# Autoconf

Creating Automatic Configuration Scripts
Edition 2.8, for Autoconf version 2.8
January 1996

by David MacKenzie

# 1 Introduction

> A physicist, an engineer, and a computer scientist were discussing the nature of God. Surely a Physicist, said the physicist, because early in the Creation, God made Light; and you know, Maxwell's equations, the dual nature of electro-magnetic waves, the relativist consequences... An Engineer!, said the engineer, because before making Light, God split the Chaos into Land and Water; it takes a hell of an engineer to handle that big amount of mud, and orderly separation of solids from liquids... The computer scientist shouted: And the Chaos, where do you think it was coming from, hmm?
>
> —Anonymous

Autoconf is a tool for producing shell scripts that automatically configure software source code packages to adapt to many kinds of UNIX-like systems. The configuration scripts produced by Autoconf are independent of Autoconf when they are run, so their users do not need to have Autoconf.

The configuration scripts produced by Autoconf require no manual user intervention when run; they do not normally even need an argument specifying the system type. Instead, they test for the presence of each feature that the software package they are for might need individually. (Before each check, they print a one-line message stating what they are checking for, so the user doesn't get too bored while waiting for the script to finish.) As a result, they deal well with systems that are hybrids or customized from the more common UNIX variants. There is no need to maintain files that list the features supported by each release of each variant of UNIX.

For each software package that Autoconf is used with, it creates a configuration script from a template file that lists the system features that the package needs or can use. After the shell code to recognize and respond to a system feature has been written, Autoconf allows it to be shared by many software packages that can use (or need) that feature. If it later turns out that the shell code needs adjustment for some reason, it needs to be changed in only one place; all of the configuration scripts can be regenerated automatically to take advantage of the updated code.

The Metaconfig package is similar in purpose to Autoconf, but the scripts it produces require manual user intervention, which is quite inconvenient when configuring large source trees. Unlike Metaconfig scripts, Autoconf scripts can support cross-compiling, if some care is taken in writing them.

There are several jobs related to making portable software packages that Autoconf currently does not do. Among these are automatically creating 'Makefile' files with all of the standard targets, and supplying replacements for standard library functions and header files on systems that lack them. Work is in progress to add those features in the future.

Autoconf imposes some restrictions on the names of macros used with `#ifdef` in C programs (see [Preprocessor Symbol Index], page 102).

Autoconf requires GNU `m4` in order to generate the scripts. It uses features that some UNIX versions of `m4` do not have. It also overflows internal limits of some versions of `m4`, including GNU `m4` 1.0. You must use version 1.1 or later of GNU `m4`. Using version 1.3 or later will be much faster than 1.1 or 1.2.

See Chapter 13 [Upgrading], page 87, for information about upgrading from version 1. See Chapter 14 [History], page 92, for the story of Autoconf's development. See Chapter 12 [Questions], page 83, for answers to some common questions about Autoconf.

Mail suggestions and bug reports for Autoconf to `bug-gnu-utils@prep.ai.mit.edu`. Please include the Autoconf version number, which you can get by running '`autoconf --version`'.

# 2 Making configure Scripts

The configuration scripts that Autoconf produces are by convention called `configure`. When run, `configure` creates several files, replacing configuration parameters in them with appropriate values. The files that `configure` creates are:

- one or more '`Makefile`' files, one in each subdirectory of the package (see Section 3.3 [Makefile Substitutions], page 12);
- optionally, a C header file, the name of which is configurable, containing `#define` directives (see Section 3.4 [Configuration Headers], page 17);
- a shell script called '`config.status`' that, when run, will recreate the files listed above (see Chapter 11 [Invoking config.status], page 81);
- a shell script called '`config.cache`' that saves the results of running many of the tests (see Section 6.3.2 [Cache Files], page 59);
- a file called '`config.log`' containing any messages produced by compilers, to help debugging if `configure` makes a mistake.

To create a `configure` script with Autoconf, you need to write an Autoconf input file '`configure.in`' and run `autoconf` on it. If you write your own feature tests to supplement those that come with Autoconf, you might also write files called '`aclocal.m4`' and '`acsite.m4`'. If you use a C header file to contain `#define` directives, you might also write '`acconfig.h`', and you will distribute the Autoconf-generated file '`config.h.in`' with the package.

Here is a diagram showing how the files that can be used in configuration are produced. Programs that are executed are suffixed by '`*`'. Optional files are enclosed in square brackets ('`[]`'). `autoconf` and `autoheader` also read the installed Autoconf macro files (by reading '`autoconf.m4`').

Files used in preparing a software package for distribution:

```
your source files --> [autoscan*] --> [configure.scan] --> configure.in

configure.in --.   .------> autoconf* -----> configure
               +---+
[aclocal.m4] --+   `---.
[acsite.m4] ---'       |
                       +--> [autoheader*] -> [config.h.in]
[acconfig.h] ----.     |
                 +-----'
[config.h.top] --+
[config.h.bot] --'

Makefile.in -------------------------------> Makefile.in
```

Files used in configuring a software package:

```
                       .-------------> config.cache
configure* ------------+-------------> config.log
                       |
[config.h.in] -.       v            .-> [config.h] -.
               +--> config.status* -+               +--> make*
Makefile.in ---'                    `-> Makefile ---'
```

## 2.1 Writing 'configure.in'

To produce a configure script for a software package, create a file called 'configure.in' that contains invocations of the Autoconf macros that test the system features your package needs or can use. Autoconf macros already exist to check for many features; see Chapter 4 [Existing Tests], page 23, for their descriptions. For most other features, you can use Autoconf template macros to produce custom checks; see Chapter 5 [Writing Tests], page 45, for information about them. For especially tricky or specialized features, 'configure.in' might need to contain some hand-crafted shell commands. The autoscan program can give you a good start in writing 'configure.in' (see Section 2.2 [Invoking autoscan], page 6, for more information).

The order in which 'configure.in' calls the Autoconf macros is not important, with a few exceptions. Every 'configure.in' must contain a call to AC_INIT before the checks, and a call to AC_OUTPUT at the end (see Section 3.2 [Output], page 11). Additionally, some macros rely on other macros having been called first, because they check previously set values of some variables to decide what to do. These macros are noted in the individual descriptions (see Chapter 4 [Existing Tests], page 23), and they also warn you when creating configure if they are called out of order.

To encourage consistency, here is a suggested order for calling the Autoconf macros. Generally speaking, the things near the end of this list could depend on things earlier in it. For example, library functions could be affected by typedefs and libraries.

`AC_INIT(`*file*`)`
checks for programs
checks for libraries
checks for header files
checks for typedefs
checks for structures
checks for compiler characteristics
checks for library functions
checks for system services
`AC_OUTPUT(`[*file...*]`)`

It is best to put each macro call on its own line in '`configure.in`'. Most of the macros don't add extra newlines; they rely on the newline after the macro call to terminate the commands. This approach makes the generated `configure` script a little easier to read by not inserting lots of blank lines. It is generally safe to set shell variables on the same line as a macro call, because the shell allows assignments without intervening newlines.

When calling macros that take arguments, there must not be any blank space between the macro name and the open parenthesis. Arguments can be more than one line long if they are enclosed within the `m4` quote characters '`[`' and '`]`'. If you have a long line such as a list of file names, you can generally use a backslash at the end of a line to continue it logically on the next line (this is implemented by the shell, not by anything special that Autoconf does).

Some macros handle two cases: what to do if the given condition is met, and what to do if the condition is not met. In some places you might want to do something if a condition is true but do nothing if it's false, or vice versa. To omit the true case, pass an empty value for the *action-if-found* argument to the macro. To omit the false case, omit the *action-if-not-found* argument to the macro, including the comma before it.

You can include comments in '`configure.in`' files by starting them with the `m4` builtin macro `dnl`, which discards text up through the next newline. These comments do not appear in the generated `configure` scripts. For example, it is helpful to begin '`configure.in`' files with a line like this:

```
dnl Process this file with autoconf to produce a configure script.
```

## 2.2 Using autoscan to Create 'configure.in'

The autoscan program can help you create a 'configure.in' file for a software package. autoscan examines source files in the directory tree rooted at a directory given as a command line argument, or the current directory if none is given. It searches the source files for common portability problems and creates a file 'configure.scan' which is a preliminary 'configure.in' for that package.

You should manually examine 'configure.scan' before renaming it to 'configure.in'; it will probably need some adjustments. Occasionally autoscan outputs a macro in the wrong order relative to another macro, so that autoconf produces a warning; you need to move such macros manually. Also, if you want the package to use a configuration header file, you must add a call to AC_CONFIG_HEADER (see Section 3.4 [Configuration Headers], page 17). You might also have to change or add some #if directives to your program in order to make it work with Autoconf (see Section 2.3 [Invoking ifnames], page 7, for information about a program that can help with that job).

autoscan uses several data files, which are installed along with the distributed Autoconf macro files, to determine which macros to output when it finds particular symbols in a package's source files. These files all have the same format. Each line consists of a symbol, whitespace, and the Autoconf macro to output if that symbol is encountered. Lines starting with '#' are comments.

autoscan is only installed if you already have Perl installed. autoscan accepts the following options:

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

--macrodir=*dir*
: Look for the data files in directory *dir* instead of the default installation directory. You can also set the AC_MACRODIR environment variable to a directory; this option overrides the environment variable.

--verbose
: Print the names of the files it examines and the potentially interesting symbols it finds in them. This output can be voluminous.

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

## 2.3 Using `ifnames` to List Conditionals

`ifnames` can help when writing a '`configure.in`' for a software package. It prints the identifiers that the package already uses in C preprocessor conditionals. If a package has already been set up to have some portability, this program can help you figure out what its `configure` needs to check for. It may help fill in some gaps in a '`configure.in`' generated by `autoscan` (see Section 2.2 [Invoking autoscan], page 6).

`ifnames` scans all of the C source files named on the command line (or the standard input, if none are given) and writes to the standard output a sorted list of all the identifiers that appear in those files in `#if`, `#elif`, `#ifdef`, or `#ifndef` directives. It prints each identifier on a line, followed by a space-separated list of the files in which that identifier occurs.

`ifnames` accepts the following options:

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

`--macrodir=`*dir*
`-m` *dir*
: Look for the Autoconf macro files in directory *dir* instead of the default installation directory. Only used to get the version number. You can also set the `AC_MACRODIR` environment variable to a directory; this option overrides the environment variable.

`--version`
: Print the version number of Autoconf and exit.

## 2.4 Using `autoconf` to Create `configure`

To create `configure` from '`configure.in`', run the `autoconf` program with no arguments. `autoconf` processes '`configure.in`' with the `m4` macro processor, using the Autoconf macros. If you give `autoconf` an argument, it reads that file instead of '`configure.in`' and writes the configuration script to the standard output instead of to `configure`. If you give `autoconf` the argument '`-`', it reads the standard input instead of '`configure.in`' and writes the configuration script on the standard output.

The Autoconf macros are defined in several files. Some of the files are distributed with Autoconf; `autoconf` reads them first. Then it looks for the optional file '`acsite.m4`' in the directory that contains the distributed Autoconf macro files, and for the optional file '`aclocal.m4`' in the current directory. Those files can contain your site's or the package's own Autoconf macro definitions (see

Chapter 7 [Writing Macros], page 62, for more information). If a macro is defined in more than one of the files that `autoconf` reads, the last definition it reads overrides the earlier ones.

`autoconf` accepts the following options:

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

`--localdir=`*dir*
`-l` *dir* Look for the package file '`aclocal.m4`' in directory *dir* instead of in the current directory.

`--macrodir=`*dir*
`-m` *dir* Look for the installed macro files in directory *dir*. You can also set the `AC_MACRODIR` environment variable to a directory; this option overrides the environment variable.

`--version`
Print the version number of Autoconf and exit.

## 2.5 Using autoreconf to Update configure Scripts

If you have a lot of Autoconf-generated `configure` scripts, the `autoreconf` program can save you some work. It runs `autoconf` (and `autoheader`, where appropriate) repeatedly to remake the Autoconf `configure` scripts and configuration header templates in the directory tree rooted at the current directory. By default, it only remakes those files that are older than their '`configure.in`' or (if present) '`aclocal.m4`'. Since `autoheader` does not change the timestamp of its output file if the file wouldn't be changing, this is not necessarily the minimum amount of work. If you install a new version of Autoconf, you can make `autoreconf` remake *all* of the files by giving it the '`--force`' option.

If you give `autoreconf` the '`--macrodir=`*dir*' or '`--localdir=`*dir*' options, it passes them down to `autoconf` and `autoheader` (with relative paths adjusted properly).

See Section 3.3.3 [Automatic Remaking], page 16, for '`Makefile`' rules to automatically remake `configure` scripts when their source files change. That method handles the timestamps of configuration header templates properly, but does not pass '`--macrodir=`*dir*' or '`--localdir=`*dir*'.

`autoreconf` accepts the following options:

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

`--force`
`-f` Remake even '`configure`' scripts and configuration headers that are newer than their input files ('`configure.in`' and, if present, '`aclocal.m4`').

`--localdir=`*dir*
`-l` *dir* Look for the package files '`aclocal.m4`' and '`acconfig.h`' (but not '*file*`.top`' and '*file*`.bot`') in directory *dir* instead of in the directory containing each '`configure.in`'.

`--macrodir=`*dir*
`-m` *dir* Look for the Autoconf macro files in directory *dir* instead of the default installation directory. You can also set the `AC_MACRODIR` environment variable to a directory; this option overrides the environment variable.

`--verbose`
Print the name of each directory where `autoreconf` runs `autoconf` (and `autoheader`, if appropriate).

`--version`
Print the version number of Autoconf and exit.

# 3 Initialization and Output Files

Autoconf-generated `configure` scripts need some information about how to initialize, such as how to find the package's source files; and about the output files to produce. The following sections describe initialization and creating output files.

## 3.1 Finding `configure` Input

Every `configure` script must call `AC_INIT` before doing anything else. The only other required macro is `AC_OUTPUT` (see Section 3.2 [Output], page 11).

**AC_INIT** (*unique-file-in-source-dir*) Macro

Process any command-line arguments and find the source code directory. *unique-file-in-source-dir* is some file that is in the package's source directory; `configure` checks for this file's existence to make sure that the directory that it is told contains the source code in fact does. Occasionally people accidentally specify the wrong directory with '`--srcdir`'; this is a safety check. See Chapter 10 [Invoking configure], page 77, for more information.

Packages that do manual configuration or use the `install` program might need to tell `configure` where to find some other shell scripts by calling `AC_CONFIG_AUX_DIR`, though the default places it looks are correct for most cases.

**AC_CONFIG_AUX_DIR**(*dir*) Macro

Use the '`install-sh`', '`config.sub`', '`config.guess`', and Cygnus `configure` scripts that are in directory *dir*. These are auxiliary files used in configuration. *dir* can be either absolute or relative to '*srcdir*'. The default is '*srcdir*' or '*srcdir*`/..`' or '*srcdir*`/../..`', whichever is the first that contains '`install-sh`'. The other files are not checked for, so that using `AC_PROG_INSTALL` does not automatically require distributing the other auxiliary files. It checks for '`install.sh`' also, but that name is obsolete because some `make` programs have a rule that creates '`install`' from it if there is no '`Makefile`'.

## 3.2 Creating Output Files

Every Autoconf-generated `configure` script must finish by calling `AC_OUTPUT`. It is the macro that creates the '`Makefile`'s and optional other files resulting from configuration. The only other required macro is `AC_INIT` (see Section 3.1 [Input], page 10).

**AC_OUTPUT** ([*file*. . .] [,*extra-cmds*] [,*init-cmds*]) Macro

Create output files. The *file*. . . argument is a whitespace-separated list of output files; it may be empty. This macro creates each file '*file*' by copying an input file (by default named '*file*`.in`'), substituting the output variable values. See Section 3.3 [Makefile Substitutions], page 12, for more information on using output variables. See Section 6.2 [Setting Output Variables], page 56, for more information on creating them. This macro creates the directory that the file is in if it doesn't exist (but not the parents of that directory). Usually, '`Makefile`'s are created this way, but other files, such as '`.gdbinit`', can be specified as well.

If `AC_CONFIG_HEADER`, `AC_LINK_FILES`, or `AC_CONFIG_SUBDIRS` has been called, this macro also creates the files named as their arguments.

A typical call to `AC_OUTPUT` looks like this:

```
AC_OUTPUT(Makefile src/Makefile man/Makefile X/Imakefile)
```

You can override an input file name by appending it to *file*, separated by a colon. For example,

```
AC_OUTPUT(Makefile:templates/top.mk lib/Makefile:templates/lib.mk)
```

If you pass *extra-cmds*, those commands will be inserted into '`config.status`' to be run after all its other processing. If *init-cmds* are given, they are inserted just before *extra-cmds*, with shell variable, command, and backslash substitutions performed on them in `configure`. You can use *init-cmds* to pass variables from `configure` to the *extra-cmds*.

If you run `make` on subdirectories, you should run it using the `make` variable `MAKE`. Most versions of `make` set `MAKE` to the name of the `make` program plus any options it was given. (But many do not include in it the values of any variables set on the command line, so those are not passed on automatically.) Some old versions of `make` do not set this variable. The following macro allows you to use it even with those versions.

**AC_PROG_MAKE_SET** Macro

If `make` predefines the variable `MAKE`, define output variable `SET_MAKE` to be empty. Otherwise, define `SET_MAKE` to contain '`MAKE=make`'. Calls `AC_SUBST` for `SET_MAKE`.

To use this macro, place a line like this in each '`Makefile.in`' that runs `MAKE` on other directories:

```
@SET_MAKE@
```

## 3.3 Substitutions in Makefiles

Each subdirectory in a distribution that contains something to be compiled or installed should come with a file '`Makefile.in`', from which `configure` will create a '`Makefile`' in that directory. To create a '`Makefile`', `configure` performs a simple variable substitution, replacing occurrences of '`@`*variable*`@`' in '`Makefile.in`' with the value that `configure` has determined for that variable. Variables that are substituted into output files in this way are called *output variables*. They are ordinary shell variables that are set in `configure`. To make `configure` substitute a particular variable into the output files, the macro `AC_SUBST` must be called with that variable name as an argument. Any occurrences of '`@`*variable*`@`' for other variables are left unchanged. See Section 6.2 [Setting Output Variables], page 56, for more information on creating output variables with `AC_SUBST`.

A software package that uses a `configure` script should be distributed with a file '`Makefile.in`', but no '`Makefile`'; that way, the user has to properly configure the package for the local system before compiling it.

See section "Makefile Conventions" in *The GNU Coding Standards*, for more information on what to put in '`Makefile`'s.

### 3.3.1 Preset Output Variables

Some output variables are preset by the Autoconf macros. Some of the Autoconf macros set additional output variables, which are mentioned in the descriptions for those macros. See [Output Variable Index], page 101, for a complete list of output variables. Here is what each of the preset ones contains. See section "Variables for Installation Directories" in *The GNU Coding Standards*, for more information about the variables with names that end in 'dir'.

**bindir** Variable

The directory for installing executables that users run.

**configure_input** Variable

A comment saying that the file was generated automatically by `configure` and giving the name of the input file. `AC_OUTPUT` adds a comment line containing this variable to the top of every 'Makefile' it creates. For other files, you should reference this variable in a comment at the top of each input file. For example, an input shell script should begin like this:

```
#! /bin/sh
# @configure_input@
```

The presence of that line also reminds people editing the file that it needs to be processed by `configure` in order to be used.

**datadir** Variable

The directory for installing read-only architecture-independent data.

**exec_prefix** Variable

The installation prefix for architecture-dependent files.

**includedir** Variable

The directory for installing C header files.

**infodir** Variable

The directory for installing documentation in Info format.

**libdir** Variable

The directory for installing object code libraries.

**libexecdir** Variable

The directory for installing executables that other programs run.

**localstatedir** Variable

The directory for installing modifiable single-machine data.

**mandir** Variable

The top-level directory for installing documentation in man format.

**oldincludedir** Variable

The directory for installing C header files for non-gcc compilers.

**prefix** Variable

The installation prefix for architecture-independent files.

**sbindir** Variable

The directory for installing executables that system administrators run.

**sharedstatedir** Variable

The directory for installing modifiable architecture-independent data.

**srcdir** Variable

The directory that contains the source code for that '`Makefile`'.

**sysconfdir** Variable

The directory for installing read-only single-machine data.

**top_srcdir** Variable

The top-level source code directory for the package. In the top-level directory, this is the same as `srcdir`.

**CFLAGS** Variable

Debugging and optimization options for the C compiler. If it is not set in the environment when `configure` runs, the default value is set when you call `AC_PROG_CC` (or empty if you don't). `configure` uses this variable when compiling programs to test for C features.

**CPPFLAGS** Variable

Header file search directory ('-I*dir*') and any other miscellaneous options for the C preprocessor and compiler. If it is not set in the environment when `configure` runs, the default value is empty. `configure` uses this variable when compiling or preprocessing programs to test for C features.

**CXXFLAGS** Variable

Debugging and optimization options for the C++ compiler. If it is not set in the environment when `configure` runs, the default value is set when you call `AC_PROG_CXX` (or empty if you don't). `configure` uses this variable when compiling programs to test for C++ features.

**DEFS** Variable

'-D' options to pass to the C compiler. If `AC_CONFIG_HEADER` is called, `configure` replaces '@DEFS@' with '-DHAVE_CONFIG_H' instead (see Section 3.4 [Configuration Headers], page 17). This variable is not defined while `configure` is performing its tests, only when creating the output files. See Section 6.2 [Setting Output Variables], page 56, for how to check the results of previous tests.

**LDFLAGS** Variable

Stripping ('-s') and any other miscellaneous options for the linker. If it is not set in the environment when `configure` runs, the default value is empty. `configure` uses this variable when linking programs to test for C features.

**LIBS** Variable

'-l' and '-L' options to pass to the linker.

### 3.3.2 Build Directories

You can support compiling a software package for several architectures simultaneously from the same copy of the source code. The object files for each architecture are kept in their own directory.

To support doing this, `make` uses the `VPATH` variable to find the files that are in the source directory. GNU `make` and most other recent `make` programs can do this. Older `make` programs do not support `VPATH`; when using them, the source code must be in the same directory as the object files.

To support VPATH, each 'Makefile.in' should contain two lines that look like:

```
srcdir = @srcdir@
VPATH = @srcdir@
```

Do not set VPATH to the value of another variable, for example 'VPATH = $(srcdir)', because some versions of make do not do variable substitutions on the value of VPATH.

configure substitutes in the correct value for srcdir when it produces 'Makefile'.

Do not use the make variable $<, which expands to the pathname of the file in the source directory (found with VPATH), except in implicit rules. (An implicit rule is one such as '.c.o', which tells how to create a '.o' file from a '.c' file.) Some versions of make do not set $< in explicit rules; they expand it to an empty value.

Instead, 'Makefile' command lines should always refer to source files by prefixing them with '$(srcdir)/'. For example:

```
time.info: time.texinfo
        $(MAKEINFO) $(srcdir)/time.texinfo
```

### 3.3.3 Automatic Remaking

You can put rules like the following in the top-level 'Makefile.in' for a package to automatically update the configuration information when you change the configuration files. This example includes all of the optional files, such as 'aclocal.m4' and those related to configuration header files. Omit from the 'Makefile.in' rules any of these files that your package does not use.

The '${srcdir}/' prefix is included because of limitations in the VPATH mechanism.

The 'stamp-' files are necessary because the timestamps of 'config.h.in' and 'config.h' will not be changed if remaking them does not change their contents. This feature avoids unnecessary recompilation. You should include the file 'stamp-h.in' your package's distribution, so make will consider 'config.h.in' up to date. On some old BSD systems, touch or any command that results in an empty file does not update the timestamps, so use a command like echo as a workaround.

```
${srcdir}/configure: configure.in aclocal.m4
        cd ${srcdir} && autoconf

# autoheader might not change config.h.in, so touch a stamp file.
${srcdir}/config.h.in: stamp-h.in
${srcdir}/stamp-h.in: configure.in aclocal.m4 acconfig.h \
    config.h.top config.h.bot
        cd ${srcdir} && autoheader
        echo timestamp > ${srcdir}/stamp-h.in

config.h: stamp-h
stamp-h: config.h.in config.status
        ./config.status

Makefile: Makefile.in config.status
        ./config.status

config.status: configure
        ./config.status --recheck
```

In addition, you should pass 'echo timestamp > stamp-h' in the *extra-cmds* argument to AC_OUTPUT, so 'config.status' will ensure that 'config.h' is considered up to date. See Section 3.2 [Output], page 11, for more information about AC_OUTPUT.

See Chapter 11 [Invoking config.status], page 81, for more examples of handling configuration-related dependencies.

## 3.4 Configuration Header Files

When a package tests more than a few C preprocessor symbols, the command lines to pass '-D' options to the compiler can get quite long. This causes two problems. One is that the make output is hard to visually scan for errors. More seriously, the command lines can exceed the length limits of some operating systems. As an alternative to passing '-D' options to the compiler, configure scripts can create a C header file containing '#define' directives. The AC_CONFIG_HEADER macro selects this kind of output. It should be called right after AC_INIT.

The package should '#include' the configuration header file before any other header files, to prevent inconsistencies in declarations (for example, if it redefines const). Use '#include <config.h>' instead of '#include "config.h"', and pass the C compiler a '-I.' option (or '-I..'; whichever directory contains 'config.h'). That way, even if the source directory is configured itself (perhaps to make a distribution), other build directories can also be configured without finding the '`config.h`' from the source directory.

**AC_CONFIG_HEADER** (*header-to-create* ...) Macro

Make `AC_OUTPUT` create the file(s) in the whitespace-separated list *header-to-create* containing C preprocessor `#define` statements, and replace '`@DEFS@`' in generated files with '`-DHAVE_CONFIG_H`' instead of the value of `DEFS`. The usual name for *header-to-create* is '`config.h`'.

If *header-to-create* already exists and its contents are identical to what `AC_OUTPUT` would put in it, it is left alone. Doing this allows some changes in configuration without needlessly causing object files that depend on the header file to be recompiled.

Usually the input file is named '*header-to-create*`.in`'; however, you can override the input file name by appending it to *header-to-create*, separated by a colon. For example,

```
AC_CONFIG_HEADER(defines.h:defines.hin)
```

Doing this allows you to keep your filenames acceptable to MS-DOS.

### 3.4.1 Configuration Header Templates

Your distribution should contain a template file that looks as you want the final header file to look, including comments, with default values in the `#define` statements. For example, suppose your '`configure.in`' makes these calls:

```
AC_CONFIG_HEADER(conf.h)
AC_CHECK_HEADERS(unistd.h)
```

Then you could have code like the following in '`conf.h.in`'. On systems that have '`unistd.h`', `configure` will change the 0 to a 1. On other systems, it will leave the line unchanged.

```
/* Define as 1 if you have unistd.h.  */
#define HAVE_UNISTD_H 0
```

Alternately, if your code tests for configuration options using `#ifdef` instead of `#if`, a default value can be to `#undef` the variable instead of to define it to a value. On systems that have 'unistd.h', configure will change the second line to read '#define HAVE_UNISTD_H 1'. On other systems, it will comment that line out (in case the system predefines that symbol).

```
/* Define if you have unistd.h.  */
#undef HAVE_UNISTD_H
```

### 3.4.2 Using autoheader to Create 'config.h.in'

The autoheader program can create a template file of C '#define' statements for configure to use. If 'configure.in' invokes AC_CONFIG_HEADER(*file*), autoheader creates '*file*.in'; if multiple file arguments are given, the first one is used. Otherwise, autoheader creates 'config.h.in'.

If you give autoheader an argument, it uses that file instead of 'configure.in' and writes the header file to the standard output instead of to 'config.h.in'. If you give autoheader an argument of '-', it reads the standard input instead of 'configure.in' and writes the header file to the standard output.

autoheader scans 'configure.in' and figures out which C preprocessor symbols it might define. It copies comments and #define and #undef statements from a file called 'acconfig.h', which comes with and is installed with Autoconf. It also uses a file called 'acconfig.h' in the current directory, if present. If you AC_DEFINE any additional symbols, you must create that file with entries for them. For symbols defined by AC_CHECK_HEADERS, AC_CHECK_FUNCS, AC_CHECK_SIZEOF, or AC_CHECK_LIB, autoheader generates comments and #undef statements itself rather than copying them from a file, since the possible symbols are effectively limitless.

The file that autoheader creates contains mainly #define and #undef statements and their accompanying comments. If './acconfig.h' contains the string '@TOP@', autoheader copies the lines before the line containing '@TOP@' into the top of the file that it generates. Similarly, if './acconfig.h' contains the string '@BOTTOM@', autoheader copies the lines after that line to the end of the file it generates. Either or both of those strings may be omitted.

An alternate way to produce the same effect is to create the files '*file*.top' (typically 'config.h.top') and/or '*file*.bot' in the current directory. If they exist, autoheader copies them to the beginning and end, respectively, of its output. Their use is discouraged because they have file names that contain two periods, and so can not be stored on MS-DOS; also, they are two more files to clutter up the directory. But if you use the '--localdir=*dir*' option to use an 'acconfig.h' in another directory, they give you a way to put custom boilerplate in each individual 'config.h.in'.

`autoheader` accepts the following options:

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

`--localdir=`*dir*
`-l` *dir*
: Look for the package files '`aclocal.m4`' and '`acconfig.h`' (but not '*file*`.top`' and '*file*`.bot`') in directory *dir* instead of in the current directory.

`--macrodir=`*dir*
`-m` *dir*
: Look for the installed macro files and '`acconfig.h`' in directory *dir*. You can also set the `AC_MACRODIR` environment variable to a directory; this option overrides the environment variable.

`--version`
: Print the version number of Autoconf and exit.

## 3.5 Configuring Other Packages in Subdirectories

In most situations, calling `AC_OUTPUT` is sufficient to produce '`Makefile`'s in subdirectories. However, `configure` scripts that control more than one independent package can use `AC_CONFIG_SUBDIRS` to run `configure` scripts for other packages in subdirectories.

**AC_CONFIG_SUBDIRS** (*dir* ...) Macro

Make `AC_OUTPUT` run `configure` in each subdirectory *dir* in the given whitespace-separated list. If a given *dir* is not found, no error is reported, so a `configure` script can configure whichever parts of a large source tree are present. If a given *dir* contains '`configure.in`' but no `configure`, the Cygnus `configure` script found by `AC_CONFIG_AUXDIR` is used. The subdirectory `configure` scripts are given the same command line options that were given to this `configure` script, with minor changes if needed (e.g., to adjust a relative path for the cache file or source directory). This macro also sets the output variable `subdirs` to the list of directories '*dir* ...'. '`Makefile`' rules can use this variable to determine which subdirectories to recurse into.

## 3.6 Default Prefix

By default, `configure` sets the prefix for files it installs to '`/usr/local`'. The user of `configure` can select a different prefix using the '`--prefix`' and '`--exec-prefix`' options. There are two ways to change the default: when creating `configure`, and when running it.

Some software packages might want to install in a directory besides '`/usr/local`' by default. To accomplish that, use the `AC_PREFIX_DEFAULT` macro.

**AC_PREFIX_DEFAULT** (*prefix*) Macro

Set the default installation prefix to *prefix* instead of '`/usr/local`'.

It may be convenient for users to have `configure` guess the installation prefix from the location of a related program that they have already installed. If you wish to do that, you can call `AC_PREFIX_PROGRAM`.

**AC_PREFIX_PROGRAM** (*program*) Macro

If the user did not specify an installation prefix (using the '`--prefix`' option), guess a value for it by looking for *program* in `PATH`, the way the shell does. If *program* is found, set the prefix to the parent of the directory containing *program*; otherwise leave the prefix specified in '`Makefile.in`' unchanged. For example, if *program* is `gcc` and the `PATH` contains '`/usr/local/gnu/bin/gcc`', set the prefix to '`/usr/local/gnu`'.

## 3.7 Version Numbers in `configure`

The following macros manage version numbers for `configure` scripts. Using them is optional.

**AC_PREREQ** (*version*) Macro

Ensure that a recent enough version of Autoconf is being used. If the version of Autoconf being used to create `configure` is earlier than *version*, print an error message on the standard error output and do not create `configure`. For example:

```
AC_PREREQ(1.8)
```

This macro is useful if your '`configure.in`' relies on non-obvious behavior that changed between Autoconf releases. If it merely needs recently added macros, then `AC_PREREQ` is less useful, because the `autoconf` program already tells the user which macros are not found. The same thing happens if '`configure.in`' is processed by a version of Autoconf older than when `AC_PREREQ` was added.

**AC_REVISION** (*revision-info*) Macro

Copy revision stamp *revision-info* into the `configure` script, with any dollar signs or double-quotes removed. This macro lets you put a revision stamp from '`configure.in`' into `configure` without RCS or CVS changing it when you check in `configure`. That way, you can determine easily which revision of '`configure.in`' a particular `configure` corresponds to.

It is a good idea to call this macro before `AC_INIT` so that the revision number is near the top of both '`configure.in`' and `configure`. To support doing that, the `AC_REVISION` output begins with '`#! /bin/sh`', like the normal start of a `configure` script does.

For example, this line in '`configure.in`':

```
AC_REVISION($Revision: 1.30 $)dnl
```

produces this in `configure`:

```
#! /bin/sh
# From configure.in Revision: 1.30
```

# 4 Existing Tests

These macros test for particular system features that packages might need or want to use. If you need to test for a kind of feature that none of these macros check for, you can probably do it by calling primitive test macros with appropriate arguments (see Chapter 5 [Writing Tests], page 45).

These tests print messages telling the user which feature they're checking for, and what they find. They cache their results for future `configure` runs (see Section 6.3 [Caching Results], page 57).

Some of these macros set output variables. See Section 3.3 [Makefile Substitutions], page 12, for how to get their values. The phrase "define *name*" is used below as a shorthand to mean "define C preprocessor symbol *name* to the value 1". See Section 6.1 [Defining Symbols], page 55, for how to get those symbol definitions into your program.

## 4.1 Alternative Programs

These macros check for the presence or behavior of particular programs. They are used to choose between several alternative programs and to decide what to do once one has been chosen. If there is no macro specifically defined to check for a program you need, and you don't need to check for any special properties of it, then you can use one of the general program check macros.

### 4.1.1 Particular Program Checks

These macros check for particular programs—whether they exist, and in some cases whether they support certain features.

**AC_DECL_YYTEXT** Macro

Define `YYTEXT_POINTER` if `yytext` is a '`char *`' instead of a '`char []`'. Also set output variable `LEX_OUTPUT_ROOT` to the base of the file name that the lexer generates; usually '`lex.yy`', but sometimes something else. These results vary according to whether `lex` or `flex` is being used.

**AC_PROG_AWK** Macro

Check for `mawk`, `gawk`, `nawk`, and `awk`, in that order, and set output variable `AWK` to the first one that it finds. It tries `mawk` first because that is reported to be the fastest implementation.

**AC_PROG_CC** Macro

Determine a C compiler to use. If `CC` is not already set in the environment, check for `gcc`, and use `cc` if it's not found. Set output variable `CC` to the name of the compiler found.

If using the GNU C compiler, set shell variable `GCC` to '`yes`', empty otherwise. If output variable `CFLAGS` was not already set, set it to '`-g -O`' for the GNU C compiler ('`-O`' on systems where GCC does not accept '`-g`'), or '`-g`' for other compilers.

**AC_PROG_CC_C_O** Macro

If the C compiler does not accept the '`-c`' and '`-o`' options simultaneously, define `NO_MINUS_C_MINUS_O`.

**AC_PROG_CPP** Macro

Set output variable `CPP` to a command that runs the C preprocessor. If '`$CC -E`' doesn't work, it uses '`/lib/cpp`'. It is only portable to run `CPP` on files with a '`.c`' extension.

If the current language is C (see Section 5.8 [Language Choice], page 53), many of the specific test macros use the value of `CPP` indirectly by calling `AC_TRY_CPP`, `AC_CHECK_HEADER`, `AC_EGREP_HEADER`, or `AC_EGREP_CPP`.

**AC_PROG_CXX** Macro

Determine a C++ compiler to use. Check if the environment variable `CXX` or `CCC` (in that order) is set; if so, set output variable `CXX` to its value. Otherwise search for a C++ compiler under likely names (`c++`, `g++`, `gcc`, `CC`, and `cxx`). If none of those checks succeed, as a last resort set `CXX` to `gcc`.

If using the GNU C++ compiler, set shell variable `GXX` to '`yes`', empty otherwise. If output variable `CXXFLAGS` was not already set, set it to '`-g -O`' for the GNU C++ compiler ('`-O`' on systems where G++ does not accept '`-g`'), or '`-g`' for other compilers.

**AC_PROG_CXXCPP** Macro

Set output variable `CXXCPP` to a command that runs the C++ preprocessor. If '`$CXX -E`' doesn't work, it uses '`/lib/cpp`'. It is only portable to run `CXXCPP` on files with a '`.c`', '`.C`', or '`.cc`' extension.

If the current language is C++ (see Section 5.8 [Language Choice], page 53), many of the specific test macros use the value of `CXXCPP` indirectly by calling `AC_TRY_CPP`, `AC_CHECK_HEADER`, `AC_EGREP_HEADER`, or `AC_EGREP_CPP`.

### AC_PROG_GCC_TRADITIONAL — Macro

Add '`-traditional`' to output variable `CC` if using the GNU C compiler and `ioctl` does not work properly without '`-traditional`'. That usually happens when the fixed header files have not been installed on an old system. Since recent versions of the GNU C compiler fix the header files automatically when installed, this is becoming a less prevalent problem.

### AC_PROG_INSTALL — Macro

Set output variable `INSTALL` to the path of a BSD compatible `install` program, if one is found in the current `PATH`. Otherwise, set `INSTALL` to '*dir*`/install-sh -c`', checking the directories specified to `AC_CONFIG_AUX_DIR` (or its default directories) to determine *dir* (see Section 3.2 [Output], page 11). Also set the variable `INSTALL_PROGRAM` to '`${INSTALL}`' and `INSTALL_DATA` to '`${INSTALL} -m 644`'.

This macro screens out various instances of `install` known to not work. It prefers to find a C program rather than a shell script, for speed. Instead of '`install-sh`', it can also use '`install.sh`', but that name is obsolete because some `make` programs have a rule that creates '`install`' from it if there is no '`Makefile`'.

A copy of '`install-sh`' which you may use comes with Autoconf. If you use `AC_PROG_INSTALL`, you must include either '`install-sh`' or '`install.sh`' in your distribution, or `configure` will produce an error message saying it can't find them—even if the system you're on has a good `install` program. This check is a safety measure to prevent you from accidentally leaving that file out, which would prevent your package from installing on systems that don't have a BSD-compatible `install` program.

If you need to use your own installation program because it has features not found in standard `install` programs, there is no reason to use `AC_PROG_INSTALL`; just put the pathname of your program into your '`Makefile.in`' files.

### AC_PROG_LEX — Macro

If `flex` is found, set output variable `LEX` to '`flex`' and `LEXLIB` to '`-lfl`', if that library is in a standard place. Otherwise set `LEX` to '`lex`' and `LEXLIB` to '`-ll`'.

**AC_PROG_LN_S** Macro

If '`ln -s`' works on the current filesystem (the operating system and filesystem support symbolic links), set output variable `LN_S` to '`ln -s`', otherwise set it to '`ln`'.

**AC_PROG_RANLIB** Macro

Set output variable `RANLIB` to '`ranlib`' if `ranlib` is found, otherwise to '`:`' (do nothing).

**AC_PROG_YACC** Macro

If `bison` is found, set output variable `YACC` to '`bison -y`'. Otherwise, if `byacc` is found, set `YACC` to '`byacc`'. Otherwise set `YACC` to '`yacc`'.

### 4.1.2 Generic Program Checks

These macros are used to find programs not covered by the particular test macros. If you need to check the behavior of a program as well as find out whether it is present, you have to write your own test for it (see Chapter 5 [Writing Tests], page 45). By default, these macros use the environment variable `PATH`. If you need to check for a program that might not be in the user's `PATH`, you can pass a modified path to use instead, like this:

```
AC_PATH_PROG(INETD, inetd, /usr/libexec/inetd,
  $PATH:/usr/libexec:/usr/sbin:/usr/etc:etc)
```

**AC_CHECK_PROG** (*variable, prog-to-check-for, value-if-found* [, *value-if-not-found* [, *path*, [ *reject* ]]]) Macro

Check whether program *prog-to-check-for* exists in `PATH`. If it is found, set *variable* to *value-if-found*, otherwise to *value-if-not-found*, if given. Always pass over *reject* (an absolute file name) even if it is the first found in the search path; in that case, set *variable* using the absolute file name of the *prog-to-check-for* found that is not *reject*. If *variable* was already set, do nothing. Calls `AC_SUBST` for *variable*.

**AC_CHECK_PROGS** (*variable, progs-to-check-for* [, *value-if-not-found* [, *path*]]) Macro

Check for each program in the whitespace-separated list *progs-to-check-for* exists in `PATH`. If it is found, set *variable* to the name of that program. Otherwise, continue checking the next program in the list. If none of the programs in the list are found, set *variable* to *value-if-not-found*; if *value-if-not-found* is not specified, the value of *variable* is not changed. Calls `AC_SUBST` for *variable*.

**AC_CHECK_TOOL** (*variable, prog-to-check-for* [, *value-if-not-found* [, *path*]]) Macro

Like `AC_CHECK_PROG`, but first looks for *prog-to-check-for* with a prefix of the host type as determined by `AC_CANONICAL_HOST`, followed by a dash (see Section 8.2 [Canonicalizing], page 68). For example, if the user runs '`configure --host=i386-gnu`', then this call:

```
AC_CHECK_TOOL(RANLIB, ranlib, :)
```

sets `RANLIB` to '`i386-gnu-ranlib`' if that program exists in `PATH`, or to '`ranlib`' if that program exists in `PATH`, or to '`:`' if neither program exists.

**AC_PATH_PROG** (*variable, prog-to-check-for* [, *value-if-not-found* [, *path*]]) Macro

Like `AC_CHECK_PROG`, but set *variable* to the entire path of *prog-to-check-for* if found.

**AC_PATH_PROGS** (*variable, progs-to-check-for* [, *value-if-not-found* [, *path*]]) Macro

Like `AC_CHECK_PROGS`, but if any of *progs-to-check-for* are found, set *variable* to the entire path of the program found.

## 4.2 Library Files

The following macros check for the presence of certain C library archive files.

**AC_CHECK_LIB** (*library, function* [, *action-if-found* [, *action-if-not-found* [, *other-libraries*]]]) Macro

Try to ensure that C function *function* is available by checking whether a test C program can be linked with the library *library* to get the function. *library* is the base name of the library; e.g., to check for '`-lmp`', use '`mp`' as the *library* argument.

*action-if-found* is a list of shell commands to run if the link with the library succeeds; *action-if-not-found* is a list of shell commands to run if the link fails. If *action-if-found* and *action-if-not-found* are not specified, the default action is to add '`-l`*library*' to `LIBS` and define '`HAVE_LIB`*library*' (in all capitals).

If linking with *library* results in unresolved symbols, which would be resolved by linking with additional libraries, give those libraries as the *other-libraries* argument, separated by spaces: '`-lXt -lX11`'. Otherwise this macro will fail to detect that *library* is present, because linking the test program will always fail with unresolved symbols.

**AC_HAVE_LIBRARY** (*library*, [, *action-if-found* [, *action-if-not-found* [, *other-libraries*]]]) Macro

This macro is equivalent to calling `AC_CHECK_LIB` with a *function* argument of `main`. In addition, *library* can be written as any of '`foo`', '`-lfoo`', or '`libfoo.a`'. In all of those cases, the compiler is passed '`-lfoo`'. However, *library* can not be a shell variable; it must be a literal name. This macro is considered obsolete.

## 4.3 Library Functions

The following macros check for particular C library functions. If there is no macro specifically defined to check for a function you need, and you don't need to check for any special properties of it, then you can use one of the general function check macros.

### 4.3.1 Particular Function Checks

These macros check for particular C functions—whether they exist, and in some cases how they respond when given certain arguments.

**AC_FUNC_ALLOCA** Macro

Check how to get `alloca`. Tries to get a builtin version by checking for '`alloca.h`' or the predefined C preprocessor macros `__GNUC__` and `_AIX`. If this macro finds '`alloca.h`', it defines `HAVE_ALLOCA_H`.

If those attempts fail, it looks for the function in the standard C library. If any of those methods succeed, it defines `HAVE_ALLOCA`. Otherwise, it sets the output variable `ALLOCA` to '`alloca.o`' and defines `C_ALLOCA` (so programs can periodically call '`alloca(0)`' to garbage collect). This variable is separate from `LIBOBJS` so multiple programs can share the value of `ALLOCA` without needing to create an actual library, in case only some of them use the code in `LIBOBJS`.

This macro does not try to get `alloca` from the System V R3 '`libPW`' or the System V R4 '`libucb`' because those libraries contain some incompatible functions that cause trouble. Some versions do not even contain `alloca` or contain a buggy version. If you still want to use their `alloca`, use `ar` to extract '`alloca.o`' from them instead of compiling '`alloca.c`'.

Source files that use `alloca` should start with a piece of code like the following, to declare it properly. In some versions of AIX, the declaration of `alloca` must precede everything else except for comments and preprocessor directives. The `#pragma` directive is indented so that pre-ANSI C compilers will ignore it, rather than choke on it.

```
/* AIX requires this to be the first thing in the file.  */
#ifdef __GNUC__
# define alloca __builtin_alloca
#else
# if HAVE_ALLOCA_H
#  include <alloca.h>
# else
#  ifdef _AIX
 #pragma alloca
#  else
#   ifndef alloca /* predefined by HP cc +Olibcalls */
char *alloca ();
#   endif
#  endif
# endif
#endif
```

**AC_FUNC_CLOSEDIR_VOID** Macro

If the `closedir` function does not return a meaningful value, define `CLOSEDIR_VOID`. Otherwise, callers ought to check its return value for an error indicator.

**AC_FUNC_GETLOADAVG** Macro

Check how to get the system load averages. If the system has the `getloadavg` function, this macro defines `HAVE_GETLOADAVG`, and adds to `LIBS` any libraries needed to get that function.

Otherwise, it adds '`getloadavg.o`' to the output variable `LIBOBJS`, and possibly defines several other C preprocessor macros and output variables:

1. It defines `SVR4`, `DGUX`, `UMAX`, or `UMAX4_3` if on those systems.

2. If it finds 'nlist.h', it defines NLIST_STRUCT.
3. If 'struct nlist' has an 'n_un' member, it defines NLIST_NAME_UNION.
4. If compiling 'getloadavg.c' defines LDAV_PRIVILEGED, programs need to be installed specially on this system for getloadavg to work, and this macro defines GETLOADAVG_PRIVILEGED.
5. This macro sets the output variable NEED_SETGID. The value is 'true' if special installation is required, 'false' if not. If NEED_SETGID is 'true', this macro sets KMEM_GROUP to the name of the group that should own the installed program.

**AC_FUNC_GETMNTENT** Macro

Check for getmntent in the 'sun', 'seq', and 'gen' libraries, for Irix 4, PTX, and Unixware, respectively. Then, if getmntent is available, define HAVE_GETMNTENT.

**AC_FUNC_GETPGRP** Macro

If getpgrp takes no argument (the POSIX.1 version), define GETPGRP_VOID. Otherwise, it is the BSD version, which takes a process ID as an argument. This macro does not check whether getpgrp exists at all; if you need to work in that situation, first call AC_CHECK_FUNC for getpgrp.

**AC_FUNC_MEMCMP** Macro

If the memcmp function is not available, or does not work on 8-bit data (like the one on SunOS 4.1.3), add 'memcmp.o' to output variable LIBOBJS.

**AC_FUNC_MMAP** Macro

If the mmap function exists and works correctly on memory mapped files, define HAVE_MMAP.

**AC_FUNC_SETVBUF_REVERSED** Macro

If setvbuf takes the buffering type as its second argument and the buffer pointer as the third, instead of the other way around, define SETVBUF_REVERSED. This is the case on System V before release 3.

**AC_FUNC_STRCOLL** Macro

If the strcoll function exists and works correctly, define HAVE_STRCOLL. This does a bit more than 'AC_CHECK_FUNCS(strcoll)', because some systems have incorrect definitions of strcoll, which should not be used.

**AC_FUNC_STRFTIME** Macro

Check for `strftime` in the '`intl`' library, for SCO UNIX. Then, if `strftime` is available, define `HAVE_STRFTIME`.

**AC_FUNC_UTIME_NULL** Macro

If '`utime(`*file*`, NULL)`' sets *file*'s timestamp to the present, define `HAVE_UTIME_NULL`.

**AC_FUNC_VFORK** Macro

If '`vfork.h`' is found, define `HAVE_VFORK_H`. If a working `vfork` is not found, define `vfork` to be `fork`. This macro checks for several known errors in implementations of `vfork` and considers the system to not have a working `vfork` if it detects any of them. It is not considered to be an implementation error if a child's invocation of `signal` modifies the parent's signal handler, since child processes rarely change their signal handlers.

**AC_FUNC_VPRINTF** Macro

If `vprintf` is found, define `HAVE_VPRINTF`. Otherwise, if `_doprnt` is found, define `HAVE_DOPRNT`. (If `vprintf` is available, you may assume that `vfprintf` and `vsprintf` are also available.)

**AC_FUNC_WAIT3** Macro

If `wait3` is found and fills in the contents of its third argument (a '`struct rusage *`'), which HP-UX does not do, define `HAVE_WAIT3`.

### 4.3.2 Generic Function Checks

These macros are used to find functions not covered by the particular test macros. If the functions might be in libraries other than the default C library, first call `AC_CHECK_LIB` for those libraries. If you need to check the behavior of a function as well as find out whether it is present, you have to write your own test for it (see Chapter 5 [Writing Tests], page 45).

**AC_CHECK_FUNC** (*function*, [*action-if-found* [, *action-if-not-found*]]) Macro

If C function *function* is available, run shell commands *action-if-found*, otherwise *action-if-not-found*. If you just want to define a symbol if the function is available, consider using `AC_CHECK_FUNCS` instead. This macro checks for functions with C linkage even when `AC_LANG_CPLUSPLUS` has been called, since C++ is more standardized than C is. (see Section 5.8 [Language Choice], page 53, for more information about selecting the language for checks.)

**AC_CHECK_FUNCS** (*function*... [, *action-if-found* [, *action-if-not-found*]]) Macro

For each given *function* in the whitespace-separated argument list that is available, define `HAVE_`*function* (in all capitals). If *action-if-found* is given, it is additional shell code to execute when one of the functions is found. You can give it a value of '`break`' to break out of the loop on the first match. If *action-if-not-found* is given, it is executed when one of the functions is not found.

**AC_REPLACE_FUNCS** (*function-name*...) Macro

For each given *function-name* in the whitespace-separated argument list that is not in the C library, add '*function-name*`.o`' to the value of the output variable `LIBOBJS`.

## 4.4 Header Files

The following macros check for the presence of certain C header files. If there is no macro specifically defined to check for a header file you need, and you don't need to check for any special properties of it, then you can use one of the general header file check macros.

### 4.4.1 Particular Header Checks

These macros check for particular system header files—whether they exist, and in some cases whether they declare certain symbols.

**AC_DECL_SYS_SIGLIST** Macro

Define `SYS_SIGLIST_DECLARED` if the variable `sys_siglist` is declared in a system header file, either '`signal.h`' or '`unistd.h`'.

**AC_DIR_HEADER** Macro

Like calling `AC_HEADER_DIRENT` and `AC_FUNC_CLOSEDIR_VOID`, but defines a different set of C preprocessor macros to indicate which header file is found. This macro and the names it defines are considered obsolete. The names it defines are:

'dirent.h'
: DIRENT

'sys/ndir.h'
: SYSNDIR

'sys/dir.h'
: SYSDIR

'ndir.h'
: NDIR

In addition, if the `closedir` function does not return a meaningful value, define `VOID_CLOSEDIR`.

### AC_HEADER_DIRENT — Macro

Check for the following header files, and for the first one that is found and defines '`DIR`', define the listed C preprocessor macro:

'dirent.h'
: HAVE_DIRENT_H

'sys/ndir.h'
: HAVE_SYS_NDIR_H

'sys/dir.h'
: HAVE_SYS_DIR_H

'ndir.h'
: HAVE_NDIR_H

The directory library declarations in the source code should look something like the following:

```
#if HAVE_DIRENT_H
# include <dirent.h>
# define NAMLEN(dirent) strlen((dirent)->d_name)
#else
# define dirent direct
# define NAMLEN(dirent) (dirent)->d_namlen
# if HAVE_SYS_NDIR_H
#  include <sys/ndir.h>
# endif
# if HAVE_SYS_DIR_H
#  include <sys/dir.h>
# endif
# if HAVE_NDIR_H
#  include <ndir.h>
# endif
#endif
```

Using the above declarations, the program would declare variables to be type `struct dirent`, not `struct direct`, and would access the length of a directory entry name by passing a pointer to a `struct dirent` to the `NAMLEN` macro.

This macro also checks for the SCO Xenix '`dir`' and '`x`' libraries.

### AC_HEADER_MAJOR — Macro

If '`sys/types.h`' does not define `major`, `minor`, and `makedev`, but '`sys/mkdev.h`' does, define `MAJOR_IN_MKDEV`; otherwise, if '`sys/sysmacros.h`' does, define `MAJOR_IN_SYSMACROS`.

### AC_HEADER_STDC — Macro

Define `STDC_HEADERS` if the system has ANSI C header files. Specifically, this macro checks for '`stdlib.h`', '`stdarg.h`', '`string.h`', and '`float.h`'; if the system has those, it probably has the rest of the ANSI C header files. This macro also checks whether '`string.h`' declares `memchr` (and thus presumably the other `mem` functions), whether '`stdlib.h`' declare `free` (and thus presumably `malloc` and other related functions), and whether the '`ctype.h`' macros work on characters with the high bit set, as ANSI C requires.

Use `STDC_HEADERS` instead of `__STDC__` to determine whether the system has ANSI-compliant header files (and probably C library functions) because many systems that have GCC do not have ANSI C header files.

On systems without ANSI C headers, there is so much variation that it is probably easier to declare the functions you use than to figure out exactly what the system header files declare. Some systems contain a mix of functions ANSI and BSD; some are mostly ANSI but lack 'memmove'; some define the BSD functions as macros in 'string.h' or 'strings.h'; some have only the BSD functions but 'string.h'; some declare the memory functions in 'memory.h', some in 'string.h'; etc. It is probably sufficient to check for one string function and one memory function; if the library has the ANSI versions of those then it probably has most of the others. If you put the following in 'configure.in':

```
AC_HEADER_STDC
AC_CHECK_FUNCS(strchr memcpy)
```

then, in your code, you can put declarations like this:

```
#if STDC_HEADERS
# include <string.h>
#else
# ifndef HAVE_STRCHR
#  define strchr index
#  define strrchr rindex
# endif
char *strchr (), *strrchr ();
# ifndef HAVE_MEMCPY
#  define memcpy(d, s, n) bcopy ((s), (d), (n))
#  define memmove(d, s, n) bcopy ((s), (d), (n))
# endif
#endif
```

If you use a function like `memchr`, `memset`, `strtok`, or `strspn`, which have no BSD equivalent, then macros won't suffice; you must provide an implementation of each function. An easy way to incorporate your implementations only when needed (since the ones in system C libraries may be hand optimized) is to, taking `memchr` for example, put it in 'memchr.c' and use 'AC_REPLACE_FUNCS(memchr)'.

### AC_HEADER_SYS_WAIT — Macro

If 'sys/wait.h' exists and is compatible with POSIX.1, define `HAVE_SYS_WAIT_H`. Incompatibility can occur if 'sys/wait.h' does not exist, or if it uses the old BSD `union wait` instead of `int` to store a status value. If 'sys/wait.h' is not POSIX.1 compatible, then instead of including it, define the POSIX.1 macros with their usual interpretations. Here is an example:

```
#include <sys/types.h>
#if HAVE_SYS_WAIT_H
# include <sys/wait.h>
#endif
#ifndef WEXITSTATUS
# define WEXITSTATUS(stat_val) ((unsigned)(stat_val) >> 8)
#endif
#ifndef WIFEXITED
# define WIFEXITED(stat_val) (((stat_val) & 255) == 0)
#endif
```

### AC_MEMORY_H — Macro

Define `NEED_MEMORY_H` if `memcpy`, `memcmp`, etc. are not declared in '`string.h`' and '`memory.h`' exists. This macro is obsolete; instead, use `AC_CHECK_HEADERS(memory.h)`. See the example for `AC_HEADER_STDC`.

### AC_UNISTD_H — Macro

Define `HAVE_UNISTD_H` if the system has '`unistd.h`'. This macro is obsolete; instead, use '`AC_CHECK_HEADERS(unistd.h)`'.

The way to check if the system supports POSIX.1 is:

```
#if HAVE_UNISTD_H
# include <sys/types.h>
# include <unistd.h>
#endif

#ifdef _POSIX_VERSION
/* Code for POSIX.1 systems.  */
#endif
```

`_POSIX_VERSION` is defined when '`unistd.h`' is included on POSIX.1 systems. If there is no '`unistd.h`', it is definitely not a POSIX.1 system. However, some non-POSIX.1 systems do have '`unistd.h`'.

### AC_USG — Macro

Define `USG` if the system does not have '`strings.h`', `rindex`, `bzero`, etc. This implies that it has '`string.h`', `strrchr`, `memset`, etc.

The symbol `USG` is obsolete. Instead of this macro, see the example for `AC_HEADER_STDC`.

### 4.4.2 Generic Header Checks

These macros are used to find system header files not covered by the particular test macros. If you need to check the contents of a header as well as find out whether it is present, you have to write your own test for it (see Chapter 5 [Writing Tests], page 45).

**AC_CHECK_HEADER** (*header-file*, [*action-if-found* [, *action-if-not-found*]]) Macro

If the system header file *header-file* exists, execute shell commands *action-if-found*, otherwise execute *action-if-not-found*. If you just want to define a symbol if the header file is available, consider using `AC_CHECK_HEADERS` instead.

**AC_CHECK_HEADERS** (*header-file*... [, *action-if-found* [, *action-if-not-found*]]) Macro

For each given system header file *header-file* in the whitespace-separated argument list that exists, define `HAVE_`*header-file* (in all capitals). If *action-if-found* is given, it is additional shell code to execute when one of the header files is found. You can give it a value of '`break`' to break out of the loop on the first match. If *action-if-not-found* is given, it is executed when one of the header files is not found.

## 4.5 Structures

The following macros check for certain structures or structure members. To check structures not listed here, use `AC_EGREP_CPP` (see Section 5.1 [Examining Declarations], page 45) or `AC_TRY_COMPILE` (see Section 5.2 [Examining Syntax], page 46).

**AC_HEADER_STAT** Macro

If the macros `S_ISDIR`, `S_ISREG` et al. defined in '`sys/stat.h`' do not work properly (returning false positives), define `STAT_MACROS_BROKEN`. This is the case on Tektronix UTekV, Amdahl UTS and Motorola System V/88.

**AC_HEADER_TIME** Macro

If a program may include both '`time.h`' and '`sys/time.h`', define `TIME_WITH_SYS_TIME`. On some older systems, '`sys/time.h`' includes '`time.h`', but '`time.h`' is not protected against multiple inclusion, so programs should not explicitly include both files. This macro is useful in programs that use, for example, `struct timeval` or struct timezone as well as struct tm. It is best used in conjunction with HAVE_SYS_TIME_H, which can be checked for using AC_CHECK_HEADERS(sys/time.h).

```
#if TIME_WITH_SYS_TIME
# include <sys/time.h>
# include <time.h>
#else
# if HAVE_SYS_TIME_H
#  include <sys/time.h>
# else
#  include <time.h>
# endif
#endif
```

**AC_STRUCT_ST_BLKSIZE** Macro

If struct stat contains an st_blksize member, define HAVE_ST_BLKSIZE.

**AC_STRUCT_ST_BLOCKS** Macro

If struct stat contains an st_blocks member, define HAVE_ST_BLOCKS. Otherwise, add 'fileblocks.o' to the output variable LIBOBJS.

**AC_STRUCT_ST_RDEV** Macro

If struct stat contains an st_rdev member, define HAVE_ST_RDEV.

**AC_STRUCT_TM** Macro

If 'time.h' does not define struct tm, define TM_IN_SYS_TIME, which means that including 'sys/time.h' had better define struct tm.

**AC_STRUCT_TIMEZONE** Macro

Figure out how to get the current timezone. If struct tm has a tm_zone member, define HAVE_TM_ZONE. Otherwise, if the external array tzname is found, define HAVE_TZNAME.

## 4.6 Typedefs

The following macros check for C typedefs. If there is no macro specifically defined to check for a typedef you need, and you don't need to check for any special properties of it, then you can use a general typedef check macro.

### 4.6.1 Particular Typedef Checks

These macros check for particular C typedefs in 'sys/types.h' and 'stdlib.h' (if it exists).

**AC_TYPE_GETGROUPS** Macro

Define `GETGROUPS_T` to be whichever of `gid_t` or `int` is the base type of the array argument to `getgroups`.

**AC_TYPE_MODE_T** Macro

If `mode_t` is not defined, define `mode_t` to be `int`.

**AC_TYPE_OFF_T** Macro

If `off_t` is not defined, define `off_t` to be `long`.

**AC_TYPE_PID_T** Macro

If `pid_t` is not defined, define `pid_t` to be `int`.

**AC_TYPE_SIGNAL** Macro

If 'signal.h' declares `signal` as returning a pointer to a function returning `void`, define `RETSIGTYPE` to be `void`; otherwise, define it to be `int`.

Define signal handlers as returning type `RETSIGTYPE`:

```
RETSIGTYPE
hup_handler ()
{
...
}
```

**AC_TYPE_SIZE_T** Macro

If `size_t` is not defined, define `size_t` to be `unsigned`.

**AC_TYPE_UID_T** Macro

If `uid_t` is not defined, define `uid_t` to be `int` and `gid_t` to be `int`.

### 4.6.2 Generic Typedef Checks

This macro is used to check for typedefs not covered by the particular test macros.

**AC_CHECK_TYPE** (*type, default*) Macro

If the type *type* is not defined in '`sys/types.h`' or '`stdlib.h`' (if it exists), define it to be the C (or C++) builtin type *default*; e.g., '`short`' or '`unsigned`'.

## 4.7 Compiler Characteristics

The following macros check for C compiler or machine architecture features. To check for characteristics not listed here, use `AC_TRY_COMPILE` (see Section 5.2 [Examining Syntax], page 46) or `AC_TRY_RUN` (see Section 5.4 [Run Time], page 48)

**AC_C_BIGENDIAN** Macro

If words are stored with the most significant byte first (like Motorola and SPARC, but not Intel and VAX, CPUs), define `WORDS_BIGENDIAN`.

**AC_C_CONST** Macro

If the C compiler does not fully support the keyword `const`, define `const` to be empty. Some C compilers that do not define `__STDC__` do support `const`; some compilers that define `__STDC__` do not completely support `const`. Programs can simply use `const` as if every C compiler supported it; for those that don't, the '`Makefile`' or configuration header file will define it as empty.

**AC_C_INLINE** Macro

If the C compiler supports the keyword `inline`, do nothing. Otherwise define `inline` to `__inline__` or `__inline` if it accepts one of those, otherwise define `inline` to be empty.

**AC_C_CHAR_UNSIGNED** Macro

If the C type `char` is unsigned, define `__CHAR_UNSIGNED__`, unless the C compiler predefines it.

**AC_C_LONG_DOUBLE** Macro

If the C compiler supports the `long double` type, define `HAVE_LONG_DOUBLE`. Some C compilers that do not define `__STDC__` do support the `long double` type; some compilers that define `__STDC__` do not support `long double`.

**AC_CHECK_SIZEOF** (*type* [, *cross-size*]) Macro

Define `SIZEOF_`*uctype* to be the size in bytes of the C (or C++) builtin type *type*, e.g. '`int`' or '`char *`'. If '`type`' is unknown to the compiler, it gets a size of 0. *uctype* is *type*, with lowercase converted to uppercase, spaces changed to underscores, and asterisks changed to '`P`'. If cross-compiling, the value *cross-size* is used if given, otherwise `configure` exits with an error message.

For example, the call

```
AC_CHECK_SIZEOF(int *)
```

defines `SIZEOF_INT_P` to be 8 on DEC Alpha AXP systems.

**AC_INT_16_BITS** Macro

If the C type `int` is 16 bits wide, define `INT_16_BITS`. This macro is obsolete; it is more general to use '`AC_CHECK_SIZEOF(int)`' instead.

**AC_LONG_64_BITS** Macro

If the C type `long int` is 64 bits wide, define `LONG_64_BITS`. This macro is obsolete; it is more general to use '`AC_CHECK_SIZEOF(long)`' instead.

## 4.8 System Services

The following macros check for operating system services or capabilities.

**AC_SYS_INTERPRETER** Macro

Check whether the system supports starting scripts with a line of the form '`#! /bin/csh`' to select the interpreter to use for the script. After running this macro, shell code in `configure.in` can check the variable `ac_cv_sys_interpreter`; it will be set to '`yes`' if the system supports '`#!`', '`no`' if not.

**AC_PATH_X** Macro

Try to locate the X Window System include files and libraries. If the user gave the command line options '`--x-includes=`*dir*' and '`--x-libraries=`*dir*', use those directories. If either or both were not given, get the missing values by running `xmkmf` on a trivial '`Imakefile`' and examining the '`Makefile`' that it produces. If that fails (such as if `xmkmf` is not present), look for them in several directories where they often reside. If either method is successful, set the shell variables `x_includes` and `x_libraries` to their locations, unless they are in directories the compiler searches by default.

If both methods fail, or the user gave the command line option '`--without-x`', set the shell variable `no_x` to '`yes`'; otherwise set it to the empty string.

**AC_PATH_XTRA** Macro

An enhanced version of `AC_PATH_X`. It adds the C compiler flags that X needs to output variable `X_CFLAGS`, and the X linker flags to `X_LIBS`. If X is not available, adds '`-DX_DISPLAY_MISSING`' to `X_CFLAGS`.

This macro also checks for special libraries that some systems need in order to compile X programs. It adds any that the system needs to output variable `X_EXTRA_LIBS`. And it checks for special X11R6 libraries that need to be linked with before '`-lX11`', and adds any found to the output variable `X_PRE_LIBS`.

**AC_SYS_LONG_FILE_NAMES** Macro

If the system supports file names longer than 14 characters, define `HAVE_LONG_FILE_NAMES`.

**AC_SYS_RESTARTABLE_SYSCALLS** Macro

If the system automatically restarts a system call that is interrupted by a signal, define `HAVE_RESTARTABLE_SYSCALLS`.

## 4.9 UNIX Variants

The following macros check for certain operating systems that need special treatment for some programs, due to exceptional oddities in their header files or libraries. These macros are warts; they will be replaced by a more systematic approach, based on the functions they make available or the environments they provide.

**AC_AIX** Macro

If on AIX, define `_ALL_SOURCE`. Allows the use of some BSD functions. Should be called before any macros that run the C compiler.

**AC_DYNIX_SEQ** Macro

If on Dynix/PTX (Sequent UNIX), add '`-lseq`' to output variable `LIBS`. This macro is obsolete; instead, use `AC_FUNC_GETMNTENT`.

**AC_IRIX_SUN** Macro

If on IRIX (Silicon Graphics UNIX), add '`-lsun`' to output variable `LIBS`. This macro is obsolete. If you were using it to get `getmntent`, use `AC_FUNC_GETMNTENT` instead. If you used it for the NIS versions of the password and group functions, use '`AC_CHECK_LIB(sun, getpwnam)`'.

**AC_ISC_POSIX** Macro

If on a POSIXized ISC UNIX, define `_POSIX_SOURCE` and add '`-posix`' (for the GNU C compiler) or '`-Xp`' (for other C compilers) to output variable `CC`. This allows the use of POSIX facilities. Must be called after `AC_PROG_CC` and before any other macros that run the C compiler.

**AC_MINIX** Macro

If on Minix, define `_MINIX` and `_POSIX_SOURCE` and define `_POSIX_1_SOURCE` to be 2. This allows the use of POSIX facilities. Should be called before any macros that run the C compiler.

**AC_SCO_INTL** Macro

If on SCO UNIX, add '`-lintl`' to output variable `LIBS`. This macro is obsolete; instead, use `AC_FUNC_STRFTIME`.

**AC_XENIX_DIR** Macro

If on Xenix, add '`-lx`' to output variable `LIBS`. Also, if '`dirent.h`' is being used, add '`-ldir`' to `LIBS`. This macro is obsolete; use `AC_HEADER_DIRENT` instead.

# 5 Writing Tests

If the existing feature tests don't do something you need, you have to write new ones. These macros are the building blocks. They provide ways for other macros to check whether various kinds of features are available and report the results.

This chapter contains some suggestions and some of the reasons why the existing tests are written the way they are. You can also learn a lot about how to write Autoconf tests by looking at the existing ones. If something goes wrong in one or more of the Autoconf tests, this information can help you understand the assumptions behind them, which might help you figure out how to best solve the problem.

These macros check the output of the C compiler system. They do not cache the results of their tests for future use (see Section 6.3 [Caching Results], page 57), because they don't know enough about the information they are checking for to generate a cache variable name. They also do not print any messages, for the same reason. The checks for particular kinds of C features call these macros and do cache their results and print messages about what they're checking for.

When you write a feature test that could be applicable to more than one software package, the best thing to do is encapsulate it in a new macro. See Chapter 7 [Writing Macros], page 62, for how to do that.

## 5.1 Examining Declarations

The macro `AC_TRY_CPP` is used to check whether particular header files exist. You can check for one at a time, or more than one if you need several header files to all exist for some purpose.

**AC_TRY_CPP** (*includes*, [*action-if-true* [, *action-if-false*]]) Macro

*includes* is C or C++ `#include` statements and declarations, on which shell variable, backquote, and backslash substitutions are performed. (Actually, it can be any C program, but other statements are probably not useful.) If the preprocessor produces no error messages while processing it, run shell commands *action-if-true*. Otherwise run shell commands *action-if-false*.

This macro uses `CPPFLAGS`, but not `CFLAGS`, because '`-g`', '`-O`', etc. are not valid options to many C preprocessors.

Here is how to find out whether a header file contains a particular declaration, such as a typedef, a structure, a structure member, or a function. Use `AC_EGREP_HEADER` instead of running `grep` directly on the header file; on some systems the symbol might be defined in another header file that the file you are checking '`#include`'s.

**AC_EGREP_HEADER** (*pattern, header-file, action-if-found* [, *action-if-not-found*]) Macro

If the output of running the preprocessor on the system header file *header-file* matches the `egrep` regular expression *pattern*, execute shell commands *action-if-found*, otherwise execute *action-if-not-found*.

To check for C preprocessor symbols, either defined by header files or predefined by the C preprocessor, use `AC_EGREP_CPP`. Here is an example of the latter:

```
AC_EGREP_CPP(yes,
[#ifdef _AIX
  yes
#endif
], is_aix=yes, is_aix=no)
```

**AC_EGREP_CPP** (*pattern, program,* [*action-if-found* [, *action-if-not-found*]]) Macro

*program* is the text of a C or C++ program, on which shell variable, backquote, and backslash substitutions are performed. If the output of running the preprocessor on *program* matches the `egrep` regular expression *pattern*, execute shell commands *action-if-found*, otherwise execute *action-if-not-found*.

This macro calls `AC_PROG_CPP` or `AC_PROG_CXXCPP` (depending on which language is current, see Section 5.8 [Language Choice], page 53), if it hasn't been called already.

## 5.2 Examining Syntax

To check for a syntax feature of the C or C++ compiler, such as whether it recognizes a certain keyword, use `AC_TRY_COMPILE` to try to compile a small program that uses that feature. You can also use it to check for structures and structure members that are not present on all systems.

**AC_TRY_COMPILE** (*includes, function-body,* [*action-if-found* [, *action-if-not-found*]]) Macro

Create a test C program to see whether a function whose body consists of *function-body* can be compiled; *includes* is any `#include` statements needed by the code in *function-body*. If the file compiles successfully, run shell commands *action-if-found*, otherwise run *action-if-not-found*. This macro uses `CFLAGS` or `CXXFLAGS`, and `CPPFLAGS`, when compiling. It does not try to link; use `AC_TRY_LINK` if you need to do that (see Section 5.3 [Examining Libraries], page 47).

## 5.3 Examining Libraries

To check for a library, a function, or a global variable, Autoconf `configure` scripts try to compile and link a small program that uses it. This is unlike Metaconfig, which by default uses `nm` or `ar` on the C library to try to figure out which functions are available. Trying to link with the function is usually a more reliable approach because it avoids dealing with the variations in the options and output formats of `nm` and `ar` and in the location of the standard libraries. It also allows configuring for cross-compilation or checking a function's runtime behavior if needed. On the other hand, it can be slower than scanning the libraries once.

A few systems have linkers that do not return a failure exit status when there are unresolved functions in the link. This bug makes the configuration scripts produced by Autoconf unusable on those systems. However, some of them can be given options that make the exit status correct. This is a problem that Autoconf does not currently handle automatically. If users encounter this problem, they might be able to solve it by setting `LDFLAGS` in the environment to pass whatever options the linker needs (for example, '`-Wl,-dn`' on MIPS RISC/OS).

`AC_TRY_LINK` is used to compile test programs to test for functions and global variables. It is also used (by `AC_CHECK_LIB`) to check for libraries, by adding the library being checked for to `LIBS` temporarily and trying to link a small program.

**AC_TRY_LINK** (*includes, function-body,* [*action-if-found* [, *action-if-not-found*]]) Macro

Create a test C program to see whether a function whose body consists of *function-body* can be compiled and linked; *includes* is any `#include` statements needed by the code in *function-body*. If the file compiles and links successfully, run shell commands *action-if-found*, otherwise run *action-if-not-found*. This macro uses `CFLAGS` or `CXXFLAGS`, `CPPFLAGS`, `LDFLAGS`, and `LIBS` when compiling.

**AC_COMPILE_CHECK** (*echo-text, includes, function-body, action-if-found* [, *action-if-not-found*]) Macro

This is an obsolete version of `AC_TRY_LINK`, with the addition that it prints '`checking for` *echo-text*' to the standard output first, if *echo-text* is non-empty. Use `AC_MSG_CHECKING` and `AC_MSG_RESULT` instead to print messages (see Section 6.4 [Printing Messages], page 60).

## 5.4 Checking Run Time Behavior

Sometimes you need to find out how a system performs at run time, such as whether a given function has a certain capability or bug. If you can, make such checks when your program runs instead of when it is configured. You can check for things like the machine's endianness when your program initializes itself.

If you really need to test for a run-time behavior while configuring, you can write a test program to determine the result, and compile and run it using `AC_TRY_RUN`. Avoid running test programs if possible, because using them prevents people from configuring your package for cross-compiling.

### 5.4.1 Running Test Programs

Use the following macro if you need to test run-time behavior of the system while configuring.

**AC_TRY_RUN** (*program,* [*action-if-true* [, *action-if-false* [, *action-if-cross-compiling*]]]) Macro

*program* is the text of a C program, on which shell variable and backquote substitutions are performed. If it compiles and links successfully and returns an exit status of 0 when executed, run shell commands *action-if-true*. Otherwise run shell commands *action-if-false*; the exit status of the program is available in the shell variable '`$?`'. This macro uses `CFLAGS` or `CXXFLAGS`, `CPPFLAGS`, `LDFLAGS`, and `LIBS` when compiling.

If the C compiler being used does not produce executables that run on the system where `configure` is being run, then the test program is not run. If the optional shell commands *action-if-cross-compiling* are given, they are run instead and this macro calls `AC_C_CROSS` if it has not already been called. Otherwise, `configure` prints an error message and exits.

Try to provide a pessimistic default value to use when cross-compiling makes run-time tests impossible. You do this by passing the optional last argument to `AC_TRY_RUN`. `autoconf` prints a warning message when creating `configure` each time it encounters a call to `AC_TRY_RUN` with no *action-if-cross-compiling* argument given. You may ignore the warning, though users will not be able to configure your package for cross-compiling. A few of the macros distributed with Autoconf produce this warning message.

To configure for cross-compiling you can also choose a value for those parameters based on the canonical system name (see Chapter 8 [Manual Configuration], page 67). Alternatively, set up a test results cache file with the correct values for the target system (see Section 6.3 [Caching Results], page 57).

To provide a default for calls of `AC_TRY_RUN` that are embedded in other macros, including a few of the ones that come with Autoconf, you can call `AC_C_CROSS` before running them. Then, if the shell variable `cross_compiling` is set to '`yes`', use an alternate method to get the results instead of calling the macros.

**AC_C_CROSS** Macro

> If the C compiler being used does not produce executables that can run on the system where `configure` is being run, set the shell variable `cross_compiling` to '`yes`', otherwise '`no`'.

### 5.4.2 Guidelines for Test Programs

Test programs should not write anything to the standard output. They should return 0 if the test succeeds, nonzero otherwise, so that success can be distinguished easily from a core dump or other failure; segmentation violations and other failures produce a nonzero exit status. Test programs should `exit`, not `return`, from `main`, because on some systems (old Suns, at least) the argument to `return` in `main` is ignored.

Test programs can use `#if` or `#ifdef` to check the values of preprocessor macros defined by tests that have already run. For example, if you call `AC_HEADER_STDC`, then later on in '`configure.in`' you can have a test program that includes an ANSI C header file conditionally:

```
#if STDC_HEADERS
# include <stdlib.h>
#endif
```

If a test program needs to use or create a data file, give it a name that starts with 'conftest', such as 'conftestdata'. The configure script cleans up by running 'rm -rf conftest*' after running test programs and if the script is interrupted.

### 5.4.3 Test Functions

Function declarations in test programs should have a prototype conditionalized for C++. In practice, though, test programs rarely need functions that take arguments.

```
#ifdef __cplusplus
foo(int i)
#else
foo(i) int i;
#endif
```

Functions that test programs declare should also be conditionalized for C++, which requires 'extern "C"' prototypes. Make sure to not include any header files containing clashing prototypes.

```
#ifdef __cplusplus
extern "C" void *malloc(size_t);
#else
char *malloc();
#endif
```

If a test program calls a function with invalid parameters (just to see whether it exists), organize the program to ensure that it never invokes that function. You can do this by calling it in another function that is never invoked. You can't do it by putting it after a call to exit, because GCC version 2 knows that exit never returns and optimizes out any code that follows it in the same block.

If you include any header files, make sure to call the functions relevant to them with the correct number of arguments, even if they are just 0, to avoid compilation errors due to prototypes. GCC version 2 has internal prototypes for several functions that it automatically inlines; for example, memcpy. To avoid errors when checking for them, either pass them the correct number of arguments or redeclare them with a different return type (such as char).

## 5.5 Portable Shell Programming

When writing your own checks, there are some shell script programming techniques you should avoid in order to make your code portable. The Bourne shell and upward-compatible shells like Bash and the Korn shell have evolved over the years, but to prevent trouble, do not take advantage of features that were added after UNIX version 7, circa 1977. You should not use shell functions, aliases, negated character classes, or other features that are not found in all Bourne-compatible shells; restrict yourself to the lowest common denominator. Even `unset` is not supported by all shells! Also, include a space after the exclamation point in interpreter specifications, like this:

```
#! /usr/bin/perl
```

If you omit the space before the path, then 4.2BSD based systems (such as Sequent DYNIX) will ignore the line, because they interpret '`#! /`' as a 4-byte magic number.

The set of external programs you should run in a `configure` script is fairly small. See section "Utilities in Makefiles" in *GNU Coding Standards*, for the list. This restriction allows users to start out with a fairly small set of programs and build the rest, avoiding too many interdependencies between packages.

Some of these external utilities have a portable subset of features, as well; for example, don't rely on `ln` having a '`-f`' option or `cat` having any options. `sed` scripts should not contain comments or use branch labels longer than 8 characters. Don't use '`grep -s`' to suppress output, because '`grep -s`' on System V does not suppress output, only error messages. Instead, redirect the standard output and standard error (in case the file doesn't exist) of `grep` to '`/dev/null`'. Check the exit status of `grep` to determine whether it found a match.

## 5.6 Testing Values and Files

`configure` scripts need to test properties of many files and strings. Here are some portability problems to watch out for when doing those tests.

The `test` program is the way to perform many file and string tests. It is often invoked by the alternate name '`[`', but using that name in Autoconf code is asking for trouble since it is an `m4` quote character.

If you need to make multiple checks using `test`, combine them with the shell operators '`&&`' and '`||`' instead of using the `test` operators '`-a`' and '`-o`'. On System V, the precedence of '`-a`' and

'`-o`' is wrong relative to the unary operators; consequently, POSIX does not specify them, so using them is nonportable. If you combine '`&&`' and '`||`' in the same statement, keep in mind that they have equal precedence.

To enable `configure` scripts to support cross-compilation, they shouldn't do anything that tests features of the host system instead of the target system. But occasionally you may find it necessary to check whether some arbitrary file exists. To do so, use '`test -f`' or '`test -r`'. Do not use '`test -x`', because 4.3BSD does not have it.

Another nonportable shell programming construction is

```
var=${var:-value}
```

The intent is to set *var* to *value* only if it is not already set, but if *var* has any value, even the empty string, to leave it alone. Old BSD shells, including the Ultrix `sh`, don't accept the colon, and complain and die. A portable equivalent is

```
: ${var=value}
```

## 5.7 Multiple Cases

Some operations are accomplished in several possible ways, depending on the UNIX variant. Checking for them essentially requires a "case statement". Autoconf does not directly provide one; however, it is easy to simulate by using a shell variable to keep track of whether a way to perform the operation has been found yet.

Here is an example that uses the shell variable `fstype` to keep track of whether the remaining cases need to be checked.

```
AC_MSG_CHECKING(how to get filesystem type)
fstype=no
# The order of these tests is important.
AC_TRY_CPP([#include <sys/statvfs.h>
#include <sys/fstyp.h>], AC_DEFINE(FSTYPE_STATVFS) fstype=SVR4)
if test $fstype = no; then
AC_TRY_CPP([#include <sys/statfs.h>
#include <sys/fstyp.h>], AC_DEFINE(FSTYPE_USG_STATFS) fstype=SVR3)
fi
if test $fstype = no; then
AC_TRY_CPP([#include <sys/statfs.h>
#include <sys/vmount.h>], AC_DEFINE(FSTYPE_AIX_STATFS) fstype=AIX)
fi
# (more cases omitted here)
AC_MSG_RESULT($fstype)
```

## 5.8 Language Choice

Packages that use both C and C++ need to test features of both compilers. Autoconf-generated `configure` scripts check for C features by default. The following macros determine which language's compiler is used in tests that follow in '`configure.in`'.

**AC_LANG_C** Macro

Do compilation tests using `CC` and `CPP` and use extension '`.c`' for test programs.

**AC_LANG_CPLUSPLUS** Macro

Do compilation tests using `CXX` and `CXXCPP` and use extension '`.C`' for test programs.

**AC_LANG_SAVE** Macro

Remember the current language (as set by `AC_LANG_C` or `AC_LANG_CPLUSPLUS`) on a stack. Does not change which language is current. Use this macro and `AC_LANG_RESTORE` in macros that need to temporarily switch to a particular language.

**AC_LANG_RESTORE** Macro

Select the language that is saved on the top of the stack, as set by `AC_LANG_SAVE`, and remove it from the stack. This macro is equivalent to either `AC_LANG_C` or `AC_LANG_CPLUSPLUS`, whichever had been run most recently when `AC_LANG_SAVE` was last called.

Do not call this macro more times than `AC_LANG_SAVE`.

**AC_REQUIRE_CPP** Macro

Ensure that whichever preprocessor would currently be used for tests has been found. Calls `AC_REQUIRE` (see Section 7.4.1 [Prerequisite Macros], page 65) with an argument of either `AC_PROG_CPP` or `AC_PROG_CXXCPP`, depending on which language is current.

# 6 Results of Tests

Once `configure` has determined whether a feature exists, what can it do to record that information? There are four sorts of things it can do: define a C preprocessor symbol, set a variable in the output files, save the result in a cache file for future `configure` runs, and print a message letting the user know the result of the test.

## 6.1 Defining C Preprocessor Symbols

A common action to take in response to a feature test is to define a C preprocessor symbol indicating the results of the test. That is done by calling `AC_DEFINE` or `AC_DEFINE_UNQUOTED`.

By default, `AC_OUTPUT` places the symbols defined by these macros into the output variable `DEFS`, which contains an option '`-D`*symbol*`=`*value*' for each symbol defined. Unlike in Autoconf version 1, there is no variable `DEFS` defined while `configure` is running. To check whether Autoconf macros have already defined a certain C preprocessor symbol, test the value of the appropriate cache variable, as in this example:

```
AC_CHECK_FUNC(vprintf, AC_DEFINE(HAVE_VPRINTF))
if test "$ac_cv_func_vprintf" != yes; then
AC_CHECK_FUNC(_doprnt, AC_DEFINE(HAVE_DOPRNT))
fi
```

If `AC_CONFIG_HEADER` has been called, then instead of creating `DEFS`, `AC_OUTPUT` creates a header file by substituting the correct values into `#define` statements in a template file. See Section 3.4 [Configuration Headers], page 17, for more information about this kind of output.

**AC_DEFINE** (*variable* [, *value*]) Macro

Define C preprocessor variable *variable*. If *value* is given, set *variable* to that value (verbatim), otherwise set it to 1. *value* should not contain literal newlines, and if you are not using `AC_CONFIG_HEADER` it should not contain any '`#`' characters, as `make` tends to eat them. To use a shell variable (which you need to do in order to define a value containing the `m4` quote characters '`[`' or '`]`'), use `AC_DEFINE_UNQUOTED` instead. The following example defines the C preprocessor variable `EQUATION` to be the string constant '`"$a > $b"`':

```
AC_DEFINE(EQUATION, "$a > $b")
```

**AC_DEFINE_UNQUOTED** (*variable* [, *value*]) Macro

Like `AC_DEFINE`, but three shell expansions are performed—once—on *variable* and *value*: variable expansion (‘`$`’), command substitution (‘`‘`’), and backslash escaping (‘`\`’). Single and double quote characters in the value have no special meaning. Use this macro instead of `AC_DEFINE` when *variable* or *value* is a shell variable. Examples:

```
AC_DEFINE_UNQUOTED(config_machfile, "${machfile}")
AC_DEFINE_UNQUOTED(GETGROUPS_T, $ac_cv_type_getgroups)
AC_DEFINE_UNQUOTED(${ac_tr_hdr})
```

Due to the syntactical bizarreness of the Bourne shell, do not use semicolons to separate `AC_DEFINE` or `AC_DEFINE_UNQUOTED` calls from other macro calls or shell code; that can cause syntax errors in the resulting `configure` script. Use either spaces or newlines. That is, do this:

```
AC_CHECK_HEADER(elf.h, AC_DEFINE(SVR4) LIBS="$LIBS -lelf")
```

or this:

```
AC_CHECK_HEADER(elf.h,
  AC_DEFINE(SVR4)
  LIBS="$LIBS -lelf")
```

instead of this:

```
AC_CHECK_HEADER(elf.h, AC_DEFINE(SVR4); LIBS="$LIBS -lelf")
```

## 6.2 Setting Output Variables

One way to record the results of tests is to set *output variables*, which are shell variables whose values are substituted into files that `configure` outputs. The two macros below create new output variables. See Section 3.3.1 [Preset Output Variables], page 12, for a list of output variables that are always available.

**AC_SUBST** (*variable*) Macro

Create an output variable from a shell variable. Make `AC_OUTPUT` substitute the variable *variable* into output files (typically one or more ‘`Makefile`’s). This means that `AC_OUTPUT` will replace instances of ‘`@`*variable*`@`’ in input files with the value that the shell variable *variable* has when `AC_OUTPUT` is called. The value of *variable* should not contain literal newlines.

**AC_SUBST_FILE** (*variable*) Macro

Another way to create an output variable from a shell variable. Make `AC_OUTPUT` insert (without substitutions) the contents of the file named by shell variable *variable* into output files. This means that `AC_OUTPUT` will replace instances of '`@`*variable*`@`' in output files (such as '`Makefile.in`') with the contents of the file that the shell variable *variable* names when `AC_OUTPUT` is called. Set the variable to '`/dev/null`' for cases that do not have a file to insert.

This macro is useful for inserting '`Makefile`' fragments containing special dependencies or other `make` directives for particular host or target types into '`Makefile`'s. For example, '`configure.in`' could contain:

```
AC_SUBST_FILE(host_frag)dnl
host_frag=$srcdir/conf/sun4.mh
```

and then a '`Makefile.in`' could contain:

```
@host_frag@
```

## 6.3 Caching Results

To avoid checking for the same features repeatedly in various `configure` scripts (or repeated runs of one script), `configure` saves the results of many of its checks in a *cache file*. If, when a `configure` script runs, it finds a cache file, it reads from it the results from previous runs and avoids rerunning those checks. As a result, `configure` can run much faster than if it had to perform all of the checks every time.

**AC_CACHE_VAL** (*cache-id, commands-to-set-it*) Macro

Ensure that the results of the check identified by *cache-id* are available. If the results of the check were in the cache file that was read, and `configure` was not given the '`--quiet`' or '`--silent`' option, print a message saying that the result was cached; otherwise, run the shell commands *commands-to-set-it*. Those commands should have no side effects except for setting the variable *cache-id*. In particular, they should not call `AC_DEFINE`; the code that follows the call to `AC_CACHE_VAL` should do that, based on the cached value. Also, they should not print any messages, for example with `AC_MSG_CHECKING`; do that before calling `AC_CACHE_VAL`, so the messages are printed regardless of whether the results of the check are retrieved from the cache or determined by running the shell commands. If the shell commands are run to determine the value, the value will be saved in the cache file just before `configure` creates its output files. See Section 6.3.1 [Cache Variable Names], page 58, for how to choose the name of the *cache-id* variable.

**AC_CACHE_CHECK** (*message, cache-id, commands*) Macro

A wrapper for `AC_CACHE_VAL` that takes care of printing the messages. This macro provides a convenient shorthand for the most common way to use these macros. It calls `AC_MSG_CHECKING` for *message*, then `AC_CACHE_VAL` with the *cache-id* and *commands* arguments, and `AC_MSG_RESULT` with *cache-id*.

### 6.3.1 Cache Variable Names

The names of cache variables should have the following format:

*package-prefix*`_cv_`*value-type*`_`*specific-value*[`_`*additional-options*]

for example, '`ac_cv_header_stat_broken`' or '`ac_cv_prog_gcc_traditional`'. The parts of the variable name are:

*package-prefix*
: An abbreviation for your package or organization; the same prefix you begin local Autoconf macros with, except lowercase by convention. For cache values used by the distributed Autoconf macros, this value is '`ac`'.

`_cv_`
: Indicates that this shell variable is a cache value.

*value-type*
: A convention for classifying cache values, to produce a rational naming system. The values used in Autoconf are listed in Section 7.2 [Macro Names], page 62.

*specific-value*
: Which member of the class of cache values this test applies to. For example, which function ('`alloca`'), program ('`gcc`'), or output variable ('`INSTALL`').

*additional-options*
: Any particular behavior of the specific member that this test applies to. For example, '`broken`' or '`set`'. This part of the name may be omitted if it does not apply.

Like their names, the values that may be assigned to cache variables have a few restrictions. The values may not contain single quotes or curly braces. Usually, their values will be boolean (`yes` or `no`) or the names of files or functions; so this is not an important restriction.

### 6.3.2 Cache Files

A cache file is a shell script that caches the results of configure tests run on one system so they can be shared between configure scripts and configure runs. It is not useful on other systems. If its contents are invalid for some reason, the user may delete or edit it.

By default, configure uses `./config.cache` as the cache file, creating it if it does not exist already. `configure` accepts the `--cache-file=`*file* option to use a different cache file; that is what `configure` does when it calls `configure` scripts in subdirectories, so they share the cache. See Section 3.5 [Subdirectories], page 20, for information on configuring subdirectories with the `AC_CONFIG_SUBDIRS` macro.

Giving `--cache-file=/dev/null` disables caching, for debugging `configure`. `config.status` only pays attention to the cache file if it is given the `--recheck` option, which makes it rerun `configure`. If you are anticipating a long debugging period, you can also disable cache loading and saving for a `configure` script by redefining the cache macros at the start of `configure.in`:

```
define([AC_CACHE_LOAD], )dnl
define([AC_CACHE_SAVE], )dnl
AC_INIT(whatever)
 ... rest of configure.in ...
```

It is wrong to try to distribute cache files for particular system types. There is too much room for error in doing that, and too much administrative overhead in maintaining them. For any features that can't be guessed automatically, use the standard method of the canonical system type and linking files (see Chapter 8 [Manual Configuration], page 67).

The cache file on a particular system will gradually accumulate whenever someone runs a `configure` script; it will be initially nonexistent. Running `configure` merges the new cache results with the existing cache file. The site initialization script can specify a site-wide cache file to use instead of the default, to make it work transparently, as long as the same C compiler is used every time (see Section 9.5 [Site Defaults], page 75).

## 6.4 Printing Messages

`configure` scripts need to give users running them several kinds of information. The following macros print messages in ways appropriate for each kind. The arguments to all of them get enclosed in shell double quotes, so the shell performs variable and backquote substitution on them.

These macros are all wrappers around the `echo` shell command. `configure` scripts should rarely need to run `echo` directly to print messages for the user. Using these macros makes it easy to change how and when each kind of message is printed; such changes need only be made to the macro definitions, and all of the callers change automatically.

**AC_MSG_CHECKING** (*feature-description*) Macro

Notify the user that `configure` is checking for a particular feature. This macro prints a message that starts with '`checking `' and ends with '`...`' and no newline. It must be followed by a call to `AC_MSG_RESULT` to print the result of the check and the newline. The *feature-description* should be something like '`whether the Fortran compiler accepts C++ comments`' or '`for c89`'.

This macro prints nothing if `configure` is run with the '`--quiet`' or '`--silent`' option.

**AC_MSG_RESULT** (*result-description*) Macro

Notify the user of the results of a check. *result-description* is almost always the value of the cache variable for the check, typically '`yes`', '`no`', or a file name. This macro should follow a call to `AC_MSG_CHECKING`, and the *result-description* should be the completion of the message printed by the call to `AC_MSG_CHECKING`.

This macro prints nothing if `configure` is run with the '`--quiet`' or '`--silent`' option.

**AC_MSG_ERROR** (*error-description*) Macro

Notify the user of an error that prevents `configure` from completing. This macro prints an error message on the standard error output and exits `configure` with a nonzero status. *error-description* should be something like '`invalid value $HOME for \$HOME`'.

**AC_MSG_WARN** (*problem-description*) Macro

Notify the `configure` user of a possible problem. This macro prints the message on the standard error output; `configure` continues running afterward, so macros that call `AC_MSG_WARN` should provide a default (back-up) behavior for the situations they warn about. *problem-description* should be something like '`ln -s seems to make hard links`'.

The following two macros are an obsolete alternative to `AC_MSG_CHECKING` and `AC_MSG_RESULT`.

**AC_CHECKING** (*feature-description*) Macro

This macro is similar to `AC_MSG_CHECKING`, except that it prints a newline after the *feature-description*. It is useful mainly to print a general description of the overall purpose of a group of feature checks, e.g.,

```
AC_CHECKING(if stack overflow is detectable)
```

**AC_VERBOSE** (*result-description*) Macro

This macro is similar to `AC_MSG_RESULT`, except that it is meant to follow a call to `AC_CHECKING` instead of `AC_MSG_CHECKING`; it starts the message it prints with a tab. It is considered obsolete.

# 7 Writing Macros

When you write a feature test that could be applicable to more than one software package, the best thing to do is encapsulate it in a new macro. Here are some instructions and guidelines for writing Autoconf macros.

## 7.1 Macro Definitions

Autoconf macros are defined using the `AC_DEFUN` macro, which is similar to the `m4` builtin `define` macro. In addition to defining a macro, `AC_DEFUN` adds to it some code which is used to constrain the order in which macros are called (see Section 7.4.1 [Prerequisite Macros], page 65).

An Autoconf macro definition looks like this:

```
AC_DEFUN(macro-name, [macro-body])
```

The square brackets here do not indicate optional text: they should literally be present in the macro definition to avoid macro expansion problems (see Section 7.3 [Quoting], page 64). You can refer to any arguments passed to the macro as '`$1`', '`$2`', etc.

To introduce comments in `m4`, use the `m4` builtin `dnl`; it causes `m4` to discard the text through the next newline. It is not needed between macro definitions in '`acsite.m4`' and '`aclocal.m4`', because all output is discarded until `AC_INIT` is called.

See section "How to define new macros" in *GNU m4*, for more complete information on writing `m4` macros.

## 7.2 Macro Names

All of the Autoconf macros have all-uppercase names starting with '`AC_`' to prevent them from accidentally conflicting with other text. All shell variables that they use for internal purposes have mostly-lowercase names starting with '`ac_`'. To ensure that your macros don't conflict with present or future Autoconf macros, you should prefix your own macro names and any shell variables they use with some other sequence. Possibilities include your initials, or an abbreviation for the name of your organization or software package.

Most of the Autoconf macros' names follow a structured naming convention that indicates the kind of feature check by the name. The macro names consist of several words, separated by underscores, going from most general to most specific. The names of their cache variables use the same convention (see Section 6.3.1 [Cache Variable Names], page 58, for more information on them).

The first word of the name after '`AC_`' usually tells the category of feature being tested. Here are the categories used in Autoconf for specific test macros, the kind of macro that you are more likely to write. They are also used for cache variables, in all-lowercase. Use them where applicable; where they're not, invent your own categories.

`C` C language builtin features.

`DECL` Declarations of C variables in header files.

`FUNC` Functions in libraries.

`GROUP` UNIX group owners of files.

`HEADER` Header files.

`LIB` C libraries.

`PATH` The full path names to files, including programs.

`PROG` The base names of programs.

`STRUCT` Definitions of C structures in header files.

`SYS` Operating system features.

`TYPE` C builtin or declared types.

`VAR` C variables in libraries.

After the category comes the name of the particular feature being tested. Any further words in the macro name indicate particular aspects of the feature. For example, `AC_FUNC_UTIME_NULL` checks the behavior of the `utime` function when called with a `NULL` pointer.

A macro that is an internal subroutine of another macro should have a name that starts with the name of that other macro, followed by one or more words saying what the internal macro does. For example, `AC_PATH_X` has internal macros `AC_PATH_X_XMKMF` and `AC_PATH_X_DIRECT`.

## 7.3 Quoting

Macros that are called by other macros are evaluated by `m4` several times; each evaluation might require another layer of quotes to prevent unwanted expansions of macros or `m4` builtins, such as ‘`define`’ and ‘`$1`’. Quotes are also required around macro arguments that contain commas, since commas separate the arguments from each other. It's a good idea to quote any macro arguments that contain newlines or calls to other macros, as well.

Autoconf changes the `m4` quote characters from the default ‘`` ` ``’ and ‘`'`’ to ‘`[`’ and ‘`]`’, because many of the macros use ‘`` ` ``’ and ‘`'`’, mismatched. However, in a few places the macros need to use brackets (usually in C program text or regular expressions). In those places, they use the `m4` builtin command `changequote` to temporarily change the quote characters to ‘`<<`’ and ‘`>>`’. (Sometimes, if they don't need to quote anything, they disable quoting entirely instead by setting the quote characters to empty strings.) Here is an example:

```
AC_TRY_LINK(
changequote(<<, >>)dnl
<<#include <time.h>
#ifndef tzname /* For SGI.  */
extern char *tzname[]; /* RS6000 and others reject char **tzname.  */
#endif>>,
changequote([, ])dnl
[atoi(*tzname);], ac_cv_var_tzname=yes, ac_cv_var_tzname=no)
```

When you create a `configure` script using newly written macros, examine it carefully to check whether you need to add more quotes in your macros. If one or more words have disappeared in the `m4` output, you need more quotes. When in doubt, quote.

However, it's also possible to put on too many layers of quotes. If this happens, the resulting `configure` script will contain unexpanded macros. The `autoconf` program checks for this problem by doing ‘`grep AC_ configure`’.

## 7.4 Dependencies Between Macros

Some Autoconf macros depend on other macros having been called first in order to work correctly. Autoconf provides a way to ensure that certain macros are called if needed and a way to warn the user if macros are called in an order that might cause incorrect operation.

### 7.4.1 Prerequisite Macros

A macro that you write might need to use values that have previously been computed by other macros. For example, `AC_DECL_YYTEXT` examines the output of `flex` or `lex`, so it depends on `AC_PROG_LEX` having been called first to set the shell variable `LEX`.

Rather than forcing the user of the macros to keep track of the dependencies between them, you can use the `AC_REQUIRE` macro to do it automatically. `AC_REQUIRE` can ensure that a macro is only called if it is needed, and only called once.

**AC_REQUIRE** (*macro-name*) Macro

If the `m4` macro *macro-name* has not already been called, call it (without any arguments). Make sure to quote *macro-name* with square brackets. *macro-name* must have been defined using `AC_DEFUN` or else contain a call to `AC_PROVIDE` to indicate that it has been called.

An alternative to using `AC_DEFUN` is to use `define` and call `AC_PROVIDE`. Because this technique does not prevent nested messages, it is considered obsolete.

**AC_PROVIDE** (*this-macro-name*) Macro

Record the fact that *this-macro-name* has been called. *this-macro-name* should be the name of the macro that is calling `AC_PROVIDE`. An easy way to get it is from the `m4` builtin variable `$0`, like this:

```
AC_PROVIDE([$0])
```

### 7.4.2 Suggested Ordering

Some macros should be run before another macro if both are called, but neither *requires* that the other be called. For example, a macro that changes the behavior of the C compiler should be called before any macros that run the C compiler. Many of these dependencies are noted in the documentation.

Autoconf provides the `AC_BEFORE` macro to warn users when macros with this kind of dependency appear out of order in a '`configure.in`' file. The warning occurs when creating `configure` from '`configure.in`', not when running `configure`. For example, `AC_PROG_CPP` checks whether the C compiler can run the C preprocessor when given the '`-E`' option. It should therefore be called after any macros that change which C compiler is being used, such as `AC_PROG_CC`. So `AC_PROG_CC` contains:

```
AC_BEFORE([$0], [AC_PROG_CPP])dnl
```

This warns the user if a call to `AC_PROG_CPP` has already occurred when `AC_PROG_CC` is called.

**AC_BEFORE** (*this-macro-name, called-macro-name*) Macro

Make `m4` print a warning message on the standard error output if *called-macro-name* has already been called. *this-macro-name* should be the name of the macro that is calling `AC_BEFORE`. The macro *called-macro-name* must have been defined using `AC_DEFUN` or else contain a call to `AC_PROVIDE` to indicate that it has been called.

### 7.4.3 Obsolete Macros

Configuration and portability technology has evolved over the years. Often better ways of solving a particular problem are developed, or ad-hoc approaches are systematized. This process has occurred in many parts of Autoconf. One result is that some of the macros are now considered *obsolete*; they still work, but are no longer considered the best thing to do. Autoconf provides the `AC_OBSOLETE` macro to warn users producing `configure` scripts when they use obsolete macros, to encourage them to modernize. A sample call is:

```
AC_OBSOLETE([$0], [; use AC_CHECK_HEADERS(unistd.h) instead])dnl
```

**AC_OBSOLETE** (*this-macro-name* [, *suggestion*]) Macro

Make `m4` print a message on the standard error output warning that *this-macro-name* is obsolete, and giving the file and line number where it was called. *this-macro-name* should be the name of the macro that is calling `AC_OBSOLETE`. If *suggestion* is given, it is printed at the end of the warning message; for example, it can be a suggestion for what to use instead of *this-macro-name*.

# 8 Manual Configuration

A few kinds of features can't be guessed automatically by running test programs. For example, the details of the object file format, or special options that need to be passed to the compiler or linker. You can check for such features using ad-hoc means, such as having `configure` check the output of the `uname` program, or looking for libraries that are unique to particular systems. However, Autoconf provides a uniform method for handling unguessable features.

## 8.1 Specifying the System Type

Like other GNU `configure` scripts, Autoconf-generated `configure` scripts can make decisions based on a canonical name for the system type, which has the form:

*cpu-company-system*

`configure` can usually guess the canonical name for the type of system it's running on. To do so it runs a script called `config.guess`, which derives the name using the `uname` command or symbols predefined by the C preprocessor.

Alternately, the user can specify the system type with command line arguments to `configure`. Doing so is necessary when cross-compiling. In the most complex case of cross-compiling, three system types are involved. The options to specify them are:

`--build=`*build-type*
: the type of system on which the package is being configured and compiled (rarely needed);

`--host=`*host-type*
: the type of system on which the package will run;

`--target=`*target-type*
: the type of system for which any compiler tools in the package will produce code.

If the user gives `configure` a non-option argument, it is used as the default for the host, target, and build system types if the user does not specify them explicitly with options. The target and build types default to the host type if it is given and they are not. If you are cross-compiling, you still have to specify the names of the cross-tools you use, in particular the C compiler, on the `configure` command line, e.g.,

```
CC=m68k-coff-gcc configure --target=m68k-coff
```

`configure` recognizes short aliases for many system types; for example, '`decstation`' can be given on the command line instead of '`mips-dec-ultrix4.2`'. `configure` runs a script called `config.sub` to canonicalize system type aliases.

## 8.2 Getting the Canonical System Type

The following macros make the system type available to `configure` scripts. They run the shell script `config.guess` to determine any values for the host, target, and build types that they need and the user did not specify on the command line. They run `config.sub` to canonicalize any aliases the user gave. If you use these macros, you must distribute those two shell scripts along with your source code. See Section 3.2 [Output], page 11, for information about the `AC_CONFIG_AUX_DIR` macro which you can use to control which directory `configure` looks for those scripts in. If you do not use either of these macros, `configure` ignores any '`--host`', '`--target`', and '`--build`' options given to it.

**AC_CANONICAL_SYSTEM** Macro

Determine the system type and set output variables to the names of the canonical system types. See Section 8.3 [System Type Variables], page 68, for details about the variables this macro sets.

**AC_CANONICAL_HOST** Macro

Perform only the subset of `AC_CANONICAL_SYSTEM` relevant to the host type. This is all that is needed for programs that are not part of a compiler toolchain.

## 8.3 System Type Variables

After calling `AC_CANONICAL_SYSTEM`, the following output variables contain the system type information. After `AC_CANONICAL_HOST`, only the `host` variables below are set.

`build`, `host`, `target`
: the canonical system names;

`build_alias`, `host_alias`, `target_alias`
: the names the user specified, or the canonical names if `config.guess` was used;

`build_cpu`, `build_vendor`, `build_os`
`host_cpu`, `host_vendor`, `host_os`
`target_cpu`, `target_vendor`, `target_os`
: the individual parts of the canonical names (for convenience).

## 8.4 Using the System Type

How do you use a canonical system type? Usually, you use it in one or more `case` statements in '`configure.in`' to select system-specific C files. Then link those files, which have names based on the system name, to generic names, such as '`host.h`' or '`target.c`'. The `case` statement patterns can use shell wildcards to group several cases together, like in this fragment:

```
case "$target" in
i386-*-mach* | i386-*-gnu*) obj_format=aout emulation=mach bfd_gas=yes ;;
i960-*-bout) obj_format=bout ;;
esac
```

**AC_LINK_FILES** (*source*..., *dest*...) Macro

Make `AC_OUTPUT` link each of the existing files *source* to the corresponding link name *dest*. Makes a symbolic link if possible, otherwise a hard link. The *dest* and *source* names should be relative to the top level source or build directory.

For example, this call:

```
AC_LINK_FILES(config/${machine}.h config/${obj_format}.h, host.h object.h)
```

creates in the current directory '`host.h`', which is a link to '*srcdir*`/config/${machine}.h`', and '`object.h`', which is a link to '*srcdir*`/config/${obj_format}.h`'.

You can also use the host system type to find cross-compilation tools. See Section 4.1.2 [Generic Programs], page 26, for information about the `AC_CHECK_TOOL` macro which does that.

# 9 Site Configuration

`configure` scripts support several kinds of local configuration decisions. There are ways for users to specify where external software packages are, include or exclude optional features, install programs under modified names, and set default values for `configure` options.

## 9.1 Working With External Software

Some packages require, or can optionally use, other software packages which are already installed. The user can give `configure` command line options to specify which such external software to use. The options have one of these forms:

`--with-`*package*[=*arg*]
`--without-`*package*

For example, '`--with-gnu-ld`' means work with the GNU linker instead of some other linker. '`--with-x`' means work with The X Window System.

The user can give an argument by following the package name with '`=`' and the argument. Giving an argument of '`no`' is for packages that are used by default; it says to *not* use the package. An argument that is neither '`yes`' nor '`no`' could include a name or number of a version of the other package, to specify more precisely which other package this program is supposed to work with. If no argument is given, it defaults to '`yes`'. '`--without-`*package*' is equivalent to '`--with-`*package*`=no`'.

For each external software package that may be used, '`configure.in`' should call `AC_ARG_WITH` to detect whether the `configure` user asked to use it. Whether each package is used or not by default, and which arguments are valid, is up to you.

**AC_ARG_WITH** (*package*, *help-string* [, *action-if-given* [, *action-if-not-given*]]) Macro

If the user gave `configure` the option '`--with-`*package*' or '`--without-`*package*', run shell commands *action-if-given*. If neither option was given, run shell commands *action-if-not-given*. The name *package* indicates another software package that this program should work with. It should consist only of alphanumeric characters and dashes.

The option's argument is available to the shell commands *action-if-given* in the shell variable `withval`, which is actually just the value of the shell variable `with_`*package*, with any '-' characters changed into '_'. You may use that variable instead, if you wish.

The argument *help-string* is a description of the option which looks like this:

```
     --with-readline         support fancy command line editing
```

*help-string* may be more than one line long, if more detail is needed. Just make sure the columns line up in '`configure --help`'. Avoid tabs in the help string. You'll need to enclose it in '`[`' and '`]`' in order to produce the leading spaces.

**AC_WITH** (*package, action-if-given* [, *action-if-not-given*]) Macro

This is an obsolete version of `AC_ARG_WITH` that does not support providing a help string.

## 9.2 Choosing Package Options

If a software package has optional compile-time features, the user can give `configure` command line options to specify whether to compile them. The options have one of these forms:

`--enable-`*feature*[`=`*arg*]
`--disable-`*feature*

These options allow users to choose which optional features to build and install. '`--enable-`*feature*' options should never make a feature behave differently or cause one feature to replace another. They should only cause parts of the program to be built rather than left out.

The user can give an argument by following the feature name with '`=`' and the argument. Giving an argument of '`no`' requests that the feature *not* be made available. A feature with an argument looks like '`--enable-debug=stabs`'. If no argument is given, it defaults to '`yes`'. '`--disable-`*feature*' is equivalent to '`--enable-`*feature*`=no`'.

For each optional feature, '`configure.in`' should call `AC_ARG_ENABLE` to detect whether the `configure` user asked to include it. Whether each feature is included or not by default, and which arguments are valid, is up to you.

**AC_ARG_ENABLE** (*feature, help-string* [, *action-if-given* [, *action-if-not-given*]]) Macro

If the user gave `configure` the option '`--enable-`*feature*' or '`--disable-`*feature*', run shell commands *action-if-given*. If neither option was given, run shell commands *action-if-not-given*. The name *feature* indicates an optional user-level facility. It should consist only of alphanumeric characters and dashes.

The option's argument is available to the shell commands *action-if-given* in the shell variable `enableval`, which is actually just the value of the shell variable `enable_`*package*, with any '`-`' characters changed into '`_`'. You may use that variable instead, if you wish. The *help-string* argument is like that of `AC_ARG_WITH` (see Section 9.1 [External Software], page 70).

**AC_ENABLE** (*feature, action-if-given* [, *action-if-not-given*]) Macro

This is an obsolete version of `AC_ARG_ENABLE` that does not support providing a help string.

## 9.3 Configuring Site Details

Some software packages require complex site-specific information. Some examples are host names to use for certain services, company names, and email addresses to contact. Since some configuration scripts generated by Metaconfig ask for such information interactively, people sometimes wonder how to get that information in Autoconf-generated configuration scripts, which aren't interactive.

Such site configuration information should be put in a file that is edited *only by users*, not by programs. The location of the file can either be based on the `prefix` variable, or be a standard location such as the user's home directory. It could even be specified by an environment variable. The programs should examine that file at run time, rather than at compile time. Run time configuration is more convenient for users and makes the configuration process simpler than getting the information while configuring. See section "Variables for Installation Directories" in *GNU Coding Standards*, for more information on where to put data files.

## 9.4 Transforming Program Names When Installing

Autoconf supports changing the names of programs when installing them. In order to use these transformations, '`configure.in`' must call the macro `AC_ARG_PROGRAM`.

**AC_ARG_PROGRAM** Macro

Place in output variable `program_transform_name` a sequence of `sed` commands for changing the names of installed programs.

If any of the options described below are given to `configure`, program names are transformed accordingly. Otherwise, if `AC_CANONICAL_SYSTEM` has been called and a '`--target`' value is given that differs from the host type (specified with '`--host`' or defaulted by `config.sub`), the target type followed by a dash is used as a prefix. Otherwise, no program name transformation is done.

### 9.4.1 Transformation Options

You can specify name transformations by giving `configure` these command line options:

`--program-prefix=`*prefix*
: prepend *prefix* to the names;

`--program-suffix=`*suffix*
: append *suffix* to the names;

`--program-transform-name=`*expression*
: perform `sed` substitution *expression* on the names.

### 9.4.2 Transformation Examples

These transformations are useful with programs that can be part of a cross-compilation development environment. For example, a cross-assembler running on a Sun 4 configured with '`--target=i960-vxworks`' is normally installed as '`i960-vxworks-as`', rather than '`as`', which could be confused with a native Sun 4 assembler.

You can force a program name to begin with '`g`', if you don't want GNU programs installed on your system to shadow other programs with the same name. For example, if you configure GNU `diff` with '`--program-prefix=g`', then when you run '`make install`' it is installed as '`/usr/local/bin/gdiff`'.

As a more sophisticated example, you could use

```
--program-transform-name='s/^/g/; s/^gg/g/; s/^gless/less/'
```

to prepend '`g`' to most of the program names in a source tree, excepting those like `gdb` that already have one and those like `less` and `lesskey` that aren't GNU programs. (That is assuming that you have a source tree containing those programs that is set up to use this feature.)

One way to install multiple versions of some programs simultaneously is to append a version number to the name of one or both. For example, if you want to keep Autoconf version 1 around for awhile, you can configure Autoconf version 2 using '`--program-suffix=2`' to install the programs as '`/usr/local/bin/autoconf2`', '`/usr/local/bin/autoheader2`', etc.

### 9.4.3 Transformation Rules

Here is how to use the variable `program_transform_name` in a '`Makefile.in`':

```
transform=@program_transform_name@
install: all
        $(INSTALL_PROGRAM) myprog $(bindir)/`echo myprog|sed '$(transform)'`

uninstall:
        rm -f $(bindir)/`echo myprog|sed '$(transform)'`
```

If you have more than one program to install, you can do it in a loop:

```
PROGRAMS=cp ls rm
install:
        for p in $(PROGRAMS); do \
          $(INSTALL_PROGRAM) $$p $(bindir)/`echo $$p|sed '$(transform)'`; \
        done

uninstall:
        for p in $(PROGRAMS); do \
          rm -f $(bindir)/`echo $$p|sed '$(transform)'`; \
        done
```

Whether to do the transformations on documentation files (Texinfo or `man`) is a tricky question; there seems to be no perfect answer, due to the several reasons for name transforming. Documentation is not usually particular to a specific architecture, and Texinfo files do not conflict with system documentation. But they might conflict with earlier versions of the same files, and `man` pages sometimes do conflict with system documentation. As a compromise, it is probably best to do name transformations on `man` pages but not on Texinfo manuals.

## 9.5 Setting Site Defaults

Autoconf-generated `configure` scripts allow your site to provide default values for some configuration values. You do this by creating site- and system-wide initialization files.

If the environment variable `CONFIG_SITE` is set, `configure` uses its value as the name of a shell script to read. Otherwise, it reads the shell script '*prefix*`/share/config.site`' if it exists, then '*prefix*`/etc/config.site`' if it exists. Thus, settings in machine-specific files override those in machine-independent ones in case of conflict.

Site files can be arbitrary shell scripts, but only certain kinds of code are really appropriate to be in them. Because `configure` reads any cache file after it has read any site files, a site file can define a default cache file to be shared between all Autoconf-generated `configure` scripts run on that system. If you set a default cache file in a site file, it is a good idea to also set the output variable `CC` in that site file, because the cache file is only valid for a particular compiler, but many systems have several available.

You can examine or override the value set by a command line option to `configure` in a site file; options set shell variables that have the same names as the options, with any dashes turned into underscores. The exceptions are that '`--without-`' and '`--disable-`' options are like giving the corresponding '`--with-`' or '`--enable-`' option and the value '`no`'. Thus, '`--cache-file=localcache`' sets the variable `cache_file` to the value '`localcache`'; '`--enable-warnings=no`' or '`--disable-warnings`' sets the variable `enable_warnings` to the value '`no`'; '`--prefix=/usr`' sets the variable `prefix` to the value '`/usr`'; etc.

Site files are also good places to set default values for other output variables, such as `CFLAGS`, if you need to give them non-default values: anything you would normally do, repetitively, on the command line. If you use non-default values for *prefix* or *exec_prefix* (wherever you locate the site file), you can set them in the site file if you specify it with the `CONFIG_SITE` environment variable.

You can set some cache values in the site file itself. Doing this is useful if you are cross-compiling, so it is impossible to check features that require running a test program. You could "prime the cache" by setting those values correctly for that system in '*prefix*`/etc/config.site`'. To find out the names of the cache variables you need to set, look for shell variables with '`_cv_`' in their names in the affected `configure` scripts, or in the Autoconf `m4` source code for those macros.

The cache file is careful to not override any variables set in the site files. Similarly, you should not override command-line options in the site files. Your code should check that variables such as `prefix` and `cache_file` have their default values (as set near the top of `configure`) before changing them.

Here is a sample file '`/usr/share/local/gnu/share/config.site`'. The command '`configure --prefix=/usr/share/local/gnu`' would read this file (if `CONFIG_SITE` is not set to a different file).

```
# config.site for configure
#
# Default --prefix and --exec-prefix.
test "$prefix" = NONE && prefix=/usr/share/local/gnu
test "$exec_prefix" = NONE && exec_prefix=/usr/local/gnu
#
# Give Autoconf 2.x generated configure scripts a shared default
# cache file for feature test results, architecture-specific.
if test "$cache_file" = ./config.cache; then
  cache_file="$prefix/var/config.cache"
  # A cache file is only valid for one C compiler.
  CC=gcc
fi
```

# 10 Running `configure` Scripts

Below are instructions on how to configure a package that uses a `configure` script, suitable for inclusion as an '`INSTALL`' file in the package. A plain-text version of '`INSTALL`' which you may use comes with Autoconf.

## 10.1 Basic Installation

These are generic installation instructions.

The `configure` shell script attempts to guess correct values for various system-dependent variables used during compilation. It uses those values to create a '`Makefile`' in each directory of the package. It may also create one or more '`.h`' files containing system-dependent definitions. Finally, it creates a shell script '`config.status`' that you can run in the future to recreate the current configuration, a file '`config.cache`' that saves the results of its tests to speed up reconfiguring, and a file '`config.log`' containing compiler output (useful mainly for debugging `configure`).

If you need to do unusual things to compile the package, please try to figure out how `configure` could check whether to do them, and mail diffs or instructions to the address given in the '`README`' so they can be considered for the next release. If at some point '`config.cache`' contains results you don't want to keep, you may remove or edit it.

The file '`configure.in`' is used to create '`configure`' by a program called `autoconf`. You only need '`configure.in`' if you want to change it or regenerate '`configure`' using a newer version of `autoconf`.

The simplest way to compile this package is:

1. `cd` to the directory containing the package's source code and type '`./configure`' to configure the package for your system. If you're using `csh` on an old version of System V, you might need to type '`sh ./configure`' instead to prevent `csh` from trying to execute `configure` itself.

   Running `configure` takes awhile. While running, it prints some messages telling which features it is checking for.
2. Type '`make`' to compile the package.
3. Optionally, type '`make check`' to run any self-tests that come with the package.
4. Type '`make install`' to install the programs and any data files and documentation.

5. You can remove the program binaries and object files from the source code directory by typing '`make clean`'. To also remove the files that `configure` created (so you can compile the package for a different kind of computer), type '`make distclean`'. There is also a '`make maintainer-clean`' target, but that is intended mainly for the package's developers. If you use it, you may have to get all sorts of other programs in order to regenerate files that came with the distribution.

## 10.2 Compilers and Options

Some systems require unusual options for compilation or linking that the `configure` script does not know about. You can give `configure` initial values for variables by setting them in the environment. Using a Bourne-compatible shell, you can do that on the command line like this:

```
CC=c89 CFLAGS=-O2 LIBS=-lposix ./configure
```

Or on systems that have the `env` program, you can do it like this:

```
env CPPFLAGS=-I/usr/local/include LDFLAGS=-s ./configure
```

## 10.3 Compiling For Multiple Architectures

You can compile the package for more than one kind of computer at the same time, by placing the object files for each architecture in their own directory. To do this, you must use a version of `make` that supports the `VPATH` variable, such as GNU `make`. `cd` to the directory where you want the object files and executables to go and run the `configure` script. `configure` automatically checks for the source code in the directory that `configure` is in and in '`..`'.

If you have to use a `make` that does not supports the `VPATH` variable, you have to compile the package for one architecture at a time in the source code directory. After you have installed the package for one architecture, use '`make distclean`' before reconfiguring for another architecture.

## 10.4 Installation Names

By default, '`make install`' will install the package's files in '`/usr/local/bin`', '`/usr/local/man`', etc. You can specify an installation prefix other than '`/usr/local`' by giving `configure` the option '`--prefix=`*path*'.

You can specify separate installation prefixes for architecture-specific files and architecture-independent files. If you give `configure` the option '`--exec-prefix=`*path*', the package will use *path* as the prefix for installing programs and libraries. Documentation and other data files will still use the regular prefix.

In addition, if you use an unusual directory layout you can give options like '`--bindir=`*path*' to specify different values for particular kinds of files. Run '`configure --help`' for a list of the directories you can set and what kinds of files go in them.

If the package supports it, you can cause programs to be installed with an extra prefix or suffix on their names by giving `configure` the option '`--program-prefix=`*PREFIX*' or '`--program-suffix=`*SUFFIX*'.

## 10.5 Optional Features

Some packages pay attention to '`--enable-`*feature*' options to `configure`, where *feature* indicates an optional part of the package. They may also pay attention to '`--with-`*package*' options, where *package* is something like '`gnu-as`' or '`x`' (for the X Window System). The '`README`' should mention any '`--enable-`' and '`--with-`' options that the package recognizes.

For packages that use the X Window System, `configure` can usually find the X include and library files automatically, but if it doesn't, you can use the `configure` options '`--x-includes=`*dir*' and '`--x-libraries=`*dir*' to specify their locations.

## 10.6 Specifying the System Type

There may be some features `configure` can not figure out automatically, but needs to determine by the type of host the package will run on. Usually `configure` can figure that out, but if it prints a message saying it can not guess the host type, give it the '`--host=`*type*' option. *type* can either be a short name for the system type, such as '`sun4`', or a canonical name with three fields:

*cpu-company-system*

See the file '`config.sub`' for the possible values of each field. If '`config.sub`' isn't included in this package, then this package doesn't need to know the host type.

If you are building compiler tools for cross-compiling, you can also use the '`--target=`*type*' option to select the type of system they will produce code for and the '`--build=`*type*' option to select the type of system on which you are compiling the package.

## 10.7 Sharing Defaults

If you want to set default values for `configure` scripts to share, you can create a site shell script called '`config.site`' that gives default values for variables like `CC`, `cache_file`, and `prefix`. `configure` looks for '*prefix*`/share/config.site`' if it exists, then '*prefix*`/etc/config.site`' if it exists. Or, you can set the `CONFIG_SITE` environment variable to the location of the site script. A warning: not all `configure` scripts look for a site script.

## 10.8 Operation Controls

`configure` recognizes the following options to control how it operates.

`--cache-file=`*file*
: Use and save the results of the tests in *file* instead of '`./config.cache`'. Set *file* to '`/dev/null`' to disable caching, for debugging `configure`.

`--help`
: Print a summary of the options to `configure`, and exit.

`--quiet`
`--silent`
`-q`
: Do not print messages saying which checks are being made.

`--srcdir=`*dir*
: Look for the package's source code in directory *dir*. Usually `configure` can determine that directory automatically.

`--version`
: Print the version of Autoconf used to generate the `configure` script, and exit.

`configure` also accepts some other, not widely useful, options.

# 11 Recreating a Configuration

The `configure` script creates a file named '`config.status`' which describes which configuration options were specified when the package was last configured. This file is a shell script which, if run, will recreate the same configuration.

You can give '`config.status`' the '`--recheck`' option to update itself. This option is useful if you change `configure`, so that the results of some tests might be different from the previous run. The '`--recheck`' option re-runs `configure` with the same arguments you used before, plus the '`--no-create`' option, which prevent `configure` from running '`config.status`' and creating '`Makefile`' and other files, and the '`--no-recursion`' option, which prevents `configure` from running other `configure` scripts in subdirectories. (This is so other '`Makefile`' rules can run '`config.status`' when it changes; see Section 3.3.3 [Automatic Remaking], page 16, for an example).

'`config.status`' also accepts the options '`--help`', which prints a summary of the options to '`config.status`', and '`--version`', which prints the version of Autoconf used to create the `configure` script that generated '`config.status`'.

'`config.status`' checks several optional environment variables that can alter its behavior:

**CONFIG_SHELL** Variable

The shell with which to run `configure` for the '`--recheck`' option. It must be Bourne-compatible. The default is '`/bin/sh`'.

**CONFIG_STATUS** Variable

The file name to use for the shell script that records the configuration. The default is '`./config.status`'. This variable is useful when one package uses parts of another and the `configure` scripts shouldn't be merged because they are maintained separately.

The following variables provide one way for separately distributed packages to share the values computed by `configure`. Doing so can be useful if some of the packages need a superset of the features that one of them, perhaps a common library, does. These variables allow a '`config.status`' file to create files other than the ones that its '`configure.in`' specifies, so it can be used for a different package.

**CONFIG_FILES** Variable

The files in which to perform '@*variable*@' substitutions. The default is the arguments given to AC_OUTPUT in 'configure.in'.

**CONFIG_HEADERS** Variable

The files in which to substitute C #define statements. The default is the arguments given to AC_CONFIG_HEADER; if that macro was not called, 'config.status' ignores this variable.

These variables also allow you to write 'Makefile' rules that regenerate only some of the files. For example, in the dependencies given above (see Section 3.3.3 [Automatic Remaking], page 16), 'config.status' is run twice when 'configure.in' has changed. If that bothers you, you can make each run only regenerate the files for that rule:

```
config.h: stamp-h
stamp-h: config.h.in config.status
        CONFIG_FILES= CONFIG_HEADERS=config.h ./config.status
        echo > stamp-h

Makefile: Makefile.in config.status
        CONFIG_FILES=Makefile CONFIG_HEADERS= ./config.status
```

(If 'configure.in' does not call AC_CONFIG_HEADER, there is no need to set CONFIG_HEADERS in the make rules.)

# 12 Questions About Autoconf

Several questions about Autoconf come up occasionally. Here some of them are addressed.

## 12.1 Distributing `configure` Scripts

> What are the restrictions on distributing `configure` scripts that Autoconf generates? How does that affect my programs that use them?

There are no restrictions on how the configuration scripts that Autoconf produces may be distributed or used. In Autoconf version 1, they were covered by the GNU General Public License. We still encourage software authors to distribute their work under terms like those of the GPL, but doing so is not required to use Autoconf.

Of the other files that might be used with `configure`, '`config.h.in`' is under whatever copyright you use for your '`configure.in`', since it is derived from that file and from the public domain file '`acconfig.h`'. '`config.sub`' and '`config.guess`' have an exception to the GPL when they are used with an Autoconf-generated `configure` script, which permits you to distribute them under the same terms as the rest of your package. '`install-sh`' is from the X Consortium and is not copyrighted.

## 12.2 Why Require GNU `m4`?

> Why does Autoconf require GNU `m4`?

Many `m4` implementations have hard-coded limitations on the size and number of macros, which Autoconf exceeds. They also lack several builtin macros that it would be difficult to get along without in a sophisticated application like Autoconf, including:

```
builtin
indir
patsubst
__file__
__line__
```

Since only software maintainers need to use Autoconf, and since GNU `m4` is simple to configure and install, it seems reasonable to require GNU `m4` to be installed also. Many maintainers of GNU and other free software already have most of the GNU utilities installed, since they prefer them.

## 12.3 How Can I Bootstrap?

> If Autoconf requires GNU `m4` and GNU `m4` has an Autoconf `configure` script, how do I bootstrap? It seems like a chicken and egg problem!

This is a misunderstanding. Although GNU `m4` does come with a `configure` script produced by Autoconf, Autoconf is not required in order to run the script and install GNU `m4`. Autoconf is only required if you want to change the `m4` `configure` script, which few people have to do (mainly its maintainer).

## 12.4 Why Not Imake?

> Why not use Imake instead of `configure` scripts?

Several people have written addressing this question, so I include adaptations of their explanations here.

The following answer is based on one written by Richard Pixley:

Autoconf generated scripts frequently work on machines which it has never been set up to handle before. That is, it does a good job of inferring a configuration for a new system. Imake cannot do this.

Imake uses a common database of host specific data. For X11, this makes sense because the distribution is made as a collection of tools, by one central authority who has control over the database.

GNU tools are not released this way. Each GNU tool has a maintainer; these maintainers are scattered across the world. Using a common database would be a maintenance nightmare. Autoconf may appear to be this kind of database, but in fact it is not. Instead of listing host dependencies, it lists program requirements.

If you view the GNU suite as a collection of native tools, then the problems are similar. But the GNU development tools can be configured as cross tools in almost any host+target permutation. All of these configurations can be installed concurrently. They can even be configured to share host independent files across hosts. Imake doesn't address these issues.

Imake templates are a form of standardization. The GNU coding standards address the same issues without necessarily imposing the same restrictions.

Here is some further explanation, written by Per Bothner:

One of the advantages of Imake is that it easy to generate large Makefiles using `cpp`'s '`#include`' and macro mechanisms. However, `cpp` is not programmable: it has limited conditional facilities, and no looping. And `cpp` cannot inspect its environment.

All of these problems are solved by using `sh` instead of `cpp`. The shell is fully programmable, has macro substitution, can execute (or source) other shell scripts, and can inspect its environment.

Paul Eggert elaborates more:

With Autoconf, installers need not assume that Imake itself is already installed and working well. This may not seem like much of an advantage to people who are accustomed to Imake. But on many hosts Imake is not installed or the default installation is not working well, and requiring Imake to install a package hinders the acceptance of that package on those hosts. For example, the Imake template and configuration files might not be installed properly on a host, or the Imake build procedure might wrongly assume that all source files are in one big directory tree, or the Imake configuration might assume one compiler whereas the package or the installer needs to use another, or there might be a version mismatch between the Imake expected by the package and the Imake supported by the host. These problems are much rarer with Autoconf, where each package comes with its own independent configuration processor.

Also, Imake often suffers from unexpected interactions between `make` and the installer's C preprocessor. The fundamental problem here is that the C preprocessor was designed to preprocess C programs, not '`Makefile`'s. This is much less of a problem with Autoconf, which uses the general-purpose preprocessor `m4`, and where the package's author (rather than the installer) does the preprocessing in a standard way.

Finally, Mark Eichin notes:

Imake isn't all that extensible, either. In order to add new features to Imake, you need to provide your own project template, and duplicate most of the features of the existing one. This means that for a sophisticated project, using the vendor-provided Imake templates fails to provide any leverage—since they don't cover anything that your own project needs (unless it is an X11 program).

On the other side, though:

The one advantage that Imake has over `configure`: '`Imakefile`'s tend to be much shorter (likewise, less redundant) than '`Makefile.in`'s. There is a fix to this, however—at least for the Kerberos V5 tree, we've modified things to call in common '`post.in`' and '`pre.in`' '`Makefile`' fragments for the entire tree. This means that a lot of common things don't have to be duplicated, even though they normally are in `configure` setups.

# 13 Upgrading From Version 1

Autoconf version 2 is mostly backward compatible with version 1. However, it introduces better ways to do some things, and doesn't support some of the ugly things in version 1. So, depending on how sophisticated your 'configure.in' files are, you might have to do some manual work in order to upgrade to version 2. This chapter points out some problems to watch for when upgrading. Also, perhaps your configure scripts could benefit from some of the new features in version 2; the changes are summarized in the file 'NEWS' in the Autoconf distribution.

First, make sure you have GNU m4 version 1.1 or higher installed, preferably 1.3 or higher. Versions before 1.1 have bugs that prevent them from working with Autoconf version 2. Versions 1.3 and later are much faster than earlier versions, because as of version 1.3, GNU m4 has a more efficient implementation of diversions and can freeze its internal state in a file that it can read back quickly.

## 13.1 Changed File Names

If you have an 'aclocal.m4' installed with Autoconf (as opposed to in a particular package's source directory), you must rename it to 'acsite.m4'. See Section 2.4 [Invoking autoconf], page 7.

If you distribute 'install.sh' with your package, rename it to 'install-sh' so make builtin rules won't inadvertently create a file called 'install' from it. AC_PROG_INSTALL looks for the script under both names, but it is best to use the new name.

If you were using 'config.h.top' or 'config.h.bot', you still can, but you will have less clutter if you merge them into 'acconfig.h'. See Section 3.4.2 [Invoking autoheader], page 19.

## 13.2 Changed Makefiles

Add '@CFLAGS@', '@CPPFLAGS@', and '@LDFLAGS@' in your 'Makefile.in' files, so they can take advantage of the values of those variables in the environment when configure is run. Doing this isn't necessary, but it's a convenience for users.

Also add '@configure_input@' in a comment to each non-'Makefile' input file for AC_OUTPUT, so that the output files will contain a comment saying they were produced by configure. Automatically selecting the right comment syntax for all the kinds of files that people call `AC_OUTPUT` on became too much work.

Add '`config.log`' and '`config.cache`' to the list of files you remove in `distclean` targets.

If you have the following in '`Makefile.in`':

```
prefix = /usr/local
exec_prefix = ${prefix}
```

you must change it to:

```
prefix = @prefix@
exec_prefix = @exec_prefix@
```

The old behavior of replacing those variables without '`@`' characters around them has been removed.

## 13.3 Changed Macros

Many of the macros were renamed in Autoconf version 2. You can still use the old names, but the new ones are clearer, and it's easier to find the documentation for them. See Chapter 15 [Old Macro Names], page 96, for a table showing the new names for the old macros. Use the `autoupdate` program to convert your '`configure.in`' to using the new macro names. See Section 13.4 [Invoking autoupdate], page 89.

Some macros have been superseded by similar ones that do the job better, but are not call-compatible. If you get warnings about calling obsolete macros while running `autoconf`, you may safely ignore them, but your `configure` script will generally work better if you follow the advice it prints about what to replace the obsolete macros with. In particular, the mechanism for reporting the results of tests has changed. If you were using `echo` or `AC_VERBOSE` (perhaps via `AC_COMPILE_CHECK`), your `configure` script's output will look better if you switch to `AC_MSG_CHECKING` and `AC_MSG_RESULT`. See Section 6.4 [Printing Messages], page 60. Those macros work best in conjunction with cache variables. See Section 6.3 [Caching Results], page 57.

## 13.4 Using `autoupdate` to Modernize `configure`

The `autoupdate` program updates a '`configure.in`' file that calls Autoconf macros by their old names to use the current macro names. In version 2 of Autoconf, most of the macros were renamed to use a more uniform and descriptive naming scheme. See Section 7.2 [Macro Names], page 62, for a description of the new scheme. Although the old names still work (see Chapter 15 [Old Macro Names], page 96, for a list of the old macro names and the corresponding new names), you can make your '`configure.in`' files more readable and make it easier to use the current Autoconf documentation if you update them to use the new macro names.

If given no arguments, `autoupdate` updates '`configure.in`', backing up the original version with the suffix '`~`' (or the value of the environment variable `SIMPLE_BACKUP_SUFFIX`, if that is set). If you give `autoupdate` an argument, it reads that file instead of '`configure.in`' and writes the updated file to the standard output.

`autoupdate` accepts the following options:

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

`--macrodir=`*dir*
`-m` *dir* Look for the Autoconf macro files in directory *dir* instead of the default installation directory. You can also set the `AC_MACRODIR` environment variable to a directory; this option overrides the environment variable.

`--version`
Print the version number of `autoupdate` and exit.

## 13.5 Changed Results

If you were checking the results of previous tests by examining the shell variable `DEFS`, you need to switch to checking the values of the cache variables for those tests. `DEFS` no longer exists while `configure` is running; it is only created when generating output files. This difference from version 1 is because properly quoting the contents of that variable turned out to be too cumbersome and inefficient to do every time `AC_DEFINE` is called. See Section 6.3.1 [Cache Variable Names], page 58.

For example, here is a '`configure.in`' fragment written for Autoconf version 1:

```
AC_HAVE_FUNCS(syslog)
```

```
case "$DEFS" in
*-DHAVE_SYSLOG*) ;;
*) # syslog is not in the default libraries.  See if it's in some other.
  saved_LIBS="$LIBS"
  for lib in bsd socket inet; do
    AC_CHECKING(for syslog in -l$lib)
    LIBS="$saved_LIBS -l$lib"
    AC_HAVE_FUNCS(syslog)
    case "$DEFS" in
    *-DHAVE_SYSLOG*) break ;;
    *) ;;
    esac
    LIBS="$saved_LIBS"
  done ;;
esac
```

Here is a way to write it for version 2:

```
AC_CHECK_FUNCS(syslog)
if test $ac_cv_func_syslog = no; then
  # syslog is not in the default libraries.  See if it's in some other.
  for lib in bsd socket inet; do
    AC_CHECK_LIB($lib, syslog, [AC_DEFINE(HAVE_SYSLOG)
      LIBS="$LIBS $lib"; break])
  done
fi
```

If you were working around bugs in `AC_DEFINE_UNQUOTED` by adding backslashes before quotes, you need to remove them. It now works predictably, and does not treat quotes (except backquotes) specially. See Section 6.2 [Setting Output Variables], page 56.

All of the boolean shell variables set by Autoconf macros now use '`yes`' for the true value. Most of them use '`no`' for false, though for backward compatibility some use the empty string instead. If you were relying on a shell variable being set to something like 1 or '`t`' for true, you need to change your tests.

## 13.6 Changed Macro Writing

When defining your own macros, you should now use `AC_DEFUN` instead of `define`. `AC_DEFUN` automatically calls `AC_PROVIDE` and ensures that macros called via `AC_REQUIRE` do not interrupt other macros, to prevent nested '`checking...`' messages on the screen. There's no actual harm in continuing to use the older way, but it's less convenient and attractive. See Section 7.1 [Macro Definitions], page 62.

You probably looked at the macros that came with Autoconf as a guide for how to do things. It would be a good idea to take a look at the new versions of them, as the style is somewhat improved and they take advantage of some new features.

If you were doing tricky things with undocumented Autoconf internals (macros, variables, diversions), check whether you need to change anything to account for changes that have been made. Perhaps you can even use an officially supported technique in version 2 instead of kludging. Or perhaps not.

To speed up your locally written feature tests, add caching to them. See whether any of your tests are of general enough usefulness to encapsulate into macros that you can share.

# 14 History of Autoconf

You may be wondering, Why was Autoconf originally written? How did it get into its present form? (Why does it look like gorilla spit?) If you're not wondering, then this chapter contains no information useful to you, and you might as well skip it. If you *are* wondering, then let there be light...

## 14.1 Genesis

In June 1991 I was maintaining many of the GNU utilities for the Free Software Foundation. As they were ported to more platforms and more programs were added, the number of '`-D`' options that users had to select in the '`Makefile`' (around 20) became burdensome. Especially for me—I had to test each new release on a bunch of different systems. So I wrote a little shell script to guess some of the correct settings for the fileutils package, and released it as part of fileutils 2.0. That `configure` script worked well enough that the next month I adapted it (by hand) to create similar `configure` scripts for several other GNU utilities packages. Brian Berliner also adapted one of my scripts for his CVS revision control system.

Later that summer, I learned that Richard Stallman and Richard Pixley were developing similar scripts to use in the GNU compiler tools; so I adapted my `configure` scripts to support their evolving interface: using the file name '`Makefile.in`' as the templates; adding '`+srcdir`', the first option (of many); and creating '`config.status`' files.

## 14.2 Exodus

As I got feedback from users, I incorporated many improvements, using Emacs to search and replace, cut and paste, similar changes in each of the scripts. As I adapted more GNU utilities packages to use `configure` scripts, updating them all by hand became impractical. Rich Murphey, the maintainer of the GNU graphics utilities, sent me mail saying that the `configure` scripts were great, and asking if I had a tool for generating them that I could send him. No, I thought, but I should! So I started to work out how to generate them. And the journey from the slavery of hand-written `configure` scripts to the abundance and ease of Autoconf began.

Cygnus `configure`, which was being developed at around that time, is table driven; it is meant to deal mainly with a discrete number of system types with a small number of mainly unguessable features (such as details of the object file format). The automatic configuration system that Brian

Fox had developed for Bash takes a similar approach. For general use, it seems to me a hopeless cause to try to maintain an up-to-date database of which features each variant of each operating system has. It's easier and more reliable to check for most features on the fly—especially on hybrid systems that people have hacked on locally or that have patches from vendors installed.

I considered using an architecture similar to that of Cygnus `configure`, where there is a single `configure` script that reads pieces of '`configure.in`' when run. But I didn't want to have to distribute all of the feature tests with every package, so I settled on having a different `configure` made from each '`configure.in`' by a preprocessor. That approach also offered more control and flexibility.

I looked briefly into using the Metaconfig package, by Larry Wall, Harlan Stenn, and Raphael Manfredi, but I decided not to for several reasons. The `Configure` scripts it produces are interactive, which I find quite inconvenient; I didn't like the ways it checked for some features (such as library functions); I didn't know that it was still being maintained, and the `Configure` scripts I had seen didn't work on many modern systems (such as System V R4 and NeXT); it wasn't very flexible in what it could do in response to a feature's presence or absence; I found it confusing to learn; and it was too big and complex for my needs (I didn't realize then how much Autoconf would eventually have to grow).

I considered using Perl to generate my style of `configure` scripts, but decided that `m4` was better suited to the job of simple textual substitutions: it gets in the way less, because output is implicit. Plus, everyone already has it. (Initially I didn't rely on the GNU extensions to `m4`.) Also, some of my friends at the University of Maryland had recently been putting `m4` front ends on several programs, including `tvtwm`, and I was interested in trying out a new language.

## 14.3 Leviticus

Since my `configure` scripts determine the system's capabilities automatically, with no interactive user intervention, I decided to call the program that generates them Autoconfig. But with a version number tacked on, that name would be too long for old UNIX file systems, so I shortened it to Autoconf.

In the fall of 1991 I called together a group of fellow questers after the Holy Grail of portability (er, that is, alpha testers) to give me feedback as I encapsulated pieces of my handwritten scripts in `m4` macros and continued to add features and improve the techniques used in the checks. Prominent among the testers were François Pinard, who came up with the idea of making an '`autoconf`' shell script to run `m4` and check for unresolved macro calls; Richard Pixley, who suggested running the compiler instead of searching the file system to find include files and symbols, for more accurate results; Karl Berry, who got Autoconf to configure TEX and added the macro index to the documentation; and Ian Taylor, who added support for creating a C header file as an alternative to putting '`-D`' options in a '`Makefile`', so he could use Autoconf for his UUCP package. The alpha testers cheerfully adjusted their files again and again as the names and calling conventions of the Autoconf macros changed from release to release. They all contributed many specific checks, great ideas, and bug fixes.

## 14.4 Numbers

In July 1992, after months of alpha testing, I released Autoconf 1.0, and converted many GNU packages to use it. I was surprised by how positive the reaction to it was. More people started using it than I could keep track of, including people working on software that wasn't part of the GNU Project (such as TCL, FSP, and Kerberos V5). Autoconf continued to improve rapidly, as many people using the `configure` scripts reported problems they encountered.

Autoconf turned out to be a good torture test for `m4` implementations. UNIX `m4` started to dump core because of the length of the macros that Autoconf defined, and several bugs showed up in GNU `m4` as well. Eventually, we realized that we needed to use some features that only GNU `m4` has. 4.3BSD `m4`, in particular, has an impoverished set of builtin macros; the System V version is better, but still doesn't provide everything we need.

More development occurred as people put Autoconf under more stresses (and to uses I hadn't anticipated). Karl Berry added checks for X11. david zuhn contributed `C++` support. François Pinard made it diagnose invalid arguments. Jim Blandy bravely coerced it into configuring GNU Emacs, laying the groundwork for several later improvements. Roland McGrath got it to configure the GNU C Library, wrote the `autoheader` script to automate the creation of C header file templates, and added a '`--verbose`' option to `configure`. Noah Friedman added the '`--macrodir`' option and `AC_MACRODIR` environment variable. (He also coined the term *autoconfiscate* to mean "adapt a software package to use Autoconf".) Roland and Noah improved the quoting protection in `AC_DEFINE` and fixed many bugs, especially when I got sick of dealing with portability problems from February through June, 1993.

## 14.5 Deuteronomy

A long wish list for major features had accumulated, and the effect of several years of patching by various people had left some residual cruft. In April 1994, while working for Cygnus Support, I began a major revision of Autoconf. I added most of the features of the Cygnus `configure` that Autoconf had lacked, largely by adapting the relevant parts of Cygnus `configure` with the help of david zuhn and Ken Raeburn. These features include support for using '`config.sub`', '`config.guess`', '`--host`', and '`--target`'; making links to files; and running `configure` scripts in subdirectories. Adding these features enabled Ken to convert GNU `as`, and Rob Savoye to convert DejaGNU, to using Autoconf.

I added more features in response to other peoples' requests. Many people had asked for `configure` scripts to share the results of the checks between runs, because (particularly when configuring a large source tree, like Cygnus does) they were frustratingly slow. Mike Haertel suggested adding site-specific initialization scripts. People distributing software that had to unpack on MS-DOS asked for a way to override the '`.in`' extension on the file names, which produced file names like '`config.h.in`' containing two dots. Jim Avera did an extensive examination of the problems with quoting in `AC_DEFINE` and `AC_SUBST`; his insights led to significant improvements. Richard Stallman asked that compiler output be sent to '`config.log`' instead of '`/dev/null`', to help people debug the Emacs `configure` script.

I made some other changes because of my dissatisfaction with the quality of the program. I made the messages showing results of the checks less ambiguous, always printing a result. I regularized the names of the macros and cleaned up coding style inconsistencies. I added some auxiliary utilities that I had developed to help convert source code packages to use Autoconf. With the help of François Pinard, I made the macros not interrupt each others' messages. (That feature revealed some performance bottlenecks in GNU `m4`, which he hastily corrected!) I reorganized the documentation around problems people want to solve. And I began a testsuite, because experience had shown that Autoconf has a pronounced tendency to regress when we change it.

Again, several alpha testers gave invaluable feedback, especially François Pinard, Jim Meyering, Karl Berry, Rob Savoye, Ken Raeburn, and Mark Eichin.

Finally, version 2.0 was ready. And there was much rejoicing. (And I have free time again. I think. Yeah, right.)

# 15 Old Macro Names

In version 2 of Autoconf, most of the macros were renamed to use a more uniform and descriptive naming scheme. Here are the old names of the macros that were renamed, followed by the current names of those macros. Although the old names are still accepted by the `autoconf` program for backward compatibility, the old names are considered obsolete. See Section 7.2 [Macro Names], page 62, for a description of the new naming scheme.

`AC_ALLOCA`
: `AC_FUNC_ALLOCA`

`AC_ARG_ARRAY`
: removed because of limited usefulness

`AC_CHAR_UNSIGNED`
: `AC_C_CHAR_UNSIGNED`

`AC_CONST`
: `AC_C_CONST`

`AC_CROSS_CHECK`
: `AC_C_CROSS`

`AC_ERROR`
: `AC_MSG_ERROR`

`AC_FIND_X`
: `AC_PATH_X`

`AC_FIND_XTRA`
: `AC_PATH_XTRA`

`AC_FUNC_CHECK`
: `AC_CHECK_FUNC`

`AC_GCC_TRADITIONAL`
: `AC_PROG_GCC_TRADITIONAL`

`AC_GETGROUPS_T`
: `AC_TYPE_GETGROUPS`

`AC_GETLOADAVG`
: `AC_FUNC_GETLOADAVG`

`AC_HAVE_FUNCS`
: `AC_CHECK_FUNCS`

`AC_HAVE_HEADERS`
: `AC_CHECK_HEADERS`

`AC_HAVE_POUNDBANG`
: `AC_SYS_INTERPRETER` (different calling convention)

`AC_HEADER_CHECK`
: `AC_CHECK_HEADER`

`AC_HEADER_EGREP`
: `AC_EGREP_HEADER`

`AC_INLINE`
: `AC_C_INLINE`

`AC_LN_S`
: `AC_PROG_LN_S`

`AC_LONG_DOUBLE`
: `AC_C_LONG_DOUBLE`

`AC_LONG_FILE_NAMES`
: `AC_SYS_LONG_FILE_NAMES`

`AC_MAJOR_HEADER`
: `AC_HEADER_MAJOR`

`AC_MINUS_C_MINUS_O`
: `AC_PROG_CC_C_O`

`AC_MMAP`
: `AC_FUNC_MMAP`

`AC_MODE_T`
: `AC_TYPE_MODE_T`

`AC_OFF_T`
: `AC_TYPE_OFF_T`

`AC_PID_T`
: `AC_TYPE_PID_T`

`AC_PREFIX`
: `AC_PREFIX_PROGRAM`

`AC_PROGRAMS_CHECK`
: `AC_CHECK_PROGS`

`AC_PROGRAMS_PATH`
: `AC_PATH_PROGS`

`AC_PROGRAM_CHECK`
: `AC_CHECK_PROG`

`AC_PROGRAM_EGREP`
: `AC_EGREP_CPP`

`AC_PROGRAM_PATH`
: `AC_PATH_PROG`

`AC_REMOTE_TAPE`
: removed because of limited usefulness

`AC_RESTARTABLE_SYSCALLS`
: `AC_SYS_RESTARTABLE_SYSCALLS`

`AC_RETSIGTYPE`
: `AC_TYPE_SIGNAL`

`AC_RSH`
: removed because of limited usefulness

`AC_SETVBUF_REVERSED`
: `AC_FUNC_SETVBUF_REVERSED`

`AC_SET_MAKE`
: `AC_PROG_MAKE_SET`

`AC_SIZEOF_TYPE`
: `AC_CHECK_SIZEOF`

`AC_SIZE_T`
: `AC_TYPE_SIZE_T`

`AC_STAT_MACROS_BROKEN`
: `AC_HEADER_STAT`

`AC_STDC_HEADERS`
: `AC_HEADER_STDC`

`AC_STRCOLL`
: `AC_FUNC_STRCOLL`

`AC_ST_BLKSIZE`
: `AC_STRUCT_ST_BLKSIZE`

`AC_ST_BLOCKS`
: `AC_STRUCT_ST_BLOCKS`

`AC_ST_RDEV`
: `AC_STRUCT_ST_RDEV`

`AC_SYS_SIGLIST_DECLARED`
: `AC_DECL_SYS_SIGLIST`

`AC_TEST_CPP`
: `AC_TRY_CPP`

`AC_TEST_PROGRAM`
: `AC_TRY_RUN`

`AC_TIMEZONE`
: `AC_STRUCT_TIMEZONE`

`AC_TIME_WITH_SYS_TIME`
: `AC_HEADER_TIME`

`AC_UID_T` `AC_TYPE_UID_T`

`AC_UTIME_NULL`
: `AC_FUNC_UTIME_NULL`

`AC_VFORK` `AC_FUNC_VFORK`

`AC_VPRINTF`
: `AC_FUNC_VPRINTF`

`AC_WAIT3` `AC_FUNC_WAIT3`

`AC_WARN` `AC_MSG_WARN`

`AC_WORDS_BIGENDIAN`
: `AC_C_BIGENDIAN`

`AC_YYTEXT_POINTER`
: `AC_DECL_YYTEXT`

# Environment Variable Index

This is an alphabetical list of the environment variables that Autoconf checks.

## A



## C



## S

# Output Variable Index

This is an alphabetical list of the variables that Autoconf can substitute into files that it creates, typically one or more 'Makefile's. See Section 6.2 [Setting Output Variables], page 56, for more information on how this is done.

## R



## S



## T



## X



## Y

# Preprocessor Symbol Index

This is an alphabetical list of the C preprocessor symbols that the Autoconf macros define. To work with Autoconf, C source code needs to use these names in `#if` directives.

# Macro Index

This is an alphabetical list of the Autoconf macros. To make the list easier to use, the macros are listed without their preceding 'AC_'.

## A



## B



## C



## D



## E



## F

## R



## S



## T



## U



## V



## W



## X



## Y

# Table of Contents