Autoconf

Generating Automatic Configuration Scripts
Edition 1.11, for Autoconf version 1.11
May 1994

by David MacKenzie, Roland McGrath, and Noah Friedman

1 Introduction

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 normally require no manual user intervention when run; they do not even take 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 operating system features that the package can use. After the shell code to recognize and respond to an operating 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 the configuration scripts can be regenerated automatically to take advantage of the updated code.

Larry Wall's Metaconfig package is similar in purpose to Autoconf, but is more general. 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. They should avoid executing test programs, since test programs compiled with a cross-compiler can not be executed on the host system. Also, they shouldn't do anything that tests features of the host system instead of the target system.

Autoconf imposes some restrictions on the names of macros used with `#ifdef` in C programs (see [Preprocessor Symbol Index], page 44).

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; so use a later version of GNU `m4`.

Autoconf does not work well with GNU C library releases before 1.06. The GNU C library contains stubs (which always return an error) for functions that are not available instead of omitting them from the library. As a result, Autoconf scripts are fooled into thinking that those functions are available. This problem does not exist with releases 1.06 and later of the GNU C library, which define C preprocessor macros that the Autoconf macros `AC_FUNC_CHECK` and `AC_REPLACE_FUNCS` test, indicating that certain functions are stubs (see Section 5.2 [General Feature Tests], page 20, for more information on checking for functions).

Autoconf was written by David MacKenzie, with help from François Pinard, Karl Berry, Richard Pixley, Ian Lance Taylor, Roland McGrath, Noah Friedman, and david d zuhn. It was inspired by Brian Fox's automatic configuration system for Bash, by Larry Wall's

Metaconfig, and by Richard Stallman, Richard Pixley, and John Gilmore's configuration tools for the GNU compiler and object file utilities.

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 Distributing Autoconf Output

The configuration scripts that Autoconf produces are covered by the GNU General Public License. This is because they consist almost entirely of parts of Autoconf itself, rearranged somewhat, and Autoconf is distributed under the terms of the GPL. As applied to Autoconf, the GPL just means that you need to distribute `configure.in`, and `aclocal.m4`, `acconfig.h`, and *`config`*`.h.top` and *`config`*`.h.bot` if you use them, along with `configure`.

Programs that use Autoconf scripts to configure themselves do not automatically come under the GPL. Distributing an Autoconf configuration script as part of a program is considered to be *mere aggregation* of that work with the Autoconf script. Such programs are not derivative works based on Autoconf; only their configuration scripts are. 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.

3 Making `configure` Scripts

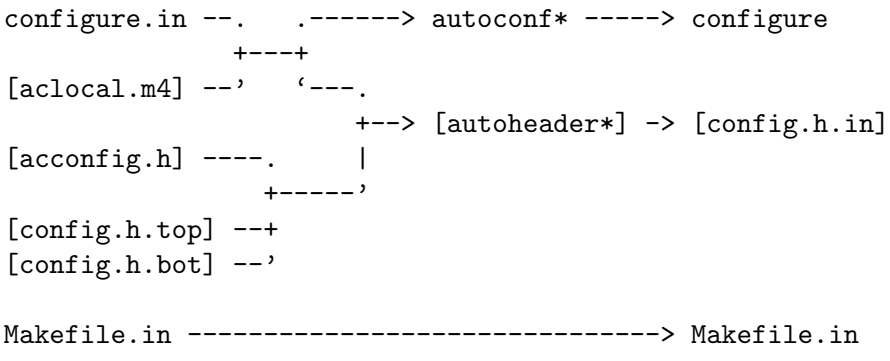
The configuration scripts that Autoconf produces are by convention called `configure`. When run, `configure` creates several files, replacing configuration parameters in them with values appropriate for the system being configured. The files that `configure` creates are:

- one or more `Makefile` files (one in each subdirectory of the package), from template `Makefile.in` files (see Chapter 7 [Makefiles], page 34);
- optionally, a C header file, the name of which is configurable, containing `#define` statements (see Section 5.1 [Setup], page 19);
- a shell script called `config.status` that, when run, will recreate the files listed above (see Section 8.2 [Invoking config.status], page 38).

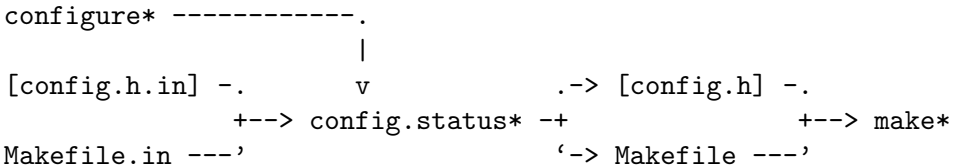
To create a `configure` script with Autoconf, you need to write an Autoconf input file (`configure.in`) and run Autoconf on it to produce the script. If you write your own feature tests to supplement those that come with Autoconf, you might also write a file called `aclocal.m4`. If you use a C header file to contain `#define` directives, you might also write `config.h.top`, `config.h.bot`, and `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 files `acgeneral.m4` and `acspecific.m4`, and also an installed `aclocal.m4` if it exists.

Files used in preparing a software package for distribution:



Files used in configuring a software package:



3.1 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

[Specific Tests], page 7, for their descriptions. For most other features, you can use Autoconf template macros to produce custom checks; see Section 5.2 [General Feature Tests], page 20, for information about them. For especially tricky or specialized features, configure.in might need to contain some hand-crafted shell commands. See Chapter 6 [Writing Macros], page 29, for guidelines on writing tests from scratch.

Every configure.in must begin with a call to AC_INIT and end with a call to AC_OUTPUT (see Section 5.1 [Setup], page 19). Other than that, the order in which configure.in calls the Autoconf macros is generally not important, except that 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 [Specific Tests], page 7).

To encourage consistency, here is a suggested order for calling the Autoconf macros. A few macros need to be called in a different order from the one given here; they are noted in their individual descriptions (see Chapter 4 [Specific Tests], page 7). Note that there must not be any space between the macro name and the open parentheses.

AC_INIT(*file*)
checks for alternative programs
checks for UNIX variants that set C preprocessor variables
checks for header files
checks for typedefs
checks for library functions
checks for structures
checks for compiler characteristics
checks for system services
other checks for UNIX variants
AC_OUTPUT([*file*...])

You can include comments in configure.in files by starting them with the m4 predefined 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.
```

See Section 9.1 [Sample configure.in], page 40, for an example of a real configure.in script.

3.2 Invoking autoconf

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 two or more files. Two of the files are distributed with Autoconf: autoconf first reads acgeneral.m4 (see Chapter 5 [General Purpose Macros], page 19), then acspecific.m4 (see Chapter 4 [Specific Tests], page 7). After reading them, autoconf looks for an optional file called aclocal.m4, first in the directory

that contains other installed Autoconf macro files, and then in the current directory. If both files exist, it uses both of them. Those files can contain your site's own locally written Autoconf macro definitions (see Chapter 6 [Writing Macros], page 29, 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.

You can override the directory where `autoconf` looks for the installed macro files by setting the `AC_MACRODIR` environment variable to a different directory. You can also give `autoconf` the '`--macrodir`' option, which overrides `AC_MACRODIR`.

`autoconf` also accepts the options `--version`, which prints the Autoconf version number and exits, and `--help`, which prints a summary of the command-line options and exits.

3.3 Invoking `autoheader`

You can use the `autoheader` program to create a template file of C '`#define`' statements for `configure` to use. By default, the file that `autoheader` creates is called `config.h.in`; if `configure.in` invokes `AC_CONFIG_HEADER(`*`file`*`)`, `autoheader` creates *`file`*`.in`.

`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; you must create that file to contain entries for any additional symbols that you `AC_DEFINE`. For symbols defined by `AC_HAVE_HEADERS`, `AC_HAVE_FUNCS`, `AC_SIZEOF_TYPE`, or `AC_HAVE_LIBRARY`, `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 usually contains only `#define` and `#undef` statements and their accompanying comments. However, if a file called *`file`*`.top` (typically `config.h.top`) exists in the current directory, `autoheader` copies that file to the beginning of its output.

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.

You can override the directory where `autoheader` looks for the installed macro and `acconfig.h` files by setting the `AC_MACRODIR` environment variable to a different directory. You can also give `autoheader` the '`--macrodir`' option, which overrides `AC_MACRODIR`.

`autoheader` also accepts the options `--version`, which prints the Autoconf version number and exits, and `--help`, which prints a summary of the command-line options and exits.

4 Specific Tests

These macros test for particular operating system features that packages might need or want to use. If you need to test for a feature that none of these macros check for, you can probably do it by calling one of the general purpose test macros with appropriate arguments (see Section 5.2 [General Feature Tests], page 20).

All of these macros that set `make` variables call `AC_SUBST` on those variables (see Section 5.4 [Setting Variables], page 24, for details about `AC_SUBST`). The phrase "define *name*" is used below as a shorthand to mean either add '`-D`*name*`=1`' to the `make` variable `DEFS`, or put '`#define` *name* `1`' in the configuration header file, depending on whether `AC_CONFIG_HEADER` has been called. See Section 5.4 [Setting Variables], page 24, for more information.

Within each section below, the macros are listed in alphabetical order. The macros are generally named for the `make` variables or C preprocessor macros that they define; those names are based largely on what existing GNU programs use. These macros are defined in the file `acspecific.m4`.

4.1 Alternative Programs

The following macros check for the presence or behavior of particular programs:

`AC_GCC_TRADITIONAL` [Macro]
: Add '`-traditional`' to `make` variable `CC` if using the GNU C compiler and `ioctl` does not work properly without '`-traditional`'. This macro calls `AC_PROG_CC` and `AC_PROG_CPP` if they haven't been called already.

`AC_LN_S` [Macro]
: If '`ln -s`' works on the current filesystem (the O.S. and filesystem support symbolic links), set shell and `make` variable `LN_S` to '`ln -s`', otherwise set it to '`ln`'.

`AC_MINUS_C_MINUS_O` [Macro]
: If the C compiler does not accept the '`-c`' and '`-o`' options simultaneously, define `NO_MINUS_C_MINUS_O`.

`AC_PROG_AWK` [Macro]
: Check for `mawk`, `gawk`, `nawk`, and `awk`, in that order, and set `make` variable `AWK` to the first one that it finds.

`AC_PROG_CC` [Macro]
: If `gcc` is found, set `make` variable `CC` to '`gcc`', and set shell variable `GCC` to 1 for use by macros such as `AC_GCC_TRADITIONAL`.

`AC_PROG_CPP` [Macro]
: Set shell and `make` 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 5.6 [Language Choice], page 26), many of the specific test macros use the value of `CPP` indirectly by calling `AC_TEST_CPP`,

`AC_HEADER_CHECK`, `AC_HEADER_EGREP`, or `AC_PROGRAM_EGREP`. Those macros call this macro first if it hasn't been called already. It calls `AC_PROG_CC` if it hasn't been called already.

`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 `make` variable `CXX` to its value. Otherwise search for a C++ compiler under likely names (`c++`, `g++`, `gcc`, and `CC`). If none of those checks succeed, as a last resort set `CXX` to `gcc`.

`AC_PROG_CXXCPP` [Macro]
Set shell and `make` 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` or `.cc` extension.

If the current language is C++ (see Section 5.6 [Language Choice], page 26), many of the specific test macros use the value of `CXXCPP` indirectly by calling `AC_TEST_CPP`, `AC_HEADER_CHECK`, `AC_HEADER_EGREP`, or `AC_PROGRAM_EGREP`. Those macros call this macro first if it hasn't been called already. This macro calls `AC_PROG_CXX` if it hasn't been called already.

`AC_PROG_INSTALL` [Macro]
Set `make` variable `INSTALL` to '`install -c`' if `install` is found and is compatible with the BSD and GNU versions. Otherwise, set `INSTALL` to '*dir*`/install.sh -c`', where it checks for `install.sh` in the directories `$srcdir`, `$srcdir/..`, and `$srcdir/../..` to determine *dir*.

This macro screens out the false matches `/etc/install`, `/usr/sbin/install`, and other instances of `install` known not to work. It also sets the variable `INSTALL_PROGRAM` to '`${INSTALL}`' and `INSTALL_DATA` to '`${INSTALL} -m 644`'.

If you need to use your own `install.sh` 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 script into your `Makefile.in` files.

`AC_PROG_LEX` [Macro]
If `flex` is found, set `make` 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_RANLIB` [Macro]
Set `make` variable `RANLIB` to '`ranlib`' if `ranlib` is found, otherwise to '`:`' (do nothing).

`AC_PROG_YACC` [Macro]
If `bison` is found, set `make` variable `YACC` to '`bison -y`'. Otherwise, if `byacc` is found, set `YACC` to '`byacc`'. Otherwise set `YACC` to '`yacc`'.

`AC_RSH` [Macro]
If a remote shell is available, put '`rtapelib.o`' in `make` variable `RTAPELIB`. Otherwise, also do so if `netdb.h` exists (implying the `rexec` function), and in addition define `HAVE_NETDB_H`. If neither a remote shell nor `rexec` is available, define `NO_REMOTE`.

`AC_SET_MAKE` [Macro]

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

In recent versions of `make`, the variable `MAKE` contains the name of the `make` program plus options it was given. It is used when running `make` recursively in subdirectories. But some old versions of `make` don't set the `MAKE` variable. This macro allows use of `MAKE` on all systems.

If you use this macro, simply place a line like this in your `Makefile.in` file(s):

```
@SET_MAKE@
```

`AC_YYTEXT_POINTER` [Macro]

Define `YYTEXT_POINTER` if `yytext` is a '`char *`' instead of a '`char []`'. This depends on whether `lex` or `flex` is being used. This macro calls `AC_PROG_CPP` (or `AC_PROG_CXXCPP` if C++ is the current language, see Section 5.6 [Language Choice], page 26) and `AC_PROG_LEX` if they haven't been called already.

This macro replaces `AC_DECLARE_YYTEXT`, which didn't work.

4.2 Header Files

The following macros check for the presence of certain C header files:

`AC_DIR_HEADER` [Macro]

If the system has `dirent.h`, define `DIRENT`; otherwise, if it has `sys/ndir.h`, define `SYSNDIR`; otherwise, if it has `sys/dir.h`, define `SYSDIR`; otherwise, if it has `ndir.h`, define `NDIR`. Also, if the directory library header file contains a declaration of the `closedir` function with a `void` return type, define `VOID_CLOSEDIR`.

The directory library declarations in the source code should look something like the following, which assumes that you have also called '`AC_HAVE_HEADERS(unistd.h)`':

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

/* unistd.h defines _POSIX_VERSION on POSIX.1 systems.  */
#if defined(DIRENT) || defined(_POSIX_VERSION)
#include <dirent.h>
#define NLENGTH(dirent) (strlen((dirent)->d_name))
#else /* not (DIRENT or _POSIX_VERSION) */
#define dirent direct
#define NLENGTH(dirent) ((dirent)->d_namlen)
#ifdef SYSNDIR
#include <sys/ndir.h>
#endif /* SYSNDIR */
#ifdef SYSDIR
#include <sys/dir.h>
#endif /* SYSDIR */
#ifdef NDIR
#include <ndir.h>
#endif /* NDIR */
#endif /* not (DIRENT or _POSIX_VERSION) */
```

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 `NLENGTH` macro.

`AC_MAJOR_HEADER` [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_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_HAVE_HEADERS(memory.h)`. See the example for `AC_STDC_HEADERS`.

`AC_STDC_HEADERS` [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.

To check whether to use the System V/ANSI C string functions and header file, you can put the following in `configure.in`:

```
AC_STDC_HEADERS
AC_HAVE_HEADERS(string.h memory.h)
```

Then, in the code, use a test like this:

```
#if STDC_HEADERS || HAVE_STRING_H
#include <string.h>
/* An ANSI string.h and pre-ANSI memory.h might conflict.  */
#if !STDC_HEADERS && HAVE_MEMORY_H
#include <memory.h>
#endif /* not STDC_HEADERS and HAVE_MEMORY_H */
#define index strchr
#define rindex strrchr
#define bcopy(s, d, n) memcpy ((d), (s), (n))
#define bcmp(s1, s2, n) memcmp ((s1), (s2), (n))
#define bzero(s, n) memset ((s), 0, (n))
#else /* not STDC_HEADERS and not HAVE_STRING_H */
#include <strings.h>
/* memory.h and strings.h conflict on some systems.  */
#endif /* not STDC_HEADERS and not HAVE_STRING_H */
```

This example asssumes that your code uses the BSD style functions. If you use the System V/ANSI C style functions, you will need to replace the macro definitions with ones that go in the other direction.

This macro calls `AC_PROG_CPP` or `AC_PROG_CXXCPP` (depending on which language is current, see Section 5.6 [Language Choice], page 26), if it hasn't been called already.

`AC_UNISTD_H` [Macro]

Define `HAVE_UNISTD_H` if the system has `unistd.h`. This macro is obsolete; instead, use '`AC_HAVE_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, use `AC_HAVE_HEADERS(string.h)` and use `HAVE_STRING_H` in your code. See the example for `AC_STDC_HEADERS`.

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

4.3 Typedefs

The following macros check for predefined C types:

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

`AC_MODE_T` [Macro]
If `mode_t` is not defined in `sys/types.h`, define `mode_t` to be `int`.

`AC_OFF_T` [Macro]
If `off_t` is not defined in `sys/types.h`, define `off_t` to be `long`.

`AC_PID_T` [Macro]
If `pid_t` is not defined in `sys/types.h`, define `pid_t` to be `int`.

`AC_RETSIGTYPE` [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_SIZE_T` [Macro]
If `size_t` is not defined in `sys/types.h`, define `size_t` to be `unsigned`.

`AC_UID_T` [Macro]
If `uid_t` is not defined in `sys/types.h`, define `uid_t` to be `int` and `gid_t` to be `int`.

4.4 Library Functions

The following macros check for particular C library functions:

`AC_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 that fails, it looks for a function in the standard C library. If that fails, it sets the `make` 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.

If this macro finds alloca.h, it defines HAVE_ALLOCA_H.

This macro does not try to get alloca from the SVR3 libPW or the SVR4 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. Note that 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 /* not __GNUC__ */
#if HAVE_ALLOCA_H
#include <alloca.h>
#else /* not HAVE_ALLOCA_H */
#ifdef _AIX
 #pragma alloca
#else /* not _AIX */
char *alloca ();
#endif /* not _AIX */
#endif /* not HAVE_ALLOCA_H */
#endif /* not __GNUC__ */
```

AC_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 make variable LIBOBJS, and possibly defines several other C preprocessor macros and make 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 always defines NEED_SETGID, for make. The value is 'true' if special installation is required, 'false' if not. If NEED_SETGID is 'true', it sets KMEM_GROUP to the name of the group that should own the installed program.

AC_MMAP [Macro]

If the mmap function exists and works correctly, define HAVE_MMAP.

`AC_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_STRCOLL` [Macro]
: If the `strcoll` function exists and works correctly, define `HAVE_STRCOLL`. This does a bit more than '`AC_HAVE_FUNCS(strcoll)`', because some systems have incorrect definitions of `strcoll`, which should not be used.

`AC_UTIME_NULL` [Macro]
: If '`utime(`*file*`, NULL)`' sets *file*'s timestamp to the present, define `HAVE_UTIME_NULL`.

`AC_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.

`AC_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_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`.

4.5 Structures

The following macros check for certain structures or structure members:

`AC_STAT_MACROS_BROKEN` [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_ST_BLKSIZE` [Macro]
: If `struct stat` contains an `st_blksize` member, define `HAVE_ST_BLKSIZE`.

`AC_ST_BLOCKS` [Macro]
: If `struct stat` contains an `st_blocks` member, define `HAVE_ST_BLOCKS`. Otherwise, add '`fileblocks.o`' to the `make` variable `LIBOBJS`.

`AC_ST_RDEV` [Macro]
: If `struct stat` contains an `st_rdev` member, define `HAVE_ST_RDEV`.

`AC_TIME_WITH_SYS_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`.

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

`AC_STRUCT_TM` [Macro]

If `time.h` does not define `struct tm`, define `TM_IN_SYS_TIME`, which means that including `sys/time.h` defines `struct tm`.

`AC_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`. This macro calls `AC_STRUCT_TM` if it hasn't been called already.

4.6 Compiler Characteristics

The following macros check for C compiler or machine architecture features:

`AC_ARG_ARRAY` [Macro]

If the address of an argument to a C function can not be used like the start of an array, define `NO_ARG_ARRAY`. This ability allows a sequence of arguments with the same type to be accessed as if they were an array of values.

`AC_CROSS_CHECK` [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 1. This information can be used by `AC_TEST_PROGRAM` to determine whether to take a default action instead of trying to run a test program (see Section 5.2 [General Feature Tests], page 20).

`AC_CHAR_UNSIGNED` [Macro]

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

`AC_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. (If using a configuration header file, the program should include it before any other header files, to prevent inconsistencies in declarations.)

`AC_INLINE` [Macro]

If the C compiler is a version of GCC that supports the keyword `__inline` but not `inline` (such as some NeXT versions), define `inline` to be `__inline`. This macro calls `AC_PROG_CC` if it hasn't been called already.

`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_SIZEOF_TYPE(int)`' instead (see Section 5.2 [General Feature Tests], page 20).

`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_SIZEOF_TYPE(long)`' instead (see Section 5.2 [General Feature Tests], page 20).

`AC_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_WORDS_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`.

4.7 System Services

The following macros check for operating system services:

`AC_FIND_X` [Macro]

Try to locate the X Window System include files and libraries. Try first 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 '`true`'; otherwise set it to the empty string.

The command line options '`--x-includes=`*dir*' and '`--x-libraries=`*dir*' override the values chosen by this macro.

`AC_FIND_XTRA` [Macro]

An enhanced version of `AC_FIND_X`. Put the C compiler flags that X needs into `make` variable `X_CFLAGS`, and the X linker flags into `X_LIBS`. If X is not available, put '`-DX_DISPLAY_MISSING`' into `X_CFLAGS`.

Also check for special libraries that some systems need in order to compile X programs. Add any that the system needs to `make` variable `X_EXTRA_LIBS`. This macro calls `AC_FIND_X` and `AC_ISC_POSIX` (see Section 4.8 [UNIX Variants], page 17) if they have not already been called. Because of the macro dependencies, if you call this macro, you should let it call `AC_FIND_X` rather than doing that yourself.

`AC_HAVE_POUNDBANG` (*action-if-supported* [, *action-if-not-supported*]) [Macro]

Check whether the system supports starting shell scripts with a line of the form '`#!/bin/csh`' to select the shell to use. If '`#!`' works, execute shell commands *action-if-supported*; if not, execute *action-if-not-supported*.

`AC_LONG_FILE_NAMES` [Macro]

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

`AC_REMOTE_TAPE` [Macro]

If BSD tape drive ioctls are available, define `HAVE_SYS_MTIO_H`, and if sockets are available add `rmt` to `make` variable `PROGS`.

`AC_RESTARTABLE_SYSCALLS` [Macro]

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

4.8 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:

`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 `make` variable `LIBS`. Allows use of some BSD system calls and `getmntent`.

`AC_IRIX_SUN` [Macro]

If on IRIX (Silicon Graphics UNIX), add '`-lsun`' to `make` variable `LIBS`. Needed to get `getmntent`. At sites using Yellow Pages/NIS, it is also needed to get properly working `gethostby*`, `getpw*`, `getgr*`, `getnetby*`, and so on.

`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 `make` 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 `make` variable `LIBS`. Used to get `strftime`. It must be called before checking for `strftime`.

`AC_XENIX_DIR` [Macro]

If on Xenix, define `VOID_CLOSEDIR` and add '`-lx`' to `make` variable `LIBS`. Also, if `sys/ndir.h` is not being used, add '`-ldir`' to `LIBS`. Needed when using the directory reading functions. This macro must be called after `AC_DIR_HEADER`.

5 General Purpose Macros

These macros provide ways for other macros to control the kind of output that Autoconf produces or to check whether various kinds of features are available. They all take arguments. When calling these macros, there must not be any blank space between the macro name and the open parentheses.

Arguments to these macros can be more than one line long if they are enclosed within the `m4` quote characters '`[`' and '`]`'.

Within each section below, the macros are listed in alphabetical order. These macros are defined in the file `acgeneral.m4`.

5.1 Controlling Autoconf Setup

The following macros control the kind of output that Autoconf produces.

`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`. This macro should be called right after `AC_INIT`. 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.

Your distribution should contain a file *`header-to-create`*`.in` that looks as you want the final header file to look, including comments, with default values in the `#define` statements. A default value can be to `#undef` the variable instead of to define it to a value, if your code tests for configuration options using `#ifdef` instead of `#if`.

You can use the program `autoheader` to create *`header-to-create`*`.in` (see Section 3.3 [Invoking autoheader], page 6).

`AC_INIT` (*`unique-file-in-source-dir`*) [Macro]

Process the 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 (see Chapter 8 [Invoking configure], page 37, for more information).

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

Create output files (typically one or more `Makefile`s) and `config.status`. If `AC_CONFIG_HEADER` has been called, also create the header file that was named as its argument. The argument is a whitespace-separated list of files to create; if it is omitted, no files are created. `AC_OUTPUT` creates each file *`file`* in the list by copying *`file`*`.in`, substituting the variable values that have been selected by calling `AC_SUBST`. It creates the directory that each file is in if it doesn't exist (but not the parents of that directory). A plausible value for the argument to `AC_OUTPUT` is '`Makefile src/Makefile man/Makefile X/Imakefile`'.

If you pass *extra-cmds*, those commands will be inserted into `config.status` to be run after all its other processing.

`AC_PREPARE` (*unique-file-in-source-dir*) [Macro]

Find the source code directory and set up shell variables necessary for other Autoconf macros to work. *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 (see Chapter 8 [Invoking configure], page 37, for more information). `AC_PREPARE` is the last thing done by `AC_INIT`. Use `AC_PREPARE` instead of `AC_INIT` if you want to do argument parsing yourself; never use both.

`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* (e.g., '`1.8`'), print an error message on the standard error output and do not create `configure`.

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
```

5.2 Checking for Kinds of Features

These macros are templates that, when called with actual parameters, check for various kinds of features. Many of these 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 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.

One shell programming construction that you should not use in the action arguments to these macros is '*var*`=${`*var*`:-`*value*`}`'. Old BSD shells, including the Ultrix `sh`,

don't understand the colon, and complain and die. If you omit the colon, it works fine: '*var*=${*var-value*}'. Using the form without the colon has one small disadvantage. Users can not select a default value by giving a variable an empty value, e.g., 'CC= configure'. Instead, they must unset the variable, e.g., 'unset CC; configure'.

See Chapter 6 [Writing Macros], page 29, for more information on how best to use these macros.

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

Print 'checking for *echo-text*' to the standard output (using AC_CHECKING, see Section 5.5 [Printing Messages], page 26). 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*.

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

If *function* is available, run shell commands *action-if-found*, otherwise *action-if-not-found*. If the functions might be in libraries other than the default C library, first call AC_HAVE_LIBRARY for those libraries. If you just want to define a symbol if the function is available, consider using AC_HAVE_FUNCS instead.

AC_HAVE_FUNCS (*function...*) [Macro]

For each given *function* in the whitespace-separated argument list that is available, define HAVE_*function* (in all caps). See Chapter 4 [Specific Tests], page 7, for a precise definition of "define" as it is used here. If the functions might be in libraries other than the default C library, first call AC_HAVE_LIBRARY for those libraries.

AC_HAVE_HEADERS (*header-file...*) [Macro]

For each given *header-file* in the whitespace-separated argument list that exists, define HAVE_*header-file* (in all caps). See Chapter 4 [Specific Tests], page 7, for a precise definition of "define" as it is used here.

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

Create a test C program to see whether that program can be linked with the specified library. *action-if-found* is a list of shell commands to run if the link succeeds (which means that the library is present); *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 '-lfoo' to LIBS and define 'HAVE_LIBfoo' for library 'foo'. *library* can be written as any of 'foo', '-lfoo', or 'libfoo.a'. In all of those cases, the compiler is passed '-lfoo'.

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

If *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_HAVE_HEADERS instead.

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

If the output of running the preprocessor on *header-file* contains the `egrep` regular expression *pattern*, execute shell commands *action-if-found*, otherwise execute

This macro calls `AC_PROG_CPP` or `AC_PROG_CXXCPP` (depending on which language is current, see Section 5.6 [Language Choice], page 26), if it hasn't been called already. *action-if-not-found*.

You can not check whether preprocessor symbols are defined this way, because they get expanded before `egrep` sees them. But you can almost always detect them by simply using `#ifdef` directives in your programs.

`AC_PROGRAM_CHECK` (*variable*, *prog-to-check-for*, *value-if-found*, *value-if-not-found*) [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 *variable* was already set, do nothing. Calls `AC_SUBST` for *variable*.

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

program is the text of a C or C++ program, on which shell variable and backquote substitutions are performed. If the output of running the preprocessor on *program* contains the `egrep` regular expression *pattern*, execute shell commands *action-if-found*, otherwise execute *action-if-not-found*. (It is an unfortunate oversight that we use the word `PROGRAM` in Autoconf macro names to sometimes mean C or C++ source code and sometimes mean a UNIX command.)

This macro calls `AC_PROG_CPP` or `AC_PROG_CXXCPP` (depending on which language is current, see Section 5.6 [Language Choice], page 26), if it hasn't been called already.

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

Similar to `AC_PROGRAM_CHECK`, but set *variable* to the entire path of *prog-to-check-for* if found. Otherwise, set *variable* to the value *value-if-not-found* and perform no path checking. If *variable* was already set, do nothing. Calls `AC_SUBST` for *variable*.

`AC_PROGRAMS_CHECK` (*variable*, *progs-to-check-for* [, *value-if-not-found*]) [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* will not be changed. Calls `AC_SUBST` for *variable*.

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

Like `AC_PROGRAMS_CHECK`, but if any of *progs-to-check-for* are found, set *variable* to the entire pathname of the program 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 `make` variable `LIBOBJS`. If the functions might be in libraries other than the default C library, first call `AC_HAVE_LIBRARY` for those libraries.

`AC_SIZEOF_TYPE (`*type*`)` [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, gets a size of 0. *uctype* is *type*, with lowercase converted to uppercase, spaces changed to underscores, and asterisks changed to '`P`'. For example, the call

```
AC_SIZEOF_TYPE(int *)
```

defines `SIZEOF_INT_P` to be 64 on DEC Alpha AXP systems.

`AC_TEST_PROGRAM (`*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*.

If the optional argument *action-if-cross-compiling* is given and 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. Instead, the shell commands *action-if-cross-compiling* are run. If that argument is given, this macro calls `AC_CROSS_CHECK` if it has not already been called (see Section 4.6 [Compiler Characteristics], page 15).

`AC_TEST_CPP (`*includes*, *action-if-true* [, *action-if-false*]`)` [Macro]
includes is C or C++ `#include` statements and declarations, on which shell variable and backquote 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 calls `AC_PROG_CPP` or `AC_PROG_CXXCPP` (depending on which language is current, see Section 5.6 [Language Choice], page 26), if it hasn't been called already.

5.3 Checking Command Line Arguments

These macros check whether the user gave `configure` various command line arguments. Like the general feature tests (see Section 5.2 [General Feature Tests], page 20), they may take an argument to use if the argument was given and one for if it was not given.

`AC_ENABLE (`*feature*, *action-if-true* [, *action-if-false*]`)` [Macro]
If the user gave `configure` the option '`--enable-`*feature*' or '`--disable-`*feature*', run shell commands *action-if-true*. Otherwise run shell commands *action-if-false*. The name *feature* should consist only of alphanumeric characters and dashes.

The *feature* indicates an optional user-level facility. This option allows 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`' indicates that the feature is *not* available. A feature with an argument looks like '`--enable-debug=stabs`'.

The argument is available to the shell commands *action-if-true* in the shell variable `enableval`. If no argument was given to '`--enable-`*feature*', `enableval` is '`yes`'. '`--disable-`*feature*' is equivalent to '`--enable-`*feature*`=no`'. At present, arguments containing blanks are not handled correctly; if you need an argument to contain a list, require the items to be separated by commas instead. (This restriction might disappear in the future.)

`AC_PREFIX` (*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`.

`AC_WITH` (*package*, *action-if-true* [, *action-if-false*]) [Macro]

If the user gave `configure` the option '`--with-`*package*' or '`--without-`*package*', run shell commands *action-if-true*. Otherwise run shell commands *action-if-false*. The name *package* should consist only of alphanumeric characters and dashes.

The *package* indicates another software package that this program should work with. For example, '`--with-gnu-ld`' means work with the GNU linker instead of some other linker. '`--with-x11`' means work with X11.

The user can give an argument by following the package name with '`=`' and the argument. Giving an argument of '`no`' is for packages that would be 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.

The argument is available to the shell commands *action-if-true* in the shell variable `withval`. If no argument was given to '`--with-`*package*', `withval` is '`yes`'. '`--without-`*package*' is equivalent to '`--with-`*package*`=no`'. At present, arguments containing blanks are not handled correctly; if you need an argument to contain a list, require the items to be separated by commas instead. (This restriction might disappear in the future.)

5.4 Setting Variables

These macros help other macros to define shell and `make` variables.

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

Define C preprocessor variable *variable*. If *value* is given, set *variable* to that value, otherwise set it to 1. To use a shell variable as the value, use `AC_DEFINE_UNQUOTED` instead and precede double quotes in the value with backslashes.

This macro adds to the shell variable `DEFS`. `AC_OUTPUT` later substitutes the values in `DEFS` into the file(s) that it generates (typically `Makefile`). Alternately, if `AC_CONFIG_HEADER` has been called, `AC_OUTPUT` creates a header file by substituting the correct values into `#define` statements in a template file.

For example, suppose your `configure.in` calls `AC_CONFIG_HEADER(conf.h)` and `AC_HAVE_HEADERS(unistd.h)`. You could have code like this in `conf.h.in`:

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

On systems that have `unistd.h`, `configure` will change the 0 to a 1. On other systems, it will leave the line unchanged. Alternately, if you prefer to use `#ifdef`, your `conf.h.in` could have code like this:

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

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).

Due to the syntactical bizarreness of the Bourne shell, do not use semicolons to separate `AC_DEFINE` 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_HEADER_CHECK(elf.h, AC_DEFINE(SVR4) LIBS="$LIBS -lelf")
```

or this:

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

instead of this:

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

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

Like `AC_DEFINE`, but it does nothing to quote *value* from various shell and `sed` expansions it will undergo. *value* will be used in many different contexts requiring different quoting, and it is up to you to make sure it works right. Use this macro instead of `AC_DEFINE` when *value* contains a shell variable. For example:

```
AC_DEFINE_UNQUOTED(config_machfile,  ${machfile})
```

`AC_SUBST` (*variable*) [Macro]

Substitute the variable *variable* when creating the output files (typically one or more `Makefile`s). This means replace instances of '`@`*variable*`@`', e.g. in `Makefile.in`, with the current value of the shell variable *variable*. If this macro were not called, the value of *variable* would not be set in the output files, even though `configure` had figured out a value for it.

You can set or add to the value of *variable* in the usual shell way. For example, to add '`-ltermcap`' to the value of the variable `LIBS`:

```
LIBS="$LIBS -ltermcap"
```

5.5 Printing Messages

`configure` scripts need to give users running them several kinds of information. The following macros print messages in ways appropriate for different kinds of information. 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. Other macros should rarely need to run `echo` directly to print messages for the `configure` 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_CHECKING` (*feature-description*) [Macro]
: Notify the user that `configure` is checking for a particular feature. This macro prints a message that starts with '`checking `'. It prints nothing if `configure` is run with the '`--silent`' or '`--quiet`' option. The *feature-description* should be something like '`whether the Fortran compiler accepts C++ comments`' or '`for c89`'.

`AC_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 stream and exits `configure` with a nonzero status. *error-description* should be something like '`invalid value $HOME for \$HOME`'.

`AC_VERBOSE` (*result-description*) [Macro]
: Notify the user of the results of a check. This information is only printed if `configure` is run with the '`--verbose`' option. *result-description* should be something like '`setting ADA to $ADA`'.

`AC_WARN` (*problem-description*) [Macro]
: Notify the `configure` user of a possible problem. This macro prints the message on the standard error stream; `configure` continues running afterward, so macros that call `AC_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`'.

5.6 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.

  This is the initial state.

`AC_LANG_CPLUSPLUS` [Macro]
: Do compilation tests using `CXX` and `CXXCPP` and use extension `.C` for test programs.

`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_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_REQUIRE_CPP` [Macro]

Ensure that whichever preprocessor would currently be used for tests has been found. Calls `AC_REQUIRE` (see Section 5.7 [Macro Ordering], page 27) with an argument of either `AC_PROG_CPP` or `AC_PROG_CXXCPP`, depending on which language is current.

5.7 Macro Ordering

These macros provide ways for other macros to make sure that they are called in the correct order.

`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 contain a call to `AC_PROVIDE` to indicate that it has been called.

This macro should be used when one macro makes changes that might affect another macro, so that the other macro should probably not be called first. 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])
```

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

`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_BEFORE`. 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*.

A sample call is:

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

`AC_PROVIDE` (*macro-name*) [Macro]

Set a flag recording that *macro-name* has been called. The argument 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])
```

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. The body of *macro-name* must contain a call to `AC_PROVIDE` to indicate that it has been called.

Macros that need some other macro to be called before they are called can use `AC_REQUIRE` to ensure that it has been, in case the person who made `configure.in` forgot or didn't know to do it. `AC_REQUIRE` and `AC_PROVIDE` together can ensure that a macro is only called if it is needed, and only called once. See Section 6.3 [Dependencies Between Macros], page 30, for more information.

6 Writing Macros

If your package needs to test for some feature that none of the macros supplied with Autoconf handles, you'll need to write one or more new Autoconf macros. Here are some suggestions and some of the rationale behind why the existing macros are written the way they are. You can also learn a lot about how to write Autoconf macros by looking at the existing ones. If something goes wrong in one or more of the Autoconf tests, this information can help you understand why they work the way they do and the assumptions behind them, which might help you figure out how to best solve the problem.

If you add macros that you think would be useful to other people, or find problems with the distributed macros, please send electronic mail to `bug-gnu-utils@prep.ai.mit.edu`, so we can consider them for future releases of Autoconf. Please include the Autoconf version number, which you can get by running '`autoconf --version`'.

6.1 Macro Format

Autoconf macros are defined as arguments to the `m4` builtin command `define`. Their overall structure looks like this:

```
define(macro-name, [macro-body])dnl
```

The square brackets here do not indicate optional text: they should literally be present in the macro definition.

All of the Autoconf macros have names starting with '`AC_`' to prevent them from accidentally conflicting with other text. All shell variables that they use for internal purposes have 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.

The `m4` builtin `dnl` prevents a newline from being inserted in the output where the macro is defined; without it, the generated `configure` script would begin with dozens of blank lines. `dnl` is also used to introduce comments in `m4`; it causes `m4` to discard the rest of the input line.

You should quote the entire macro body with square brackets to avoid macro expansion problems (see Section 6.2 [Quoting], page 29). You can refer to any arguments passed to the macro as '`$1`', '`$2`', etc.

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

6.2 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.

Autoconf (in `acgeneral.m4`) 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. In those places, they use the `m4` builtin command `changequote` to temporarily disable quoting before the code that uses brackets, like this:

```
changequote(,)dnl
```

Then they turn quoting back on again with another call to `changequote`:

```
changequote([,])dnl
```

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`'.

6.3 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.

6.3.1 Prerequisite Macros

A macro that you write might need to use values that have previously been computed by other macros. For example, if you write a new macro that uses the C preprocessor, it depends on `AC_PROG_CPP` having been called first to set the shell variable `CPP` (see Section 4.1 [Alternative Programs], page 7).

Rather than forcing the user of the macros to keep track of all of the dependencies between them, you can use the macros `AC_PROVIDE` and `AC_REQUIRE` to do it automatically. See Section 5.7 [Macro Ordering], page 27, for more information on their syntax.

The new macro that runs the C preprocessor should contain, somewhere before `CPP` is used, the statement

```
AC_REQUIRE([AC_PROG_CPP])
```

and the macro `AC_PROG_CPP` should contain the statement (anywhere in its body)

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

Then, when the new macro is run, it will invoke `AC_PROG_CPP` if and only if `AC_PROG_CPP` has not already been run.

6.3.2 Suggested Ordering

Some macros should be run before another macro if both are called, but neither requires the other to be called. For example, a macro like `AC_AIX` that changes the behavior of the C compiler (see Section 4.8 [UNIX Variants], page 17) should be called before any macros that run the C compiler. Many of these dependencies are noted in the documentation.

Autoconf provides a way 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`. It is not a fatal error; `configure` is created as usual.

The AC_BEFORE macro causes m4 to print a warning message on the standard error output when a macro is used before another macro which might change its behavior. The macro which should come first should contain a call to AC_BEFORE and the macro which should come later should contain a call to AC_PROVIDE.

For example, AC_AIX contains

```
AC_BEFORE([$0], [AC_COMPILE_CHECK])
```

and AC_COMPILE_CHECK contains

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

As a result, if AC_AIX is called after AC_COMPILE_CHECK, it will note that AC_COMPILE_CHECK has already been called and print a warning message.

6.4 Checking for Files

If you need to check whether a file other than a C header file exists, use '`test -f` *filename*'. 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.

Do not use '`test -x`', because 4.3BSD does not have it. Use '`test -f`' or '`test -r`' instead.

6.5 Checking for Symbols

If you need to check whether a symbol is defined in a C header file, you can use AC_HEADER_EGREP if the symbol is not a C preprocessor macro (see Section 5.2 [General Feature Tests], page 20), or compile a small test program that includes the file and references the symbol (see Section 6.6 [Test Programs], page 32). Don't directly `grep` for the symbol in the file, because on some systems it might be defined in another header file that the file you are checking '`#include`'s.

However, if you need to check for a particular UNIX variant which is distinguished by having certain text in a certain file, then use `grep` (or `egrep`). But 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.

To check whether the Autoconf macros have already defined a certain C preprocessor symbol, you can use a `case` statement like this:

```
case "$DEFS" in
  *HAVE_FOO*) ;;
  *) LIBOBJS="$LIBOBJS foo.o" ;;
esac
```

Make sure to enclose the variable name you are checking (usually `DEFS`) in double quotes, because otherwise some old versions of `bash` misinterpret the statement.

6.6 Test Programs

Autoconf checks for many features by compiling small test programs. To find out whether a library function is available, Autoconf tries to compile a small program that uses it. This is unlike Larry Wall's Metaconfig, which 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 and flexible 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 `configure` to check aspects of the function's runtime behavior if needed. On the other hand, it is sometimes slower than scanning the libraries.

If you need to check for a condition other than whether some symbol exists on the system or has a certain value, then you can't use `AC_COMPILE_CHECK` (see Section 5.2 [General Feature Tests], page 20). You have to write a test program by hand. You can compile and run it using `AC_TEST_PROGRAM` (see Section 5.2 [General Feature Tests], page 20).

Try to avoid writing test programs if possible, because using them prevents people from configuring your package for cross-compiling. If it's really best that you test for a run-time behavior, try to provide a default "worst case" value to use when cross-compiling makes run-time tests impossible. You do this by passing the optional last argument to `AC_TEST_PROGRAM`.

6.6.1 Guidelines for Test Programs

Test programs 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 the argument to `return` in `main` is ignored. They should not write anything to the standard output.

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_STDC_HEADERS`, 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.

6.6.2 Tricks for Test Programs

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`).

6.7 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 excerpted from the `configure.in` for GNU `find`. It uses the shell variable `fstype` to keep track of whether the remaining cases need to be checked. There are several more cases which are not shown here but follow the same pattern.

```
echo checking how to get filesystem type
# SVR4.
AC_TEST_CPP([#include <sys/statvfs.h>
#include <sys/fstyp.h>], AC_DEFINE(FSTYPE_STATVFS) fstype=1)
if test -z "$fstype"; then
# SVR3.
AC_TEST_CPP([#include <sys/statfs.h>
#include <sys/fstyp.h>], AC_DEFINE(FSTYPE_USG_STATFS) fstype=1)
fi
if test -z "$fstype"; then
# AIX.
AC_TEST_CPP([#include <sys/statfs.h>
#include <sys/vmount.h>], AC_DEFINE(FSTYPE_AIX_STATFS) fstype=1)
fi
```

7 Makefiles

Each subdirectory in a distribution should come with a file `Makefile.in`, from which `configure` will produce a `Makefile` in that directory. Most of the substitutions that `configure` does are simple: for each configuration variable that the package uses, it just replaces occurrences of '`@variable@`' with the value that `configure` has determined for that variable. Any occurrences of '`@variable@`' for variables that `configure` does not know about are passed through unchanged.

There is no point in checking for the correct value to give a variable that is never used. Every variable that the `configure` script might set a value for should appear in a '`@variable@`' reference in at least one `Makefile.in`. If `AC_CONFIG_HEADER` is called, `configure` replaces '`@DEFS@`' with '`-DHAVE_CONFIG_H`', since the contents of `DEFS` would be redundant.

See Section "Makefile Conventions" in *The GNU Coding Standards*, for more information on what to put in Makefiles. See Section 9.2 [Sample Makefile.in], page 40, for an example of a real `Makefile.in`.

7.1 Predefined Variables

Some `make` variables are predefined by the Autoconf macros. `AC_SUBST` is called for them automatically (see Section 5.4 [Setting Variables], page 24), so in your `Makefile.in` files you can get their values by enclosing their names in '`@`' characters. The variables that are defined by the general purpose Autoconf macros are:

`exec_prefix` [Variable]
: The installation prefix for architecture-specific files.

`prefix` [Variable]
: The installation prefix for architecture-independent files. See Section 7.2 [Installation Prefixes], page 35, for an alternate way to set this variable.

`srcdir` [Variable]
: The directory that contains the source code for that `Makefile`. See Section 7.2 [Installation Prefixes], page 35, for an alternate way to set this variable.

`top_srcdir` [Variable]
: The top-level source code directory for the package. In the top-level directory, this is the same as `srcdir`.

`DEFS` [Variable]
: '`-D`' options to pass to the C compiler. If `AC_CONFIG_HEADER` is called, `configure` replaces '`@DEFS@`' with '`-DHAVE_CONFIG_H`', since the contents of `DEFS` would be redundant.

`LIBS` [Variable]
: '`-l`' and '`-L`' options to pass to the linker.

`LIBOBJS` [Variable]
: Names of object files (ending in `.o`). Set by `AC_REPLACE_FUNCS` (see Section 5.2 [General Feature Tests], page 20).

7.2 Installation Prefixes

Autoconf-generated `configure` scripts support an alternate method for substituting two particular variables, for compatibility with Cygnus `configure`. This method is not recommended.

If `configure` has figured out a value for the installation prefix, either by the user supplying one on the command line (see Chapter 8 [Invoking configure], page 37) or with `AC_PREFIX` (see Section 5.2 [General Feature Tests], page 20), then it substitutes that value in `Makefile`s that it creates. Wherever a `Makefile.in` contains lines like

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

`configure` substitutes the value it figured out. The substitution only occurs if the word '`prefix`' or '`exec_prefix`' is not preceded by any other characters on the line, and `configure` has figured out a value for the prefix.

There can be separate installation prefixes for architecture-specific files (`exec_prefix`) and architecture-independent files (`prefix`). See Chapter 8 [Invoking configure], page 37, for more information on setting them.

Autoconf `configure` scripts replace these two variables without requiring them to be enclosed in '`@`' characters, and only if they have been set, because the Cygnus `configure` does so. In retrospect, being compatible in this way was a bad decision, because it created an inconsistency in Autoconf without giving significant benefits. This wart will be removed in a future release of Autoconf.

7.3 `VPATH` Substitutions

You might want to compile a software package in a different directory from the one that contains the source code. Doing this allows you to compile the package for several architectures simultaneously from the same copy of the source code and keep multiple sets of object files on disk.

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.in`.

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
```

7.4 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 `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.

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

# autoheader might not change config.h.in
config.h.in: stamp-h.in
stamp-h.in: configure.in aclocal.m4 acconfig.h config.h.top
        cd ${srcdir} && autoheader
        touch ${srcdir}/stamp-h.in

# config.status might not change config.h
config.h: stamp-h
stamp-h: config.h.in config.status
        ./config.status
        touch stamp-h

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

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

See Section 8.2 [Invoking config.status], page 38, for more information on handling configuration-related dependencies.

8 Running `configure` Scripts

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. Here is how to configure a package that uses a `configure` script.

Normally, you just `cd` to the directory containing the package's source code and type '`./configure`'. 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 it is running, it prints some messages that tell what it is doing. If you don't want to see any messages, run `configure` with its standard output redirected to `/dev/null`; for example, '`./configure >/dev/null`'.

To compile the package in a different directory from the one containing the source code, 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 for some reason `configure` is not in the source code directory that you are configuring, then it will report that it can't find the source code. In that case, run `configure` with the option '`--srcdir=`*dir*', where *dir* is the directory that contains the source code.

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*'. Alternately, you can do so by consistently giving a value for the '`prefix`' variable when you run `make`, e.g.,

```
make prefix=/usr/gnu
make prefix=/usr/gnu install
```

You can specify separate installation prefixes for architecture-specific files and architecture-independent files. If you give `configure` the option '`--exec-prefix=`*path*' or set the `make` variable '`exec_prefix`' to *path*, the package will use *path* as the prefix for installing programs and libraries. Data files and documentation will still use the regular prefix. Normally, all files are installed using the same prefix.

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

`configure` also recognizes the following options:

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

`--quiet`

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

`--verbose`
Print the results of the checks.

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

`--x-includes=`*dir*
: X include files are in *dir*.

`--x-libraries=`*dir*
: X library files are in *dir*.

`configure` also accepts and ignores some other options.

8.1 Overriding variables

On systems that require unusual options for compilation or linking that the package's `configure` script does not know about, you can give `configure` initial values for variables by setting them in the environment. In Bourne-compatible shells, you can do that on the command line like this:

```
CC='gcc -traditional' LIBS=-lposix ./configure
```

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

```
env CC='gcc -traditional' LIBS=-lposix ./configure
```

Here are the `make` variables that you might want to override with environment variables when running `configure`.

For these variables, any value given in the environment overrides the value that `configure` would choose:

`CC` [Variable]
: C compiler program. The default is `cc`.

`INSTALL` [Variable]
: Program to use to install files. The default is `install` if you have it, `cp` otherwise.

For these variables, any value given in the environment is added to the value that `configure` chooses:

`DEFS` [Variable]
: Configuration options, in the form '`-Dfoo -Dbar...`'. Do not use this variable in packages that create a configuration header file.

`LIBS` [Variable]
: Libraries to link with, in the form '`-lfoo -lbar...`'.

In the long term, most problems requiring manual intervention should be fixed by updating either the Autoconf macros or the `configure.in` file for that package. See Chapter 3 [Making configure Scripts], page 4, for a discussion of that subject.

8.2 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 prevents `configure` from running `config.status` and creating `Makefile` and other files. (This is so other `Makefile` rules can run `config.status` when it changes; see Section 7.4 [Automatic Remaking], page 36, 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. The default is `/bin/sh`.

The following two 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 7.4 [Automatic Remaking], page 36), `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.status might not change config.h
config.h: stamp-h
stamp-h: config.h.in config.status
        CONFIG_FILES= CONFIG_HEADERS=config.h ./config.status
        touch 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.)

9 An Example

Here are sample `configure.in` and `Makefile.in` files, to give a real illustration of using Autoconf. They are from the GNU `cpio` package, which also includes the `mt` and `rmt` programs. This package does not use a configuration header file; it passes '`-D`' options to the C compiler on the command line.

9.1 Sample `configure.in`

Here is `configure.in` from GNU `cpio`. The `dnl` macro after `AC_SUBST` is suppresses an extra (though harmless) newline in the generated `configure` script (because the `AC_SUBST` macro does not produce any output where it is called).

```
dnl Process this file with autoconf to produce a configure script.
AC_INIT(cpio.h)
PROGS="cpio"
AC_SUBST(PROGS)dnl
AC_PROG_CC
AC_PROG_CPP
AC_GCC_TRADITIONAL
AC_PROG_INSTALL
AC_AIX
AC_MINIX
AC_ISC_POSIX
AC_RETSIGTYPE
AC_MAJOR_HEADER
AC_REMOTE_TAPE
test -n "$have_mtio" && PROGS="$PROGS mt"
AC_RSH
AC_CONST
AC_UID_T
AC_STDC_HEADERS
AC_HAVE_HEADERS(string.h fcntl.h utime.h unistd.h sys/io/trioctl.h)
AC_REPLACE_FUNCS(fnmatch bcopy mkdir strdup)
AC_HAVE_FUNCS(strerror lchown)
AC_VPRINTF
AC_ALLOCA
AC_XENIX_DIR
AC_HAVE_LIBRARY(socket, [LIBS="$LIBS -lsocket"])
AC_HAVE_LIBRARY(nsl, [LIBS="$LIBS -lnsl"])
AC_OUTPUT(Makefile)
```

9.2 Sample `Makefile.in`

Here is `Makefile.in` from GNU `cpio`, with some irrelevant lines omitted, for brevity.

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

```
CC = @CC@

INSTALL = @INSTALL@
INSTALL_PROGRAM = @INSTALL_PROGRAM@
INSTALL_DATA = @INSTALL_DATA@

DEFS = @DEFS@
LIBS = @LIBS@
RTAPELIB = @RTAPELIB@

CFLAGS = -g
LDFLAGS = -g

prefix = /usr/local
exec_prefix = $(prefix)
binprefix =
manprefix =

bindir = $(exec_prefix)/bin
libdir = $(exec_prefix)/lib
mandir = $(prefix)/man/man1
manext = 1

SHELL = /bin/sh

SRCS = copyin.c copyout.c copypass.c defer.c dstring.c global.c \
main.c tar.c util.c error.c getopt.c getopt1.c filemode.c version.c \
rtapelib.c dirname.c idcache.c makepath.c xmalloc.c stripslash.c \
userspec.c xstrdup.c bcopy.c fnmatch.c mkdir.c strdup.c
OBJS = copyin.o copyout.o copypass.o defer.o dstring.o global.o \
main.o tar.o util.o error.o getopt.o getopt1.o filemode.o version.o \
$(RTAPELIB) dirname.o idcache.o makepath.o xmalloc.o stripslash.o \
userspec.o xstrdup.o @LIBOBJS@ @ALLOCA@
# mt source files not shared with cpio.
MT_SRCS = mt.c argmatch.c
MT_OBJS = mt.o argmatch.o error.o getopt.o getopt1.o \
xmalloc.o version.o $(RTAPELIB) @ALLOCA@
HDRS = cpio.h cpiohdr.h tar.h tarhdr.h defer.h dstring.h extern.h filetypes.h \
system.h fnmatch.h getopt.h rmt.h
DISTFILES = $(SRCS) $(HDRS) COPYING COPYING.LIB ChangeLog Makefile.in \
README NEWS INSTALL cpio.1 mt.1 makefile.pc makefile.os2 cpio.def \
configure configure.in mkinstalldirs $(MT_SRCS) rmt.c tcexparg.c alloca.c

all: @PROGS@

.c.o:
        $(CC) -c $(CPPFLAGS) $(DEFS) -I$(srcdir) $(CFLAGS) $<
```

```
install: installdirs all $(srcdir)/cpio.1 $(srcdir)/mt.1
        $(INSTALL_PROGRAM) cpio $(bindir)/$(binprefix)cpio
        test ! -f mt || $(INSTALL_PROGRAM) mt $(bindir)/$(binprefix)mt
        -test ! -f rmt || $(INSTALL_PROGRAM) rmt $(libdir)/rmt
        $(INSTALL_DATA) $(srcdir)/cpio.1 $(mandir)/$(manprefix)cpio.$(manext)
        test ! -f mt || \
        $(INSTALL_DATA) $(srcdir)/mt.1 $(mandir)/$(manprefix)mt.$(manext)

installdirs:
        $(srcdir)/mkinstalldirs $(bindir) $(libdir) $(mandir)

uninstall:
        cd $(bindir); rm -f $(binprefix)cpio $(binprefix)mt
        -rm -f $(libdir)/rmt
        cd $(mandir); rm -f $(manprefix)cpio.$(manext) $(manprefix)mt.$(manext)

check:
        @echo No tests are supplied.

cpio: $(OBJS)
        $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LIBS)

rmt: rmt.o
        $(CC) $(LDFLAGS) -o $@ rmt.o $(LIBS)

mt: $(MT_OBJS)
        $(CC) $(LDFLAGS) -o $@ $(MT_OBJS) $(LIBS)

Makefile: Makefile.in config.status
        $(SHELL) config.status
config.status: configure
        $(SHELL) config.status --recheck
configure: configure.in
        cd $(srcdir); autoconf

TAGS: $(SRCS)
        etags $(SRCS)

clean:
        rm -f cpio rmt mt *.o core

mostlyclean: clean

distclean: clean
        rm -f Makefile config.status
```

```
realclean: distclean
        rm -f TAGS

dist: $(DISTFILES)
        echo cpio-‘sed -e ’/version_string/!d’ \
        -e ’s/[^0-9.]*\([0-9.]*\).*/\1/’ -e q version.c‘ > .fname
        rm -rf ‘cat .fname‘
        mkdir ‘cat .fname‘
        -ln $(DISTFILES) ‘cat .fname‘
        for file in $(DISTFILES); do \
          test -r ‘cat .fname‘/$$file || cp -p $$file ‘cat .fname‘; \
        done
        tar chzf ‘cat .fname‘.tar.gz ‘cat .fname‘
        rm -rf ‘cat .fname‘ .fname
```

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.

S

T

U

V

W

Y

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_'.

P

R

S

T

U

V

W

X

Y

Table of Contents