixes '`bin`', '`sbin`', '`libexec`', '`sysconf`', '`localstate`', '`lib`', or '`pkglib`' (e.g. '`bin_PROGRAMS`') are installed by '`install-exec`'.

Any variable using a user-defined directory prefix with 'exec' in the name (e.g. 'myexecbin_PROGRAMS' is installed by 'install-exec'. All other user-defined prefixes are installed by 'install-data'.

Automake generates support for the 'DESTDIR' variable in all install rules. 'DESTDIR' is used during the 'make install' step to relocate install objects into a staging area. Each object and path is prefixed with the value of 'DESTDIR' before being copied into the install area. Here is an example of typical DESTDIR usage:

```
make DESTDIR=/tmp/staging install
```

This places install objects in a directory tree built under '/tmp/staging'. If '/gnu/bin/foo' and '/gnu/share/aclocal/foo.m4' are to be installed, the above command would install '/tmp/staging/gnu/bin/foo' and '/tmp/staging/gnu/share/aclocal/foo.m4'.

This feature is commonly used to build install images and packages. For more information, see section "Makefile Conventions" in *The GNU Coding Standards*.

12 What Gets Cleaned

The GNU Makefile Standards specify a number of different clean rules. Generally the files that can be cleaned are determined automatically by Automake. Of course, Automake also recognizes some variables that can be defined to specify additional files to clean. These variables are MOSTLYCLEANFILES, CLEANFILES, DISTCLEANFILES, and MAINTAINERCLEANFILES.

13 What Goes in a Distribution

The dist target in the generated 'Makefile.in' can be used to generate a gzip'd tar file for distribution. The tar file is named based on the 'PACKAGE' and 'VERSION' variables; more precisely it is named '*package*-*version*.tar.gz'. You can use the make variable 'GZIP_ENV' to control how gzip is run. The default setting is '--best'.

For the most part, the files to distribute are automatically found by Automake: all source files are automatically included in a distribution, as are all 'Makefile.am's and 'Makefile.in's. Automake also has a built-in list of commonly used files which, if present in the current directory, are automatically included. This list is printed by 'automake --help'. Also, files which are read by configure (i.e. the source files corresponding to the files specified in the AC_OUTPUT invocation) are automatically distributed.

Still, sometimes there are files which must be distributed, but which are not covered in the automatic rules. These files should be listed in the EXTRA_DIST variable. You can mention files from subdirectories in EXTRA_DIST. You can also mention a directory there; in this case the entire directory will be recursively copied into the distribution.

If you define SUBDIRS, Automake will recursively include the subdirectories in the distribution. If SUBDIRS is defined conditionally (see Chapter 18 [Conditionals], page 35), Automake will normally include all directories that could possibly appear in SUBDIRS in the distribution. If you need to specify the set of directories conditionally, you can set the variable DIST_SUBDIRS to the exact list of subdirectories to include in the distribution.

Occasionally it is useful to be able to change the distribution before it is packaged up. If the `dist-hook` target exists, it is run after the distribution directory is filled, but before the actual tar (or shar) file is created. One way to use this is for distributing files in subdirectories for which a new '`Makefile.am`' is overkill:

```
dist-hook:
        mkdir $(distdir)/random
        cp -p $(srcdir)/random/a1 $(srcdir)/random/a2 $(distdir)/random
```

Automake also generates a `distcheck` target which can be help to ensure that a given distribution will actually work. `distcheck` makes a distribution, and then tries to do a `VPATH` build.

14 Support for test suites

Automake supports two forms of test suites.

If the variable `TESTS` is defined, its value is taken to be a list of programs to run in order to do the testing. The programs can either be derived objects or source objects; the generated rule will look both in `srcdir` and '`.`'. Programs needing data files should look for them in `srcdir` (which is both an environment variable and a make variable) so they work when building in a separate directory (see section "Build Directories " in *The Autoconf Manual*), and in particular for the `distcheck` target (see Chapter 13 [Dist], page 31).

The number of failures will be printed at the end of the run. If a given test program exits with a status of 77, then its result is ignored in the final count. This feature allows non-portable tests to be ignored in environments where they don't make sense.

The variable `TESTS_ENVIRONMENT` can be used to set environment variables for the test run; the environment variable `srcdir` is set in the rule. If all your test programs are scripts, you can also set `TESTS_ENVIRONMENT` to an invocation of the shell (e.g. '`$(SHELL) -x`'); this can be useful for debugging the tests.

If '`dejagnu`' (`ftp://prep.ai.mit.edu/pub/gnu/dejagnu-1.3.tar.gz`) appears in `AUTOMAKE_OPTIONS`, then a `dejagnu`-based test suite is assumed. The value of the variable `DEJATOOL` is passed as the `--tool` argument to `runtest`; it defaults to the name of the package.

The variable `RUNTESTDEFAULTFLAGS` holds the `--tool` and `--srcdir` flags that are passed to dejagnu by default; this can be overridden if necessary.

The variables `EXPECT`, `RUNTEST` and `RUNTESTFLAGS` can also be overridden to provide project-specific values. For instance, you will need to do this if you are testing a compiler toolchain, because the default values do not take into account host and target names.

In either case, the testing is done via '`make check`'.

15 Changing Automake's Behavior

Various features of Automake can be controlled by options in the '`Makefile.am`'. Such options are listed in a special variable named `AUTOMAKE_OPTIONS`. Currently understood options are:

`gnits`
`gnu`
`foreign`

`cygnus`

Set the strictness as appropriate. The `gnits` option also implies `readme-alpha` and `check-news`.

`ansi2knr`
`path/ansi2knr`

Turn on automatic de-ANSI-fication. See Section 7.10 [ANSI], page 25. If preceded by a path, the generated '`Makefile.in`' will look in the specified directory to find the '`ansi2knr`' program. Generally the path should be a relative path to another directory in the same distribution (though Automake currently does not check this).

`check-news`

Cause `make dist` to fail unless the current version number appears in the first few lines of the '`NEWS`' file.

`dejagnu` Cause `dejagnu`-specific rules to be generated. See Chapter 14 [Tests], page 32.

`dist-shar`

Generate a `dist-shar` target as well as the ordinary `dist` target. This new target will create a shar archive of the distribution.

`dist-zip` Generate a `dist-zip` target as well as the ordinary `dist` target. This new target will create a zip archive of the distribution.

`dist-tarZ`

Generate a `dist-tarZ` target as well as the ordinary `dist` target. This new target will create a compressed tar archive of the distribution; a traditional `tar` and `compress` will be assumed. Warning: if you are actually using `GNU tar`, then the generated archive might contain nonportable constructs.

`no-dependencies`

This is similar to using '`--include-deps`' on the command line, but is useful for those situations where you don't have the necessary bits to make automatic dependency tracking work See Section 7.11 [Dependencies], page 25. In this case the effect is to effectively disable automatic dependency tracking.

`no-installinfo`

The generated '`Makefile.in`' will not cause info pages to be built or installed by default. However, `info` and `install-info` targets will still be available. This option is disallowed at '`GNU`' strictness and above.

`no-installman`

The generated '`Makefile.in`' will not cause man pages to be installed by default. However, an `install-man` target will still be available for optional installation. This option is disallowed at '`GNU`' strictness and above.

`no-texinfo.tex`

Don't require '`texinfo.tex`', even if there are texinfo files in this directory.

`readme-alpha`
: If this release is an alpha release, and the file '`README-alpha`' exists, then it will be added to the distribution. If this option is given, version numbers are expected to follow one of two forms. The first form is '*MAJOR.MINOR.ALPHA*', where each element is a number; the final period and number should be left off for non-alpha releases. The second form is '*MAJOR.MINORALPHA*', where *ALPHA* is a letter; it should be omitted for non-alpha releases.

version
: A version number (e.g. '`0.30`') can be specified. If Automake is not newer than the version specified, creation of the '`Makefile.in`' will be suppressed.

Unrecognized options are diagnosed by `automake`.

16 Miscellaneous Rules

There are a few rules and variables that didn't fit anywhere else.

16.1 Interfacing to `etags`

Automake will generate rules to generate '`TAGS`' files for use with GNU Emacs under some circumstances.

If any C, C++ or Fortran 77 source code or headers are present, then `tags` and `TAGS` targets will be generated for the directory.

At the topmost directory of a multi-directory package, a `tags` target file will be generated which, when run, will generate a '`TAGS`' file that includes by reference all '`TAGS`' files from subdirectories.

Also, if the variable `ETAGS_ARGS` is defined, a `tags` target will be generated. This variable is intended for use in directories which contain taggable source that `etags` does not understand.

Here is how Automake generates tags for its source, and for nodes in its Texinfo file:

```
ETAGS_ARGS = automake.in --lang=none \
 --regex='/^@node[ \t]+\([^,]+\)/\1/' automake.texi
```

If you add filenames to '`ETAGS_ARGS`', you will probably also want to set '`TAGS_DEPENDENCIES`'. The contents of this variable are added directly to the dependencies for the `tags` target.

Automake will also generate an `ID` target which will run `mkid` on the source. This is only supported on a directory-by-directory basis.

16.2 Handling new file extensions

It is sometimes useful to introduce a new implicit rule to handle a file type that Automake does not know about. If this is done, you must notify GNU Make of the new suffixes. This can be done by putting a list of new suffixes in the `SUFFIXES` variable.

For instance, currently Automake does not provide any Java support. If you wrote a macro to generate '`.class`' files from '`.java`' source files, you would also need to add these suffixes to the list:

```
SUFFIXES = .java .class
```

17 Include

To include another file (perhaps for common rules), the following syntax is supported:

include ($(srcdir)|$(top_srcdir))/filename

Using files in the current directory:

```
include $(srcdir)/Makefile.extra
include Makefile.generated
```

Using a file in the top level directory:

```
include $(top_srcdir)/filename
```

18 Conditionals

Automake supports a simple type of conditionals.

Before using a conditional, you must define it by using `AM_CONDITIONAL` in the `configure.in` file (see Section 5.4 [Macros], page 13). The `AM_CONDITIONAL` macro takes two arguments.

The first argument to `AM_CONDITIONAL` is the name of the conditional. This should be a simple string starting with a letter and containing only letters, digits, and underscores.

The second argument to `AM_CONDITIONAL` is a shell condition, suitable for use in a shell `if` statement. The condition is evaluated when `configure` is run.

Conditionals typically depend upon options which the user provides to the `configure` script. Here is an example of how to write a conditional which is true if the user uses the '`--enable-debug`' option.

```
AC_ARG_ENABLE(debug,
[  --enable-debug    Turn on debugging],
[case "${enableval}" in
  yes) debug=true ;;
  no)  debug=false ;;
  *) AC_MSG_ERROR(bad value ${enableval} for --enable-debug) ;;
esac],[debug=false])
AM_CONDITIONAL(DEBUG, test x$debug = xtrue)
```

Here is an example of how to use that conditional in '`Makefile.am`':

```
if DEBUG
DBG = debug
else
DBG =
endif
noinst_PROGRAMS = $(DBG)
```

This trivial example could also be handled using EXTRA_PROGRAMS (see Section 7.1 [A Program], page 16).

You may only test a single variable in an `if` statement. The `else` statement may be omitted. Conditionals may be nested to any depth.

Note that conditionals in Automake are not the same as conditionals in GNU Make. Automake conditionals are checked at configure time by the '`configure`' script, and affect

the translation from 'Makefile.in' to 'Makefile'. They are based on options passed to 'configure' and on results that 'configure' has discovered about the host system. GNU Make conditionals are checked at make time, and are based on variables passed to the make program or defined in the 'Makefile'.

Automake conditionals will work with any make program.

19 The effect of --gnu and --gnits

The '--gnu' option (or 'gnu' in the 'AUTOMAKE_OPTIONS' variable) causes automake to check the following:

- The files 'INSTALL', 'NEWS', 'README', 'COPYING', 'AUTHORS', and 'ChangeLog' are required at the topmost directory of the package.
- The options 'no-installman' and 'no-installinfo' are prohibited.

Note that this option will be extended in the future to do even more checking; it is advisable to be familiar with the precise requirements of the GNU standards. Also, '--gnu' can require certain non-standard GNU programs to exist for use by various maintainer-only targets; for instance in the future pathchk might be required for 'make dist'.

The '--gnits' option does everything that '--gnu' does, and checks the following as well:

- 'make dist' will check to make sure the 'NEWS' file has been updated to the current version.
- The file 'COPYING.LIB' is prohibited. The LGPL is apparently considered a failed experiment.
- 'VERSION' is checked to make sure its format complies with Gnits standards.
- If 'VERSION' indicates that this is an alpha release, and the file 'README-alpha' appears in the topmost directory of a package, then it is included in the distribution. This is done in '--gnits' mode, and no other, because this mode is the only one where version number formats are constrained, and hence the only mode where Automake can automatically determine whether 'README-alpha' should be included.
- The file 'THANKS' is required.

20 The effect of --cygnus

Cygnus Solutions has slightly different rules for how a 'Makefile.in' is to be constructed. Passing '--cygnus' to automake will cause any generated 'Makefile.in' to comply with Cygnus rules.

Here are the precise effects of '--cygnus':

- Info files are always created in the build directory, and not in the source directory.
- 'texinfo.tex' is not required if a Texinfo source file is specified. The assumption is that the file will be supplied, but in a place that Automake cannot find. This assumption is an artifact of how Cygnus packages are typically bundled.
- 'make dist' will look for files in the build directory as well as the source directory. This is required to support putting info files into the build directory.

- Certain tools will be searched for in the build tree as well as in the user's 'PATH'. These tools are `runtest`, `expect`, `makeinfo` and `texi2dvi`.
- `--foreign` is implied.
- The options 'no-installinfo' and 'no-dependencies' are implied.
- The macros 'AM_MAINTAINER_MODE' and 'AM_CYGWIN32' are required.
- The `check` target doesn't depend on `all`.

GNU maintainers are advised to use 'gnu' strictness in preference to the special Cygnus mode.

21 When Automake Isn't Enough

Automake's implicit copying semantics means that many problems can be worked around by simply adding some `make` targets and rules to 'Makefile.in'. Automake will ignore these additions.

There are some caveats to doing this. Although you can overload a target already used by Automake, it is often inadvisable, particularly in the topmost directory of a non-flat package. However, various useful targets have a '-local' version you can specify in your 'Makefile.in'. Automake will supplement the standard target with these user-supplied targets.

The targets that support a local version are `all`, `info`, `dvi`, `check`, `install-data`, `install-exec`, `uninstall`, and the various `clean` targets (`mostlyclean`, `clean`, `distclean`, and `maintainer-clean`). Note that there are no `uninstall-exec-local` or `uninstall-data-local` targets; just use `uninstall-local`. It doesn't make sense to uninstall just data or just executables.

For instance, here is one way to install a file in '/etc':

```
install-data-local:
        $(INSTALL_DATA) $(srcdir)/afile /etc/afile
```

Some targets also have a way to run another target, called a *hook*, after their work is done. The hook is named after the principal target, with '-hook' appended. The targets allowing hooks are `install-data`, `install-exec`, `dist`, and `distcheck`.

For instance, here is how to create a hard link to an installed program:

```
install-exec-hook:
        ln $(bindir)/program $(bindir)/proglink
```

22 Distributing 'Makefile.in's

Automake places no restrictions on the distribution of the resulting 'Makefile.in's. We still encourage software authors to distribute their work under terms like those of the GPL, but doing so is not required to use Automake.

Some of the files that can be automatically installed via the `--add-missing` switch do fall under the GPL; examine each file to see.

23 Some ideas for the future

Here are some things that might happen in the future:

- HTML support.
- The output will be cleaned up. For instance, only variables which are actually used will appear in the generated '`Makefile.in`'.
- There will be support for automatically recoding a distribution. The intent is to allow a maintainer to use whatever character set is most convenient locally, but for all distributions to be Unicode or ISO 10646 with the UTF-8 encoding.
- Rewrite in Guile. This won't happen in the near future, but it will eventually happen.

Macro and Variable Index

_

_LDADD ... 17
_LDFLAGS ... 17
_LIBADD ... 18
_SOURCES ... 16
_TEXINFOS ... 29

A

AC_ARG_PROGRAM ... 9
AC_CANONICAL_HOST ... 10
AC_CANONICAL_SYSTEM ... 10
AC_CHECK_PROG ... 12
AC_CHECK_PROGS ... 12
AC_CHECK_TOOL ... 10, 12
AC_CONFIG_AUX_DIR ... 10
AC_CONFIG_HEADER ... 10
AC_DECL_YYTEXT ... 11
AC_F77_LIBRARY_LDFLAGS ... 11
AC_FUNC_ALLOCA ... 11
AC_FUNC_FNMATCH ... 11
AC_FUNC_GETLOADAVG ... 11
AC_FUNC_MEMCMP ... 11
AC_OUTPUT ... 10
AC_PATH_PROG ... 12
AC_PATH_PROGS ... 12
AC_PATH_XTRA ... 10
AC_PROG_CXX ... 11
AC_PROG_F77 ... 11
AC_PROG_INSTALL ... 9
AC_PROG_LEX ... 12
AC_PROG_MAKE_SET ... 9
AC_PROG_RANLIB ... 11
AC_PROG_YACC ... 11
AC_REPLACE_FUNCS ... 11
AC_REPLACE_GNU_GETOPT ... 11
AC_STRUCT_ST_BLOCKS ... 11
AC_SUBST ... 12
ALL_LINGUAS ... 12
AM_C_PROTOTYPES ... 12, 14, 25
AM_CONDITIONAL ... 35
AM_CONFIG_HEADER ... 13
am_cv_sys_posix_termios ... 14
AM_FUNC_ERROR_AT_LINE ... 13
AM_FUNC_MKTIME ... 13
AM_FUNC_OBSTACK ... 14
AM_FUNC_STRTOD ... 11, 13
AM_GNU_GETTEXT ... 12
AM_HEADER_TIOCGWINSZ_NEEDS_SYS_IOCTL ... 14
AM_INIT_AUTOMAKE ... 9
AM_MAINTAINER_MODE ... 12
AM_PATH_LISPDIR ... 14
AM_PROG_LIBTOOL ... 11
AM_WITH_REGEX ... 11
AUTOMAKE_OPTIONS ... 25, 26, 32

B

bin_PROGRAMS ... 16
bin_SCRIPTS ... 27
build_alias ... 10
BUILT_SOURCES ... 27

C

check_LTLIBRARIES ... 18
CLEANFILES ... 31
COMPILE ... 19
CXX ... 21
CXXCOMPILE ... 21
CXXFLAGS ... 21
CXXLINK ... 21

D

DATA ... 4, 27
data_DATA ... 27
DEJATOOL ... 32
DESTDIR ... 31
DIST_SUBDIRS ... 31
DISTCLEANFILES ... 31

E

ELCFILES ... 28
ETAGS_ARGS ... 34
EXPECT ... 32
EXTRA_DIST ... 31
EXTRA_PROGRAMS ... 17

F

F77 ... 21
F77COMPILE ... 21
FFLAGS ... 21
FLINK ... 22

H

HAVE_PTRDIFF_T 14
HEADERS 4, 27
host_alias 10
host_triplet 10

I

include_HEADERS 27
INCLUDES 19
info_TEXINFOS 29

L

LDADD 17
LDFLAGS 19
lib_LIBRARIES 17
lib_LTLIBRARIES 18
LIBADD 18
libexec_PROGRAMS 16
libexec_SCRIPTS 27
LIBOBJS 11
LIBRARIES 4
LINK 19
LISP 4, 28
lisp_LISP 28
localstate_DATA 27

M

MAINTAINERCLEANFILES 31
man_MANS 30
MANS 4, 30
MOSTLYCLEANFILES 31

N

noinst_HEADERS 27
noinst_LIBRARIES 17
noinst_LISP 28
noinst_LTLIBRARIES 18
noinst_PROGRAMS 16
noinst_SCRIPTS 27

O

oldinclude_HEADERS 27
OMIT_DEPENDENCIES 26

P

PACKAGE 3, 9, 31
pkgdata_DATA 27
pkgdata_SCRIPTS 27
pkgdatadir 3
pkginclude_HEADERS 27
pkgincludedir 3
pkglib_LIBRARIES 17
pkglib_LTLIBRARIES 18
pkglib_PROGRAMS 16
pkglibdir 3
PROGRAMS 3, 4
ptrdiff_t 14

R

RFLAGS 21
RUNTEST 32
RUNTESTDEFAULTFLAGS 32
RUNTESTFLAGS 32

S

sbin_PROGRAMS 16
sbin_SCRIPTS 27
SCRIPTS 4, 26
sharedstate_DATA 27
SOURCES 16
SUBDIRS 2, 15
SUFFIXES 34
sysconf_DATA 27

T

TAGS_DEPENDENCIES 34
target_alias 10
TESTS 32
TESTS_ENVIRONMENT 32
TEXINFO_TEX 29
TEXINFOS 4, 29

V

VERSION 9, 31

W

WITH_DMALLOC 15
WITH_REGEX 15

Y

YACC 11

General Index

#

(special Automake comment) 2

-

--acdir 13
--add-missing 7
--amdir 7
--build-dir 8
--cygnus 8
–enable-debug, example 35
--enable-maintainer-mode 12
--foreign 8
--generate-deps 8
--gnits 8
–gnits, complete description 36
--gnu 8
–gnu, complete description 36
–gnu, required files 36
--help 8, 13
--include-deps 8
--no-force 8
--output 13
--output-dir 8
--print-ac-dir 13
--srcdir-name 9
--verbose 9, 13
--version 9, 13
--with-dmalloc 15
--with-regex 15
-a 7
-hook targets 37
-i 8
-I 13
-local targets 37
-o 8
-v 9

@

@ALLOCA@, special handling 18
@LIBOBJS@, special handling 18
@LTLIBOBJS@, special handling 18

_

_DATA primary, defined 27
_DEPENDENCIES, defined 17
_HEADERS primary, defined 27
_JAVA primary, defined 29
_LDFLAGS, defined 17
_LIBADD primary, defined 18
_LIBRARIES primary, defined 17
_LISP primary, defined 28
_LTLIBRARIES primary, defined 18
_MANS primary, defined 30
_PROGRAMS primary variable 3
_SCRIPTS primary, defined 26
_SOURCES and header files 16
_SOURCES primary, defined 16
_TEXINFOS primary, defined 29

A

AC_OUTPUT, scanning 10
acinclude.m4, defined 4
aclocal program, introduction 4
aclocal, extending 15
aclocal, Invoking 12
aclocal.m4, preexisting 4
Adding new SUFFIXES 34
all 37
all-local 37
AM_INIT_AUTOMAKE, example use 4
ansi2knr 25
Automake constraints 1
Automake options 7
Automake requirements 1, 9
Automake, invoking 7
Automake, recursive operation 2
Automatic linker selection 24

B

BUGS, reporting 1
BUILT_SOURCES, defined 27

C

C++ support ... 21
canonicalizing Automake macros ... 4
cfortran ... 22
`check` ... 37
check primary prefix, definition ... 4
`check-local` ... 37
check_LTLIBRARIES, not allowed ... 18
`clean-local` ... 37
Comment, special to Automake ... 2
Complete example ... 4
Conditional example, –enable-debug ... 35
Conditionals ... 35
config.guess ... 7
configure.in, from GNU Hello ... 5
configure.in, scanning ... 9
Constraints of Automake ... 1
cpio example ... 3
ctags Example ... 6
`cvs-dist` ... 1
cvs-dist, non-standard example ... 1
Cygnus strictness ... 36

D

DATA primary, defined ... 27
de-ANSI-fication, defined ... 25
Deep package ... 2
`dejagnu` ... 32
`dist` ... 26, 31
`dist-hook` ... 31, 37
`dist-shar` ... 33
`dist-tarZ` ... 33
`dist-zip` ... 33
`distcheck` ... 32
`distclean-local` ... 37
dmalloc, support for ... 15
`dvi` ... 37
`dvi-local` ... 37

E

E-mail, bug reports ... 1
EDITION Texinfo macro ... 29
`else` ... 35
`endif` ... 35
etags Example ... 6
Example conditional –enable-debug ... 35
Example of recursive operation ... 2
Example of shared libraries ... 18
Example, ctags and etags ... 6
Example, EXTRA_PROGRAMS ... 3
Example, GNU Hello ... 5
Example, handling Texinfo files ... 6
Example, mixed language ... 23
Example, regression test ... 6
Exit status 77, special interpretation ... 32
Extending aclocal ... 15
Extending list of installation directories ... 3
Extra files distributed with Automake ... 7
EXTRA_, prepending ... 3
EXTRA_prog_SOURCES, defined ... 16
EXTRA_PROGRAMS, defined ... 3, 17

F

Files distributed with Automake ... 7
First line of Makefile.am ... 2
Flat package ... 2
FLIBS, defined ... 23
foreign strictness ... 2
Fortran 77 support ... 21
Fortran 77, mixing with C and C++ ... 22
Fortran 77, Preprocessing ... 22
Future directions ... 38

G

Gettext support ... 28
gnits strictness ... 2
GNU Gettext support ... 28
GNU Hello, configure.in ... 5
GNU Hello, example ... 5
GNU make extensions ... 1
GNU Makefile standards ... 1
Guile rewrite ... 38

H

Header files in _SOURCES ... 16
HEADERS primary, defined ... 27
HEADERS, installation directories ... 27
Hello example ... 5
Hello, configure.in ... 5
hook targets ... 37
HP-UX 10, lex problems ... 14

HTML support, example 3

I

`id` 34
`if` 35
`include` 35
INCLUDES, example usage 6
`info` 33, 37
`info-local` 37
`install` 30
`install-data` 30, 37
`install-data-hook` 37
`install-data-local` 30, 37
`install-exec` 30, 37
`install-exec-hook` 37
`install-exec-local` 30, 37
`install-info` 29, 33
install-info target 29
`install-man` 30, 33
install-man target 30
`install-strip` 30
Installation directories, extending list 3
Installation support 30
`installdirs` 30
Installing headers 27
Installing scripts 27
Invoking aclocal 12
Invoking Automake 7

J

JAVA primary, defined 29
JAVA restrictions 29

L

lex problems with HP-UX 10 14
lex, multiple lexers 20
LIBADD primary, defined 18
LIBRARIES primary, defined 17
Linking Fortran 77 with C and C++ 22
LISP primary, defined 28
local targets 37
LTLIBRARIES primary, defined 18

M

Macros Automake recognizes 10
Macros, overriding 1
make check 32
make clean support 31
make dist 31
make distcheck 31
make install support 30
Make targets, overriding 1
Makefile.am, first line 2
MANS primary, defined 30
mdate-sh 29
Mixed language example 23
Mixing Fortran 77 with C and C++ 22
Mixing Fortran 77 with C and/or C++ 22
`mostlyclean-local` 37
Multiple configure.in files 7
Multiple lex lexers 20
Multiple yacc parsers 20

N

`no-dependencies` 26
`no-installinfo` 29
`no-installman` 30
`no-texinfo.tex` 29
noinst primary prefix, definition 4
noinstall-info target 29
noinstall-man target 30
Non-GNU packages 2
Non-standard targets 1

O

Option, ansi2knr 33
Option, check-news 33
Option, cygnus 33
Option, dejagnu 33
Option, dist-shar 33
Option, dist-tarZ 33
Option, dist-zip 33
Option, foreign 33
Option, gnits 33
Option, gnu 33
Option, no-dependencies 33
Option, no-installinfo 33
Option, no-installman 33
Option, no-texinfo 33

Option, readme-alpha 34
Option, version 34
Options, Automake 7
Overriding make macros 1
Overriding make targets 1
Overriding SUBDIRS 15

P

Package, deep 2
Package, Flat 2
Package, shallow 2
pkgdatadir, defined 3
pkgincludedir, defined 3
pkglibdir, defined 3
POSIX termios headers 14
Preprocessing Fortran 77 22
Primary variable, DATA 27
Primary variable, defined 3
Primary variable, HEADERS 27
Primary variable, JAVA 29
Primary variable, LIBADD 18
Primary variable, LIBRARIES 17
Primary variable, LISP 28
Primary variable, LTLIBRARIES 18
Primary variable, MANS 30
Primary variable, PROGRAMS 3
Primary variable, SCRIPTS 26
Primary variable, SOURCES 16
Primary variable, TEXINFOS 29
prog_LDADD, defined 17
PROGRAMS primary variable 3
PROGRAMS, bindir 16

R

Ratfor programs 22
README-alpha 36
Recognized macros by Automake 10
Recursive operation of Automake 2
regex package 15
Regression test example 6
Reporting BUGS 1
Requirements of Automake 9
Requirements, Automake 1
Restrictions for JAVA 29
rx package 15

S

Scanning configure.in 9
SCRIPTS primary, defined 26
SCRIPTS, installation directories 27
Selecting the linker automatically 24
Shallow package 2
Shared libraries, support for 18
SOURCES primary, defined 16
Special Automake comment 2
Strictness, defined 2
Strictness, foreign 2
Strictness, gnits 2
Strictness, gnu 2
SUBDIRS, deep package 2
SUBDIRS, explained 15
SUBDIRS, overriding 15
suffix .la, defined 18
suffix .lo, defined 18
SUFFIXES, adding 34
Support for C++ 21
Support for Fortran 77 21
Support for GNU Gettext 28

T

`tags` 34
TAGS support 34
Target, install-info 29
Target, install-man 30
Target, noinstall-info 29
Target, noinstall-man 30
termios POSIX headers 14
Test suites 32
Texinfo file handling example 6
Texinfo macro, EDITION 29
Texinfo macro, UPDATED 29
Texinfo macro, VERSION 29
texinfo.tex 29
TEXINFOS primary, defined 29

U

Uniform naming scheme 3
`uninstall` 30, 37
`uninstall-local` 37
UPDATED Texinfo macro 29

V

VERSION Texinfo macro 29

Y

yacc, multiple parsers 20
ylwrap 20

Z

zardoz example 4

Table of Contents

1 Introduction 1

2 General ideas 1

- 2.1 General Operation 1
- 2.2 Depth 2
- 2.3 Strictness 2
- 2.4 The Uniform Naming Scheme 3
- 2.5 How derived variables are named 4

3 Some example packages 4

- 3.1 A simple example, start to finish 4
- 3.2 A classic program 5
- 3.3 Building etags and ctags 6

4 Creating a 'Makefile.in' 7

5 Scanning 'configure.in' 9

- 5.1 Configuration requirements 9
- 5.2 Other things Automake recognizes 10
- 5.3 Auto-generating aclocal.m4 12
- 5.4 Autoconf macros supplied with Automake 13
- 5.5 Writing your own aclocal macros 15

6 The top-level 'Makefile.am' 15

7 Building Programs and Libraries 16

- 7.1 Building a program 16
- 7.2 Building a library 17
- 7.3 Special handling for LIBOBJS and ALLOCA 18
- 7.4 Building a Shared Library 18
- 7.5 Variables used when building a program 19
- 7.6 Yacc and Lex support 19
- 7.7 C++ Support 21
- 7.8 Fortran 77 Support 21
  - 7.8.1 Preprocessing Fortran 77 22
  - 7.8.2 Compiling Fortran 77 Files 22
  - 7.8.3 Mixing Fortran 77 With C and C++ 22
    - 7.8.3.1 How the Linker is Chosen 24
  - 7.8.4 Fortran 77 and Autoconf 24
- 7.9 Support for Other Languages 25
- 7.10 Automatic de-ANSI-fication 25
- 7.11 Automatic dependency tracking 25

8 Other Derived Objects 26
8.1 Executable Scripts 26
8.2 Header files 27
8.3 Architecture-independent data files 27
8.4 Built sources 27

9 Other GNU Tools 28
9.1 Emacs Lisp 28
9.2 Gettext 28
9.3 Guile 28
9.4 Libtool 28
9.5 Java 29

10 Building documentation 29
10.1 Texinfo 29
10.2 Man pages 30

11 What Gets Installed 30

12 What Gets Cleaned 31

13 What Goes in a Distribution 31

14 Support for test suites 32

15 Changing Automake's Behavior 32

16 Miscellaneous Rules 34
16.1 Interfacing to `etags` 34
16.2 Handling new file extensions 34

17 Include 35

18 Conditionals 35

19 The effect of `--gnu` and `--gnits` 36

20 The effect of `--cygnus` 36

21 When Automake Isn't Enough 37

22 Distributing 'Makefile.in's 37

23 Some ideas for the future 38

Macro and Variable Index 39

General Index 42