REF:

https://www.math.utah.edu/docs/info/autoconf\_6.html

Go to the [first](https://www.math.utah.edu/docs/info/autoconf_1.html), [previous](https://www.math.utah.edu/docs/info/autoconf_5.html), [next](https://www.math.utah.edu/docs/info/autoconf_7.html), [last](https://www.math.utah.edu/docs/info/autoconf_19.html) section, [table of contents](https://www.math.utah.edu/docs/info/autoconf_toc.html).

# [Results of Tests](https://www.math.utah.edu/docs/info/autoconf_toc.html" \l "SEC51)

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

* [Defining Symbols](https://www.math.utah.edu/docs/info/autoconf_6.html" \l "SEC52): Defining C preprocessor symbols.
* [Setting Output Variables](https://www.math.utah.edu/docs/info/autoconf_6.html" \l "SEC53): Replacing variables in output files.
* [Caching Results](https://www.math.utah.edu/docs/info/autoconf_6.html" \l "SEC54): Speeding up subsequent configure runs.
* [Printing Messages](https://www.math.utah.edu/docs/info/autoconf_6.html" \l "SEC57): Notifying users of progress or problems.

## [Defining C Preprocessor Symbols](https://www.math.utah.edu/docs/info/autoconf_toc.html" \l "SEC52)

A common action to take in response to a feature test is to define a C preprocessor symbol indicating the results of the test. That is done by calling AC\_DEFINE or AC\_DEFINE\_UNQUOTED.

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

AC\_CHECK\_FUNC(vprintf, AC\_DEFINE(HAVE\_VPRINTF))

if test "$ac\_cv\_func\_vprintf" != yes; then

AC\_CHECK\_FUNC(\_doprnt, AC\_DEFINE(HAVE\_DOPRNT))

fi

If AC\_CONFIG\_HEADER has been called, then instead of creating DEFS, AC\_OUTPUT creates a header file by substituting the correct values into #define statements in a template file. See section [Configuration Header Files](https://www.math.utah.edu/docs/info/autoconf_3.html" \l "SEC15), for more information about this kind of output.

Macro: **AC\_DEFINE** *(variable [, value])*

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

AC\_DEFINE(EQUATION, "$a > $b")

Macro: **AC\_DEFINE\_UNQUOTED** *(variable [, value])*

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

AC\_DEFINE\_UNQUOTED(config\_machfile, "${machfile}")

AC\_DEFINE\_UNQUOTED(GETGROUPS\_T, $ac\_cv\_type\_getgroups)

AC\_DEFINE\_UNQUOTED(${ac\_tr\_hdr})

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

AC\_CHECK\_HEADER(elf.h, AC\_DEFINE(SVR4) LIBS="$LIBS -lelf")

or this:

AC\_CHECK\_HEADER(elf.h,

AC\_DEFINE(SVR4)

LIBS="$LIBS -lelf")

instead of this:

AC\_CHECK\_HEADER(elf.h, AC\_DEFINE(SVR4); LIBS="$LIBS -lelf")

## [Setting Output Variables](https://www.math.utah.edu/docs/info/autoconf_toc.html" \l "SEC53)

One way to record the results of tests is to set output variables, which are shell variables whose values are substituted into files that configure outputs. The two macros below create new output variables. See section [Preset Output Variables](https://www.math.utah.edu/docs/info/autoconf_3.html" \l "SEC12), for a list of output variables that are always available.

Macro: **AC\_SUBST** *(variable)*

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

Macro: **AC\_SUBST\_FILE** *(variable)*

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

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

AC\_SUBST\_FILE(host\_frag)dnl

host\_frag=$srcdir/conf/sun4.mh

and then a `Makefile.in' could contain:

@host\_frag@

## [Caching Results](https://www.math.utah.edu/docs/info/autoconf_toc.html" \l "SEC54)

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

Macro: **AC\_CACHE\_VAL** *(cache-id, commands-to-set-it)*

Ensure that the results of the check identified by cache-id are available. If the results of the check were in the cache file that was read, and configure was not given the `--quiet' or `--silent' option, print a message saying that the result was cached; otherwise, run the shell commands commands-to-set-it. Those commands should have no side effects except for setting the variable cache-id. In particular, they should not call AC\_DEFINE; the code that follows the call to AC\_CACHE\_VAL should do that, based on the cached value. Also, they should not print any messages, for example with AC\_MSG\_CHECKING; do that before calling AC\_CACHE\_VAL, so the messages are printed regardless of whether the results of the check are retrieved from the cache or determined by running the shell commands. If the shell commands are run to determine the value, the value will be saved in the cache file just before configure creates its output files. See section [Cache Variable Names](https://www.math.utah.edu/docs/info/autoconf_6.html" \l "SEC55), for how to choose the name of the cache-id variable.

Macro: **AC\_CACHE\_CHECK** *(message, cache-id, commands)*

A wrapper for AC\_CACHE\_VAL that takes care of printing the messages. This macro provides a convenient shorthand for the most common way to use these macros. It calls AC\_MSG\_CHECKING for message, then AC\_CACHE\_VAL with the cache-id and commands arguments, and AC\_MSG\_RESULT with cache-id.

* [Cache Variable Names](https://www.math.utah.edu/docs/info/autoconf_6.html" \l "SEC55): Shell variables used in caches.
* [Cache Files](https://www.math.utah.edu/docs/info/autoconf_6.html" \l "SEC56): Files configure uses for caching.

### [Cache Variable Names](https://www.math.utah.edu/docs/info/autoconf_toc.html" \l "SEC55)

The names of cache variables should have the following format:

package-prefix\_cv\_value-type\_specific-value[\_additional-options]

for example, `ac\_cv\_header\_stat\_broken' or `ac\_cv\_prog\_gcc\_traditional'. The parts of the variable name are:

package-prefix

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

\_cv\_

Indicates that this shell variable is a cache value.

value-type

A convention for classifying cache values, to produce a rational naming system. The values used in Autoconf are listed in section [Macro Names](https://www.math.utah.edu/docs/info/autoconf_7.html" \l "SEC60).

specific-value

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

additional-options

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

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

### [Cache Files](https://www.math.utah.edu/docs/info/autoconf_toc.html" \l "SEC56)

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

By default, configure uses `./config.cache' as the cache file, creating it if it does not exist already. configure accepts the `--cache-file=file' option to use a different cache file; that is what configure does when it calls configure scripts in subdirectories, so they share the cache. See section [Configuring Other Packages in Subdirectories](https://www.math.utah.edu/docs/info/autoconf_3.html" \l "SEC18), for information on configuring subdirectories with the AC\_CONFIG\_SUBDIRS macro.

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

define([AC\_CACHE\_LOAD], )dnl

define([AC\_CACHE\_SAVE], )dnl

AC\_INIT(whatever)

... rest of configure.in ...

It is wrong to try to distribute cache files for particular system types. There is too much room for error in doing that, and too much administrative overhead in maintaining them. For any features that can't be guessed automatically, use the standard method of the canonical system type and linking files (see section [Manual Configuration](https://www.math.utah.edu/docs/info/autoconf_8.html" \l "SEC66)).

The cache file on a particular system will gradually accumulate whenever someone runs a configure script; it will be initially nonexistent. Running configure merges the new cache results with the existing cache file. The site initialization script can specify a site-wide cache file to use instead of the default, to make it work transparently, as long as the same C compiler is used every time (see section [Setting Site Defaults](https://www.math.utah.edu/docs/info/autoconf_9.html" \l "SEC79)).

## [Printing Messages](https://www.math.utah.edu/docs/info/autoconf_toc.html" \l "SEC57)

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

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

Macro: **AC\_MSG\_CHECKING** *(feature-description)*

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

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

Macro: **AC\_MSG\_RESULT** *(result-description)*

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

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

Macro: **AC\_MSG\_ERROR** *(error-description)*

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

Macro: **AC\_MSG\_WARN** *(problem-description)*

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

The following two macros are an obsolete alternative to AC\_MSG\_CHECKING and AC\_MSG\_RESULT.

Macro: **AC\_CHECKING** *(feature-description)*

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

AC\_CHECKING(if stack overflow is detectable)

Macro: **AC\_VERBOSE** *(result-description)*

This macro is similar to AC\_MSG\_RESULT, except that it is meant to follow a call to AC\_CHECKING instead of AC\_MSG\_CHECKING; it starts the message it prints with a tab. It is considered obsolete.

Go to the [first](https://www.math.utah.edu/docs/info/autoconf_1.html), [previous](https://www.math.utah.edu/docs/info/autoconf_5.html), [next](https://www.math.utah.edu/docs/info/autoconf_7.html), [last](https://www.math.utah.edu/docs/info/autoconf_19.html) section, [table of contents](https://www.math.utah.edu/docs/info/autoconf_toc.html).