

NAME

Config - access Perl configuration information

SYNOPSIS

```
use Config;
if ($Config{usethreads}) {
    print "has thread support\n"
}

use Config qw(myconfig config_sh config_vars config_re);

print myconfig();

print config_sh();

print config_re();

config_vars(qw(osname archname));
```

DESCRIPTION

The Config module contains all the information that was available to the Configure program at Perl build time (over 900 values).

Shell variables from the *config.sh* file (written by Configure) are stored in the readonly-variable `%Config`, indexed by their names.

Values stored in *config.sh* as 'undef' are returned as undefined values. The perl `exists` function can be used to check if a named variable exists.

For a description of the variables, please have a look at the Glossary file, as written in the Porting folder, or use the url: <http://perl5.git.perl.org/perl.git/blob/HEAD:/Porting/Glossary>

`myconfig()`

Returns a textual summary of the major perl configuration values. See also `-v` in "*Command Switches*" in *perlrun*.

`config_sh()`

Returns the entire perl configuration information in the form of the original *config.sh* shell variable assignment script.

`config_re($regex)`

Like `config_sh()` but returns, as a list, only the config entries whose names match the `$regex`.

`config_vars(@names)`

Prints to STDOUT the values of the named configuration variable. Each is printed on a separate line in the form:

```
name='value' ;
```

Names which are unknown are output as `name='UNKNOWN' ;`. See also `-V:name` in "*Command Switches*" in *perlrun*.

`bincompat_options()`

Returns a list of C pre-processor options used when compiling this *perl* binary, which affect its binary compatibility with extensions. `bincompat_options()` and

`non_bincompat_options()` are shown together in the output of `perl -V` as *Compile-time options*.

`non_bincompat_options()`

Returns a list of C pre-processor options used when compiling this *perl* binary, which do not affect binary compatibility with extensions.

`compile_date()`

Returns the compile date (as a string), equivalent to what is shown by `perl -V`

`local_patches()`

Returns a list of the names of locally applied patches, equivalent to what is shown by `perl -V`.

`header_files()`

Returns a list of the header files that should be used as dependencies for XS code, for this version of Perl on this platform.

EXAMPLE

Here's a more sophisticated example of using %Config:

```
use Config;
use strict;

my %sig_num;
my @sig_name;
unless($Config{sig_name} && $Config{sig_num}) {
die "No sigs?";
} else {
my @names = split ' ', $Config{sig_name};
@sig_num{@names} = split ' ', $Config{sig_num};
foreach (@names) {
    $sig_name[$sig_num{$_}] ||= $_;
}
}

print "signal #17 = $sig_name[17]\n";
if ($sig_num{ALRM}) {
print "SIGALRM is $sig_num{ALRM}\n";
}
```

WARNING

Because this information is not stored within the perl executable itself it is possible (but unlikely) that the information does not relate to the actual perl binary which is being used to access it.

The Config module is installed into the architecture and version specific library directory (`$Config{installarchlib}`) and it checks the perl version number when loaded.

The values stored in `config.sh` may be either single-quoted or double-quoted. Double-quoted strings are handy for those cases where you need to include escape sequences in the strings. To avoid runtime variable interpolation, any `$` and `@` characters are replaced by `\$` and `\@`, respectively. This isn't foolproof, of course, so don't embed `\$` or `\@` in double-quoted strings unless you're willing to deal with the consequences. (The slashes will end up escaped and the `$` or `@` will trigger variable interpolation)

GLOSSARY

Most `Config` variables are determined by the `Configure` script on platforms supported by it (which is most UNIX platforms). Some platforms have custom-made `Config` variables, and may thus not have some of the variables described below, or may have extraneous variables specific to that particular port. See the port specific documentation in such cases.

—
_a

From *Unix.U*:

This variable defines the extension used for ordinary library files. For unix, it is `.a`. The `.` is included. Other possible values include `.lib`.

_exe

From *Unix.U*:

This variable defines the extension used for executable files. `DJGPP`, `Cygwin` and `OS/2` use `.exe`. `Stratus VOS` uses `.pm`. On operating systems which do not require a specific extension for executable files, this variable is empty.

_o

From *Unix.U*:

This variable defines the extension used for object files. For unix, it is `.o`. The `.` is included. Other possible values include `.obj`.

a

afs

From *afs.U*:

This variable is set to `true` if `AFS` (Andrew File System) is used on the system, `false` otherwise. It is possible to override this with a hint value or command line option, but you'd better know what you are doing.

afsroot

From *afs.U*:

This variable is by default set to `/afs`. In the unlikely case this is not the correct root, it is possible to override this with a hint value or command line option. This will be used in subsequent tests for AFSness in the configure and test process.

alignbytes

From *alignbytes.U*:

This variable holds the number of bytes required to align a double-- or a long double when applicable. Usual values are 2, 4 and 8. The default is eight, for safety.

ansi2knr

From *ansi2knr.U*:

This variable is set if the user needs to run `ansi2knr`. Currently, this is not supported, so we just abort.

aphostname

From *d_gethname.U*:

This variable contains the command which can be used to compute the host name. The command is fully qualified by its absolute path, to make it safe when used by a process with super-user privileges.

api_revision

From *patchlevel.U*:

The three variables, `api_revision`, `api_version`, and `api_subversion`, specify the version of the oldest perl binary compatible with the present perl. In a full version string such as 5.6.1, `api_revision` is the 5. Prior to 5.5.640, the format was a floating point number, like 5.00563.

perl.c:incpush() and *lib/lib.pm* will automatically search in *\$sitelib/..* for older directories back to the limit specified by these `api_` variables. This is only useful if you have a perl library directory tree structured like the default one. See `INSTALL` for how this works. The versioned `site_perl` directory was introduced in 5.005, so that is the lowest possible value. The version list appropriate for the current system is determined in *inc_version_list.U*.

XXX To do: Since compatibility can depend on compile time options (such as `bincompat`, `longlong`, etc.) it should (perhaps) be set by `Configure`, but currently it isn't. Currently, we read a hard-wired value from *patchlevel.h*. Perhaps what we ought to do is take the hard-wired value from *patchlevel.h* but then modify it if the current `Configure` options warrant. *patchlevel.h* then would use an `#ifdef` guard.

`api_subversion`

From *patchlevel.U*:

The three variables, `api_revision`, `api_version`, and `api_subversion`, specify the version of the oldest perl binary compatible with the present perl. In a full version string such as 5.6.1, `api_subversion` is the 1. See `api_revision` for full details.

`api_version`

From *patchlevel.U*:

The three variables, `api_revision`, `api_version`, and `api_subversion`, specify the version of the oldest perl binary compatible with the present perl. In a full version string such as 5.6.1, `api_version` is the 6. See `api_revision` for full details. As a special case, 5.5.0 is rendered in the old-style as 5.005. (In the 5.005_0x maintenance series, this was the only versioned directory in *\$sitelib*.)

`api_versionstring`

From *patchlevel.U*:

This variable combines `api_revision`, `api_version`, and `api_subversion` in a format such as 5.6.1 (or 5_6_1) suitable for use as a directory name. This is filesystem dependent.

`ar`

From *Loc.U*:

This variable is used internally by `Configure` to determine the full pathname (if any) of the `ar` program. After `Configure` runs, the value is reset to a plain `ar` and is not useful.

`archlib`

From *archlib.U*:

This variable holds the name of the directory in which the user wants to put architecture-dependent public library files for `$package`. It is most often a local directory such as */usr/local/lib*. Programs using this variable must be prepared to deal with filename expansion.

`archlibexp`

From *archlib.U*:

This variable is the same as the `archlib` variable, but is filename expanded at configuration time, for convenient use.

`archname`

From *archname.U*:

This variable is a short name to characterize the current architecture. It is used mainly to construct the default archlib.

`archname64`

From *use64bits.U*:

This variable is used for the 64-bitness part of `$archname`.

`archobjs`

From *Unix.U*:

This variable defines any additional objects that must be linked in with the program on this architecture. On unix, it is usually empty. It is typically used to include emulations of unix calls or other facilities. For perl on OS/2, for example, this would include *os2/os2.obj*.

`asctime_r_proto`

From *d_asctime_r.U*:

This variable encodes the prototype of `asctime_r`. It is zero if `d_asctime_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_asctime_r` is defined.

`awk`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the awk program. After Configure runs, the value is reset to a plain `awk` and is not useful.

b

`baserev`

From *baserev.U*:

The base revision level of this package, from the *.package* file.

`bash`

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

`bin`

From *bin.U*:

This variable holds the name of the directory in which the user wants to put publicly executable images for the package in question. It is most often a local directory such as */usr/local/bin*. Programs using this variable must be prepared to deal with *~name* substitution.

`bin_ELF`

From *d/src.U*:

This variable saves the result from configure if generated binaries are in `ELF` format. Only set to defined when the test has actually been performed, and the result was positive.

`binexp`

From *bin.U*:

This is the same as the `bin` variable, but is filename expanded at configuration time, for use in your makefiles.

`bison`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the

bison program. After Configure runs, the value is reset to a plain `bison` and is not useful.

`byacc`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the `byacc` program. After Configure runs, the value is reset to a plain `byacc` and is not useful.

`byteorder`

From *byteorder.U*:

This variable holds the byte order in a `UV`. In the following, larger digits indicate more significance. The variable `byteorder` is either 4321 on a big-endian machine, or 1234 on a little-endian, or 87654321 on a Cray ... or 3412 with weird order !

C

`c`

From *n.U*:

This variable contains the `\c` string if that is what causes the `echo` command to suppress newline. Otherwise it is null. Correct usage is `$echo $n "prompt for a question: $c"`.

`castflags`

From *d_castneg.U*:

This variable contains a flag that precise difficulties the compiler has casting odd floating values to unsigned long: 0 = ok 1 = couldn't cast < 0 2 = couldn't cast >= 0x80000000 4 = couldn't cast in argument expression list

`cat`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the `cat` program. After Configure runs, the value is reset to a plain `cat` and is not useful.

`cc`

From *cc.U*:

This variable holds the name of a command to execute a C compiler which can resolve multiple global references that happen to have the same name. Usual values are `cc` and `gcc`. Fervent ANSI compilers may be called `c89`. AIX has `xlC`.

`ccdlflags`

From *d/src.U*:

This variable contains any special flags that might need to be passed with `cc -c` to compile modules to be used to create a shared library that will be used for dynamic loading. For `hpux`, this should be `+z`. It is up to the makefile to use it.

`ccdlflags`

From *d/src.U*:

This variable contains any special flags that might need to be passed to `cc` to link with a shared library for dynamic loading. It is up to the makefile to use it. For `sunos 4.1`, it should be empty.

`ccflags`

From *ccflags.U*:

This variable contains any additional C compiler flags desired by the user. It is up to the Makefile to use this.

ccflags_uselargefiles

From *uselfs.U*:

This variable contains the compiler flags needed by large file builds and added to ccflags by hints files.

ccname

From *Checkcc.U*:

This can set either by hints files or by Configure. If using gcc, this is gcc, and if not, usually equal to cc, unimpressive, no? Some platforms, however, make good use of this by storing the flavor of the C compiler being used here. For example if using the Sun WorkShop suite, ccname will be *workshop*.

ccsymbols

From *Cppsym.U*:

The variable contains the symbols defined by the C compiler alone. The symbols defined by cpp or by cc when it calls cpp are not in this list, see cppsymbols and cppccsymbols. The list is a space-separated list of symbol=value tokens.

ccversion

From *Checkcc.U*:

This can set either by hints files or by Configure. If using a (non-gcc) vendor cc, this variable may contain a version for the compiler.

cf_by

From *cf_who.U*:

Login name of the person who ran the Configure script and answered the questions. This is used to tag both *config.sh* and *config_h.SH*.

cf_email

From *cf_email.U*:

Electronic mail address of the person who ran Configure. This can be used by units that require the user's e-mail, like *MailList.U*.

cf_time

From *cf_who.U*:

Holds the output of the `date` command when the configuration file was produced. This is used to tag both *config.sh* and *config_h.SH*.

charbits

From *charsize.U*:

This variable contains the value of the CHARBITS symbol, which indicates to the C program how many bits there are in a character.

charsize

From *charsize.U*:

This variable contains the value of the CHARSIZE symbol, which indicates to the C program how many bytes there are in a character.

chgrp

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

chmod

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the `chmod` program. After Configure runs, the value is reset to a plain `chmod` and is not useful.

chown

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

clocktype

From *d_times.U*:

This variable holds the type returned by `times()`. It can be `long`, or `clock_t` on BSD sites (in which case `<sys/types.h>` should be included).

comm

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the `comm` program. After Configure runs, the value is reset to a plain `comm` and is not useful.

compress

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

config_arg0

From *Options.U*:

This variable contains the string used to invoke the Configure command, as reported by the shell in the `$0` variable.

config_argc

From *Options.U*:

This variable contains the number of command-line arguments passed to Configure, as reported by the shell in the `$#` variable. The individual arguments are stored as variables `config_arg1`, `config_arg2`, etc.

config_args

From *Options.U*:

This variable contains a single string giving the command-line arguments passed to Configure. Spaces within arguments, quotes, and escaped characters are not correctly preserved. To reconstruct the command line, you must assemble the individual command line pieces, given in `config_arg[0-9]*`.

contains

From *contains.U*:

This variable holds the command to do a `grep` with a proper return status. On most sane systems it is simply `grep`. On insane systems it is a `grep` followed by a `cat` followed by a test. This variable is primarily for the use of other Configure units.

cp

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the `cp`

program. After Configure runs, the value is reset to a plain `cp` and is not useful.

`cpio`

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

`cpp`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the `cpp` program. After Configure runs, the value is reset to a plain `cpp` and is not useful.

`cpp_stuff`

From *cpp_stuff.U*:

This variable contains an identification of the concatenation mechanism used by the C preprocessor.

`cppccsymbols`

From *Cppsym.U*:

The variable contains the symbols defined by the C compiler when it calls `cpp`. The symbols defined by the `cc` alone or `cpp` alone are not in this list, see `ccsymbols` and `cppsymbols`. The list is a space-separated list of `symbol=value` tokens.

`cppflags`

From *ccflags.U*:

This variable holds the flags that will be passed to the C pre-processor. It is up to the Makefile to use it.

`cpplast`

From *cppstdin.U*:

This variable has the same functionality as `cppminus`, only it applies to `cpprun` and not `cppstdin`.

`cppminus`

From *cppstdin.U*:

This variable contains the second part of the string which will invoke the C preprocessor on the standard input and produce to standard output. This variable will have the value `-` if `cppstdin` needs a minus to specify standard input, otherwise the value is `""`.

`cpprun`

From *cppstdin.U*:

This variable contains the command which will invoke a C preprocessor on standard input and put the output to `stdout`. It is guaranteed not to be a wrapper and may be a null string if no preprocessor can be made directly available. This preprocessor might be different from the one used by the C compiler. Don't forget to append `cpplast` after the preprocessor options.

`cppstdin`

From *cppstdin.U*:

This variable contains the command which will invoke the C preprocessor on standard input and put the output to `stdout`. It is primarily used by other Configure units that ask about preprocessor symbols.

`cppsymbols`

From *Cppsym.U*:

The variable contains the symbols defined by the C preprocessor alone. The symbols defined by cc or by cc when it calls cpp are not in this list, see `ccsymbols` and `cppccsymbols`. The list is a space-separated list of symbol=value tokens.

`crypt_r_proto`

From *d_crypt_r.U*:

This variable encodes the prototype of `crypt_r`. It is zero if `d_crypt_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_crypt_r` is defined.

`cryptlib`

From *d_crypt.U*:

This variable holds `-lcrypt` or the path to a *libcrypt.a* archive if the `crypt()` function is not defined in the standard C library. It is up to the Makefile to use this.

`csch`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the `csch` program. After Configure runs, the value is reset to a plain `csch` and is not useful.

`ctermid_r_proto`

From *d_ctermid_r.U*:

This variable encodes the prototype of `ctermid_r`. It is zero if `d_ctermid_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_ctermid_r` is defined.

`ctime_r_proto`

From *d_ctime_r.U*:

This variable encodes the prototype of `ctime_r`. It is zero if `d_ctime_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_ctime_r` is defined.

d

`d__fwalk`

From *d__fwalk.U*:

This variable conditionally defines `HAS__FWALK` if `_fwalk()` is available to apply a function to all the file handles.

`d_access`

From *d_access.U*:

This variable conditionally defines `HAS_ACCESS` if the `access()` system call is available to check for access permissions using real IDs.

`d_accessx`

From *d_accessx.U*:

This variable conditionally defines the `HAS_ACCESSX` symbol, which indicates to the C program that the `accessx()` routine is available.

`d_acosh`

From *d_acosh.U*:

This variable conditionally defines the `HAS_ACOSH` symbol, which indicates to the C program that the `acosh()` routine is available.

`d_aintl`

From *d_aintl.U*:

This variable conditionally defines the `HAS_AINTL` symbol, which indicates to the C program that the `aintl()` routine is available. If `copysignl` is also present we can emulate `modfl`.

`d_alarm`

From *d_alarm.U*:

This variable conditionally defines the `HAS_ALARM` symbol, which indicates to the C program that the `alarm()` routine is available.

`d_archlib`

From *archlib.U*:

This variable conditionally defines `ARCHLIB` to hold the pathname of architecture-dependent library files for `$package`. If `$archlib` is the same as `$privlib`, then this is set to `undef`.

`d_asctime64`

From *d_timefuncs64.U*:

This variable conditionally defines the `HAS_ASCTIME64` symbol, which indicates to the C program that the `asctime64()` routine is available.

`d_asctime_r`

From *d_asctime_r.U*:

This variable conditionally defines the `HAS_ASCTIME_R` symbol, which indicates to the C program that the `asctime_r()` routine is available.

`d_asinh`

From *d_asinh.U*:

This variable conditionally defines the `HAS_ASINH` symbol, which indicates to the C program that the `asinh()` routine is available.

`d_atanh`

From *d_atanh.U*:

This variable conditionally defines the `HAS_ATANH` symbol, which indicates to the C program that the `atanh()` routine is available.

`d_atolf`

From *atolf.U*:

This variable conditionally defines the `HAS_ATOLF` symbol, which indicates to the C program that the `atolf()` routine is available.

`d_atoll`

From *atoll.U*:

This variable conditionally defines the `HAS_ATOLL` symbol, which indicates to the C program that the `atoll()` routine is available.

`d_attribute_deprecated`

From *d_attribut.U*:

This variable conditionally defines `HASATTRIBUTE_DEPRECATED`, which indicates that GCC can handle the attribute for marking deprecated APIs

`d_attribute_format`

From *d_attribut.U*:

This variable conditionally defines `HASATTRIBUTE_FORMAT`, which indicates the C compiler

can check for printf-like formats.

`d_attribute_malloc`

From *d_attribut.U*:

This variable conditionally defines `HASATTRIBUTE_MALLOC`, which indicates the C compiler can understand functions as having malloc-like semantics.

`d_attribute_nonnull`

From *d_attribut.U*:

This variable conditionally defines `HASATTRIBUTE_NONNULL`, which indicates that the C compiler can know that certain arguments must not be `NULL`, and will check accordingly at compile time.

`d_attribute_noreturn`

From *d_attribut.U*:

This variable conditionally defines `HASATTRIBUTE_NORETURN`, which indicates that the C compiler can know that certain functions are guaranteed never to return.

`d_attribute_pure`

From *d_attribut.U*:

This variable conditionally defines `HASATTRIBUTE_PURE`, which indicates that the C compiler can know that certain functions are `pure` functions, meaning that they have no side effects, and only rely on function input *and/or* global data for their results.

`d_attribute_unused`

From *d_attribut.U*:

This variable conditionally defines `HASATTRIBUTE_UNUSED`, which indicates that the C compiler can know that certain variables and arguments may not always be used, and to not throw warnings if they don't get used.

`d_attribute_warn_unused_result`

From *d_attribut.U*:

This variable conditionally defines `HASATTRIBUTE_WARN_UNUSED_RESULT`, which indicates that the C compiler can know that certain functions have a return values that must not be ignored, such as `malloc()` or `open()`.

`d_backtrace`

From *d_backtrace.U*:

This variable conditionally defines the `HAS_BACKTRACE` symbol, which indicates to the C program that the `backtrace()` routine is available to get a stack trace.

`d_bcmp`

From *d_bcmp.U*:

This variable conditionally defines the `HAS_BCMP` symbol if the `bcmp()` routine is available to compare strings.

`d_bcopy`

From *d_bcopy.U*:

This variable conditionally defines the `HAS_BCOPY` symbol if the `bcopy()` routine is available to copy strings.

`d_bsd`

From *Guess.U*:

This symbol conditionally defines the symbol `BSD` when running on a BSD system.

`d_bsdgetpgrp`

From *d_getpgrp.U*:

This variable conditionally defines `USE_BSD_GETPGRP` if `getpgrp` needs one arguments whereas `USG` one needs none.

`d_bsdsetpgrp`

From *d_setpgrp.U*:

This variable conditionally defines `USE_BSD_SETPGRP` if `setpgrp` needs two arguments whereas `USG` one needs none. See also `d_setpgid` for a `POSIX` interface.

`d_builtin_choose_expr`

From *d_builtin.U*:

This conditionally defines `HAS_BUILTIN_CHOOSE_EXPR`, which indicates that the compiler supports `__builtin_choose_expr(x,y,z)`. This built-in function is analogous to the `x?y:z` operator in C, except that the expression returned has its type unaltered by promotion rules. Also, the built-in function does not evaluate the expression that was not chosen.

`d_builtin_expect`

From *d_builtin.U*:

This conditionally defines `HAS_BUILTIN_EXPECT`, which indicates that the compiler supports `__builtin_expect(exp,c)`. You may use `__builtin_expect` to provide the compiler with branch prediction information.

`d_bzero`

From *d_bzero.U*:

This variable conditionally defines the `HAS_BZERO` symbol if the `bzero()` routine is available to set memory to 0.

`d_c99_variadic_macros`

From *d_c99_variadic.U*:

This variable conditionally defines the `HAS_C99_VARIADIC_MACROS` symbol, which indicates to the C program that C99 variadic macros are available.

`d_casti32`

From *d_casti32.U*:

This variable conditionally defines `CASTI32`, which indicates whether the C compiler can cast large floats to 32-bit ints.

`d_castneg`

From *d_castneg.U*:

This variable conditionally defines `CASTNEG`, which indicates whether the C compiler can cast negative float to unsigned.

`d_cbrt`

From *d_cbrt.U*:

This variable conditionally defines the `HAS_CBRT` symbol, which indicates to the C program that the `cbrt()` (cube root) function is available.

`d_charvspr`

From *d_vprintf.U*:

This variable conditionally defines `CHARVSPRINTF` if this system has `vsprintf` returning type

(char*). The trend seems to be to declare it as "int vsprintf()".

`d_chown`

From *d_chown.U*:

This variable conditionally defines the `HAS_CHOWN` symbol, which indicates to the C program that the `chown()` routine is available.

`d_chroot`

From *d_chroot.U*:

This variable conditionally defines the `HAS_CHROOT` symbol, which indicates to the C program that the `chroot()` routine is available.

`d_chsize`

From *d_chsize.U*:

This variable conditionally defines the `CHSIZE` symbol, which indicates to the C program that the `chsize()` routine is available to truncate files. You might need a `-lx` to get this routine.

`d_class`

From *d_class.U*:

This variable conditionally defines the `HAS_CLASS` symbol, which indicates to the C program that the `class()` routine is available.

`d_clearenv`

From *d_clearenv.U*:

This variable conditionally defines the `HAS_CLEARENV` symbol, which indicates to the C program that the `clearenv()` routine is available.

`d_closedir`

From *d_closedir.U*:

This variable conditionally defines `HAS_CLOSEDIR` if `closedir()` is available.

`d_cmsgHDR_s`

From *d_cmsgHDR_s.U*:

This variable conditionally defines the `HAS_STRUCT_CMSGHDR` symbol, which indicates that the `struct cmsgHDR` is supported.

`d_const`

From *d_const.U*:

This variable conditionally defines the `HASCONST` symbol, which indicates to the C program that this C compiler knows about the `const` type.

`d_copysign`

From *d_copysign.U*:

This variable conditionally defines the `HAS_COPYSIGN` symbol, which indicates to the C program that the `copysign()` routine is available.

`d_copysignl`

From *d_copysignl.U*:

This variable conditionally defines the `HAS_COPYSIGNL` symbol, which indicates to the C program that the `copysignl()` routine is available. If `aintrl` is also present we can emulate `modfl`.

`d_cplusplus`

From *d_cplusplus.U*:

This variable conditionally defines the `USE_CPLUSPLUS` symbol, which indicates that a C++ compiler was used to compile Perl and will be used to compile extensions.

`d_crypt`

From *d_crypt.U*:

This variable conditionally defines the `CRYPT` symbol, which indicates to the C program that the `crypt()` routine is available to encrypt passwords and the like.

`d_crypt_r`

From *d_crypt_r.U*:

This variable conditionally defines the `HAS_CRYPT_R` symbol, which indicates to the C program that the `crypt_r()` routine is available.

`d_csh`

From *d_csh.U*:

This variable conditionally defines the `CSH` symbol, which indicates to the C program that the C-shell exists.

`d_ctermid`

From *d_ctermid.U*:

This variable conditionally defines `CTERMID` if `ctermid()` is available to generate filename for terminal.

`d_ctermid_r`

From *d_ctermid_r.U*:

This variable conditionally defines the `HAS_CTERMID_R` symbol, which indicates to the C program that the `ctermid_r()` routine is available.

`d_ctime64`

From *d_timefuncs64.U*:

This variable conditionally defines the `HAS_CTIME64` symbol, which indicates to the C program that the `ctime64()` routine is available.

`d_ctime_r`

From *d_ctime_r.U*:

This variable conditionally defines the `HAS_CTIME_R` symbol, which indicates to the C program that the `ctime_r()` routine is available.

`d_cuserid`

From *d_cuserid.U*:

This variable conditionally defines the `HAS_CUSERID` symbol, which indicates to the C program that the `cuserid()` routine is available to get character login names.

`d_dbl_dig`

From *d_dbl_dig.U*:

This variable conditionally defines `d_dbl_dig` if this system's header files provide `DBL_DIG`, which is the number of significant digits in a double precision number.

`d_dbminitproto`

From *d_dbminitproto.U*:

This variable conditionally defines the `HAS_DBMINIT_PROTO` symbol, which indicates to the C program that the system provides a prototype for the `dbminit()` function. Otherwise, it is up to the program to supply one.

`d_difftime`

From *d_difftime.U*:

This variable conditionally defines the `HAS_DIFFTIME` symbol, which indicates to the C program that the `difftime()` routine is available.

`d_difftime64`

From *d_timefuncs64.U*:

This variable conditionally defines the `HAS_DIFFTIME64` symbol, which indicates to the C program that the `difftime64 ()` routine is available.

`d_dir_dd_fd`

From *d_dir_dd_fd.U*:

This variable conditionally defines the `HAS_DIR_DD_FD` symbol, which indicates that the `DIR` directory stream type contains a member variable called `dd_fd`.

`d_dirfd`

From *d_dirfd.U*:

This variable conditionally defines the `HAS_DIRFD` constant, which indicates to the C program that `dirfd()` is available to return the file descriptor of a directory stream.

`d_dirnamlen`

From *i_dirent.U*:

This variable conditionally defines `DIRNAMLEN`, which indicates to the C program that the length of directory entry names is provided by a `d_namelen` field.

`d_dladdr`

From *d_dladdr.U*:

This variable conditionally defines the `HAS_DLADDR` symbol, which indicates to the C program that the `dladdr()` routine is available to get a stack trace.

`d_dLError`

From *d_dLError.U*:

This variable conditionally defines the `HAS_DLERROR` symbol, which indicates to the C program that the `dLError()` routine is available.

`d_dlopen`

From *d_dlopen.U*:

This variable conditionally defines the `HAS_DLOPEN` symbol, which indicates to the C program that the `dlopen()` routine is available.

`d_dlsymun`

From *d_dlsymun.U*:

This variable conditionally defines `DLSYM_NEEDS_UNDERSCORE`, which indicates that we need to prepend an underscore to the symbol name before calling `dlsym()`.

`d_dosuid`

From *d_dosuid.U*:

This variable conditionally defines the symbol `DOSUID`, which tells the C program that it should insert `setuid` emulation code on hosts which have `setuid #!` scripts disabled.

`d_drand48_r`

From *d_drand48_r.U*:

This variable conditionally defines the `HAS_DRAND48_R` symbol, which indicates to the C program that the `drand48_r()` routine is available.

`d_drand48proto`

From *d_drand48proto.U*:

This variable conditionally defines the `HAS_DRAND48_PROTO` symbol, which indicates to the C program that the system provides a prototype for the `drand48()` function. Otherwise, it is up to the program to supply one.

`d_dup2`

From *d_dup2.U*:

This variable conditionally defines `HAS_DUP2` if `dup2()` is available to duplicate file descriptors.

`d_duplocale`

From *d_newlocale.U*:

This variable conditionally defines the `HAS_DUPLOCALE` symbol, which indicates to the C program that the `duplocale()` routine is available to duplicate a locale object.

`d_eaccess`

From *d_eaccess.U*:

This variable conditionally defines the `HAS_EACCESS` symbol, which indicates to the C program that the `eaccess()` routine is available.

`d_endgrent`

From *d_endgrent.U*:

This variable conditionally defines the `HAS_ENDGRENT` symbol, which indicates to the C program that the `endgrent()` routine is available for sequential access of the group database.

`d_endgrent_r`

From *d_endgrent_r.U*:

This variable conditionally defines the `HAS_ENDGRENT_R` symbol, which indicates to the C program that the `endgrent_r()` routine is available.

`d_endhent`

From *d_endhent.U*:

This variable conditionally defines `HAS_ENDHOSTENT` if `endhostent()` is available to close whatever was being used for host queries.

`d_endhostent_r`

From *d_endhostent_r.U*:

This variable conditionally defines the `HAS_ENDHOSTENT_R` symbol, which indicates to the C program that the `endhostent_r()` routine is available.

`d_endnetent`

From *d_endnetent.U*:

This variable conditionally defines `HAS_ENDNETENT` if `endnetent()` is available to close whatever was being used for network queries.

`d_endnetent_r`

From *d_endnetent_r.U*:

This variable conditionally defines the `HAS_ENDNETENT_R` symbol, which indicates to the C program that the `endnetent_r()` routine is available.

`d_endpent`

From *d_endpent.U*:

This variable conditionally defines `HAS_ENDPROTOENT` if `endprotoent()` is available to close whatever was being used for protocol queries.

`d_endprotoent_r`

From *d_endprotoent_r.U*:

This variable conditionally defines the `HAS_ENDPROTOENT_R` symbol, which indicates to the C program that the `endprotoent_r()` routine is available.

`d_endpwent`

From *d_endpwent.U*:

This variable conditionally defines the `HAS_ENDPWENT` symbol, which indicates to the C program that the `endpwent()` routine is available for sequential access of the `passwd` database.

`d_endpwent_r`

From *d_endpwent_r.U*:

This variable conditionally defines the `HAS_ENDPWENT_R` symbol, which indicates to the C program that the `endpwent_r()` routine is available.

`d_endsent`

From *d_endsent.U*:

This variable conditionally defines `HAS_ENDSERVENT` if `endservent()` is available to close whatever was being used for service queries.

`d_endservent_r`

From *d_endservent_r.U*:

This variable conditionally defines the `HAS_ENDSERVENT_R` symbol, which indicates to the C program that the `endservent_r()` routine is available.

`d_eofnblk`

From *nblock_io.U*:

This variable conditionally defines `EOF_NONBLOCK` if `EOF` can be seen when reading from a non-blocking I/O source.

`d_erf`

From *d_erf.U*:

This variable conditionally defines the `HAS_ERF` symbol, which indicates to the C program that the `erf()` routine is available.

`d_erfc`

From *d_erfc.U*:

This variable conditionally defines the `HAS_ERFC` symbol, which indicates to the C program that the `erfc()` routine is available.

`d_eunice`

From *Guess.U*:

This variable conditionally defines the symbols `EUNICE` and `VAX`, which alerts the C program that it must deal with idiosyncrasies of `VMS`.

`d_exp2`

From *d_exp2.U*:

This variable conditionally defines the HAS_EXP2 symbol, which indicates to the C program that the exp2() routine is available.

`d_expm1`

From *d_expm1.U*:

This variable conditionally defines the HAS_EXPM1 symbol, which indicates to the C program that the expm1() routine is available.

`d_faststdio`

From *d_faststdio.U*:

This variable conditionally defines the HAS_FAST_STDIO symbol, which indicates to the C program that the "fast stdio" is available to manipulate the stdio buffers directly.

`d_fchdir`

From *d_fchdir.U*:

This variable conditionally defines the HAS_FCHDIR symbol, which indicates to the C program that the fchdir() routine is available.

`d_fchmod`

From *d_fchmod.U*:

This variable conditionally defines the HAS_FCHMOD symbol, which indicates to the C program that the fchmod() routine is available to change mode of opened files.

`d_fchown`

From *d_fchown.U*:

This variable conditionally defines the HAS_FCHOWN symbol, which indicates to the C program that the fchown() routine is available to change ownership of opened files.

`d_fcntl`

From *d_fcntl.U*:

This variable conditionally defines the HAS_FCNTL symbol, and indicates whether the fcntl() function exists

`d_fcntl_can_lock`

From *d_fcntl_can_lock.U*:

This variable conditionally defines the FCNTL_CAN_LOCK symbol and indicates whether file locking with fcntl() works.

`d_fd_macros`

From *d_fd_set.U*:

This variable contains the eventual value of the HAS_FD_MACROS symbol, which indicates if your C compiler knows about the macros which manipulate an fd_set.

`d_fd_set`

From *d_fd_set.U*:

This variable contains the eventual value of the HAS_FD_SET symbol, which indicates if your C compiler knows about the fd_set typedef.

`d_fdclose`

From *d_fdclose.U*:

This variable conditionally defines the HAS_FDCLOSE symbol, which indicates to the C

program that the `fdclose()` routine is available.

`d_fdim`

From *d_fdim.U*:

This variable conditionally defines the `HAS_FDIM` symbol, which indicates to the C program that the `fdim()` routine is available.

`d_fds_bits`

From *d_fd_set.U*:

This variable contains the eventual value of the `HAS_FDS_BITS` symbol, which indicates if your `fd_set` typedef contains the `fds_bits` member. If you have an `fd_set` typedef, but the dweebs who installed it did a half-fast job and neglected to provide the macros to manipulate an `fd_set`, `HAS_FDS_BITS` will let us know how to fix the gaffe.

`d_fegetround`

From *d_fegetround.U*:

This variable conditionally defines `HAS_FGETROUND` if `fegetround()` is available to get the floating point rounding mode.

`d_fgetpos`

From *d_fgetpos.U*:

This variable conditionally defines `HAS_FGETPOS` if `fgetpos()` is available to get the file position indicator.

`d_finite`

From *d_finite.U*:

This variable conditionally defines the `HAS_FINITE` symbol, which indicates to the C program that the `finite()` routine is available.

`d_finitel`

From *d_finitel.U*:

This variable conditionally defines the `HAS_FINITEL` symbol, which indicates to the C program that the `finitel()` routine is available.

`d_flexfnam`

From *d_flexfnam.U*:

This variable conditionally defines the `FLEXFILENAMES` symbol, which indicates that the system supports filenames longer than 14 characters.

`d_flock`

From *d_flock.U*:

This variable conditionally defines `HAS_FLOCK` if `flock()` is available to do file locking.

`d_flockproto`

From *d_flockproto.U*:

This variable conditionally defines the `HAS_FLOCK_PROTO` symbol, which indicates to the C program that the system provides a prototype for the `flock()` function. Otherwise, it is up to the program to supply one.

`d_fma`

From *d_fma.U*:

This variable conditionally defines the `HAS_FMA` symbol, which indicates to the C program that the `fma()` routine is available.

`d_fmax`

From *d_fmax.U*:

This variable conditionally defines the `HAS_FMAX` symbol, which indicates to the C program that the `fmax()` routine is available.

`d_fmin`

From *d_fmin.U*:

This variable conditionally defines the `HAS_FMIN` symbol, which indicates to the C program that the `fmin()` routine is available.

`d_fork`

From *d_fork.U*:

This variable conditionally defines the `HAS_FORK` symbol, which indicates to the C program that the `fork()` routine is available.

`d_fp_class`

From *d_fp_class.U*:

This variable conditionally defines the `HAS_FP_CLASS` symbol, which indicates to the C program that the `fp_class()` routine is available.

`d_fp_classify`

From *d_fpclassify.U*:

This variable conditionally defines the `HAS_FP_CLASSIFY` symbol, which indicates to the C program that the `fp_classify()` routine is available.

`d_fp_classl`

From *d_fp_classl.U*:

This variable conditionally defines the `HAS_FP_CLASSL` symbol, which indicates to the C program that the `fp_classl()` routine is available.

`d_fpathconf`

From *d_pathconf.U*:

This variable conditionally defines the `HAS_FPATHCONF` symbol, which indicates to the C program that the `pathconf()` routine is available to determine file-system related limits and options associated with a given open file descriptor.

`d_fpclass`

From *d_fpclass.U*:

This variable conditionally defines the `HAS_FPCLASS` symbol, which indicates to the C program that the `fpclass()` routine is available.

`d_fpclassify`

From *d_fpclassify.U*:

This variable conditionally defines the `HAS_FPCLASSIFY` symbol, which indicates to the C program that the `fpclassify()` routine is available.

`d_fpclassl`

From *d_fpclassl.U*:

This variable conditionally defines the `HAS_FPCLASSL` symbol, which indicates to the C program that the `fpclassl()` routine is available.

`d_fpgetround`

From *d_fpgetround.U*:

This variable conditionally defines `HAS_FPGETROUND` if `fpgetround()` is available to get the floating point rounding mode.

`d_fpos64_t`

From *d_fpos64_t.U*:

This symbol will be defined if the C compiler supports `fpos64_t`.

`d_freelocale`

From *d_newlocale.U*:

This variable conditionally defines the `HAS_FREELocale` symbol, which indicates to the C program that the `freelocale()` routine is available to deallocate the resources associated with a locale object.

`d_frexp1`

From *d_frexp1.U*:

This variable conditionally defines the `HAS_FREXP1` symbol, which indicates to the C program that the `frexp1()` routine is available.

`d_fs_data_s`

From *d_fs_data_s.U*:

This variable conditionally defines the `HAS_STRUCT_FS_DATA` symbol, which indicates that the `struct fs_data` is supported.

`d_fseeko`

From *d_fseeko.U*:

This variable conditionally defines the `HAS_FSEEKO` symbol, which indicates to the C program that the `fseeko()` routine is available.

`d_fsetpos`

From *d_fsetpos.U*:

This variable conditionally defines `HAS_FSETPOS` if `fsetpos()` is available to set the file position indicator.

`d_fstatfs`

From *d_fstatfs.U*:

This variable conditionally defines the `HAS_FSTATFS` symbol, which indicates to the C program that the `fstatfs()` routine is available.

`d_fstatvfs`

From *d_statvfs.U*:

This variable conditionally defines the `HAS_FSTATVFS` symbol, which indicates to the C program that the `fstatvfs()` routine is available.

`d_fsync`

From *d_fsync.U*:

This variable conditionally defines the `HAS_FSYNC` symbol, which indicates to the C program that the `fsync()` routine is available.

`d_ftello`

From *d_ftello.U*:

This variable conditionally defines the `HAS_FTELLO` symbol, which indicates to the C program

that the `ftello()` routine is available.

`d_fmtime`

From *d_fmtime.U*:

This variable conditionally defines the `HAS_FTIME` symbol, which indicates that the `ftime()` routine exists. The `ftime()` routine is basically a sub-second accuracy clock.

`d_futimes`

From *d_futimes.U*:

This variable conditionally defines the `HAS_FUTIMES` symbol, which indicates to the C program that the `futimes()` routine is available.

`d_Gconvert`

From *d_gconvert.U*:

This variable holds what `Gconvert` is defined as to convert floating point numbers into strings. By default, `Configure` sets `this` macro to use the first of `gconvert`, `gcvt`, or `sprintf` that pass `sprintf-%g`-like behavior tests. If perl is using long doubles, the macro uses the first of the following functions that pass `Configure`'s tests: `qgcvt`, `sprintf` (if `Configure` knows how to make `sprintf` format long doubles--see `SPRIGldbl`), `gconvert`, `gcvt`, and `sprintf` (casting to double). The `gconvert_preference` and `gconvert_ld_preference` variables can be used to alter `Configure`'s preferences, for doubles and long doubles, respectively. If present, they contain a space-separated list of one or more of the above function names in the order they should be tried.

`d_Gconvert` may be set to override `Configure` with a platform-specific function. If this function expects a double, a different value may need to be set by the *uselongdouble.cbu* call-back unit so that long doubles can be formatted without loss of precision.

`d_gdbm_ndbm_h_uses_prototypes`

From *i_ndbm.U*:

This variable conditionally defines the `NDBM_H_USES_PROTOTYPES` symbol, which indicates that the *gdbm-ndbm.h* include file uses real ANSI C prototypes instead of K&R style function declarations. K&R style declarations are unsupported in C++, so the include file requires special handling when using a C++ compiler and this variable is undefined. Consult the different `d_*ndbm_h_uses_prototypes` variables to get the same information for alternative *ndbm.h* include files.

`d_gdbmndbm_h_uses_prototypes`

From *i_ndbm.U*:

This variable conditionally defines the `NDBM_H_USES_PROTOTYPES` symbol, which indicates that the *gdbm/ndbm.h* include file uses real ANSI C prototypes instead of K&R style function declarations. K&R style declarations are unsupported in C++, so the include file requires special handling when using a C++ compiler and this variable is undefined. Consult the different `d_*ndbm_h_uses_prototypes` variables to get the same information for alternative *ndbm.h* include files.

`d_getaddrinfo`

From *d_getaddrinfo.U*:

This variable conditionally defines the `HAS_GETADDRINFO` symbol, which indicates to the C program that the `getaddrinfo()` function is available.

`d_getcwd`

From *d_getcwd.U*:

This variable conditionally defines the `HAS_GETCWD` symbol, which indicates to the C program

that the `getcwd()` routine is available to get the current working directory.

`d_getespwnam`

From *d_getespwnam.U*:

This variable conditionally defines `HAS_GETESPWNAM` if `getespwnam()` is available to retrieve enhanced (shadow) password entries by name.

`d_getfsstat`

From *d_getfsstat.U*:

This variable conditionally defines the `HAS_GETFSSTAT` symbol, which indicates to the C program that the `getfsstat()` routine is available.

`d_getgrent`

From *d_getgrent.U*:

This variable conditionally defines the `HAS_GETGREN` symbol, which indicates to the C program that the `getgrent()` routine is available for sequential access of the group database.

`d_getgrent_r`

From *d_getgrent_r.U*:

This variable conditionally defines the `HAS_GETGREN` symbol, which indicates to the C program that the `getgrent_r()` routine is available.

`d_getgrgid_r`

From *d_getgrgid_r.U*:

This variable conditionally defines the `HAS_GETGRGID_R` symbol, which indicates to the C program that the `getgrgid_r()` routine is available.

`d_getgrnam_r`

From *d_getgrnam_r.U*:

This variable conditionally defines the `HAS_GETGRNAM_R` symbol, which indicates to the C program that the `getgrnam_r()` routine is available.

`d_getgrps`

From *d_getgrps.U*:

This variable conditionally defines the `HAS_GETGROUPS` symbol, which indicates to the C program that the `getgroups()` routine is available to get the list of process groups.

`d_gethbyaddr`

From *d_gethbyad.U*:

This variable conditionally defines the `HAS_GETHOSTBYADDR` symbol, which indicates to the C program that the `gethostbyaddr()` routine is available to look up hosts by their IP addresses.

`d_gethbyname`

From *d_gethbynm.U*:

This variable conditionally defines the `HAS_GETHOSTBYNAME` symbol, which indicates to the C program that the `gethostbyname()` routine is available to look up host names in some data base or other.

`d_gethent`

From *d_gethent.U*:

This variable conditionally defines `HAS_GETHOSTENT` if `gethostent()` is available to look up host names in some data base or another.

`d_gethostname`

From *d_gethostname.U*:

This variable conditionally defines the `HAS_GETHOSTNAME` symbol, which indicates to the C program that the `gethostname()` routine may be used to derive the host name.

`d_gethostbyaddr_r`

From *d_gethostbyaddr_r.U*:

This variable conditionally defines the `HAS_GETHOSTBYADDR_R` symbol, which indicates to the C program that the `gethostbyaddr_r()` routine is available.

`d_gethostbyname_r`

From *d_gethostbyname_r.U*:

This variable conditionally defines the `HAS_GETHOSTBYNAME_R` symbol, which indicates to the C program that the `gethostbyname_r()` routine is available.

`d_gethostent_r`

From *d_gethostent_r.U*:

This variable conditionally defines the `HAS_GETHOSTENT_R` symbol, which indicates to the C program that the `gethostent_r()` routine is available.

`d_gethostprotos`

From *d_gethostprotos.U*:

This variable conditionally defines the `HAS_GETHOST_PROTOS` symbol, which indicates to the C program that `<netdb.h>` supplies prototypes for the various `gethost*()` functions. See also *netdbtype.U* for probing for various `netdb` types.

`d_getitimer`

From *d_getitimer.U*:

This variable conditionally defines the `HAS_GETITIMER` symbol, which indicates to the C program that the `getitimer()` routine is available.

`d_getlogin`

From *d_getlogin.U*:

This variable conditionally defines the `HAS_GETLOGIN` symbol, which indicates to the C program that the `getlogin()` routine is available to get the login name.

`d_getlogin_r`

From *d_getlogin_r.U*:

This variable conditionally defines the `HAS_GETLOGIN_R` symbol, which indicates to the C program that the `getlogin_r()` routine is available.

`d_getmnt`

From *d_getmnt.U*:

This variable conditionally defines the `HAS_GETMNT` symbol, which indicates to the C program that the `getmnt()` routine is available to retrieve one or more mount info blocks by filename.

`d_getmntent`

From *d_getmntent.U*:

This variable conditionally defines the `HAS_GETMNTENT` symbol, which indicates to the C program that the `getmntent()` routine is available to iterate through mounted files to get their mount info.

`d_getnameinfo`

From *d_getnameinfo.U*:

This variable conditionally defines the `HAS_GETNAMEINFO` symbol, which indicates to the C program that the `getnameinfo()` function is available.

`d_getnbyaddr`

From *d_getnbyad.U*:

This variable conditionally defines the `HAS_GETNETBYADDR` symbol, which indicates to the C program that the `getnetbyaddr()` routine is available to look up networks by their IP addresses.

`d_getnbyname`

From *d_getnbynm.U*:

This variable conditionally defines the `HAS_GETNETBYNAME` symbol, which indicates to the C program that the `getnetbyname()` routine is available to look up networks by their names.

`d_getnetent`

From *d_getnetent.U*:

This variable conditionally defines `HAS_GETNETENT` if `getnetent()` is available to look up network names in some data base or another.

`d_getnetbyaddr_r`

From *d_getnetbyaddr_r.U*:

This variable conditionally defines the `HAS_GETNETBYADDR_R` symbol, which indicates to the C program that the `getnetbyaddr_r()` routine is available.

`d_getnetbyname_r`

From *d_getnetbyname_r.U*:

This variable conditionally defines the `HAS_GETNETBYNAME_R` symbol, which indicates to the C program that the `getnetbyname_r()` routine is available.

`d_getnetent_r`

From *d_getnetent_r.U*:

This variable conditionally defines the `HAS_GETNETENT_R` symbol, which indicates to the C program that the `getnetent_r()` routine is available.

`d_getnetprotos`

From *d_getnetprotos.U*:

This variable conditionally defines the `HAS_GETNET_PROTOS` symbol, which indicates to the C program that `<netdb.h>` supplies prototypes for the various `getnet*()` functions. See also *netdbtype.U* for probing for various `netdb` types.

`d_getpagsz`

From *d_getpagsz.U*:

This variable conditionally defines `HAS_GETPAGESIZE` if `getpagesize()` is available to get the system page size.

`d_getpbyname`

From *d_getprotby.U*:

This variable conditionally defines the `HAS_GETPROTOBYNAME` symbol, which indicates to the C program that the `getprotobyname()` routine is available to look up protocols by their name.

`d_getpbynumber`

From *d_getprotby.U*:

This variable conditionally defines the `HAS_GETPROTOBYNUMBER` symbol, which indicates to the C program that the `getprotobynumber()` routine is available to look up protocols by their number.

`d_getpent`

From *d_getpent.U*:

This variable conditionally defines `HAS_GETPROTOENT` if `getprotoent()` is available to look up protocols in some data base or another.

`d_getpgid`

From *d_getpgid.U*:

This variable conditionally defines the `HAS_GETPGID` symbol, which indicates to the C program that the `getpgid(pid)` function is available to get the process group id.

`d_getpgrp`

From *d_getpgrp.U*:

This variable conditionally defines `HAS_GETPGRP` if `getpgrp()` is available to get the current process group.

`d_getpgrp2`

From *d_getpgrp2.U*:

This variable conditionally defines the `HAS_GETPGRP2` symbol, which indicates to the C program that the `getpgrp2()` (as in *DG/UX*) routine is available to get the current process group.

`d_getppid`

From *d_getppid.U*:

This variable conditionally defines the `HAS_GETPPID` symbol, which indicates to the C program that the `getppid()` routine is available to get the parent process ID.

`d_getprior`

From *d_getprior.U*:

This variable conditionally defines `HAS_GETPRIORITY` if `getpriority()` is available to get a process's priority.

`d_getprotobyname_r`

From *d_getprotobyname_r.U*:

This variable conditionally defines the `HAS_GETPROTOBYNAME_R` symbol, which indicates to the C program that the `getprotobyname_r()` routine is available.

`d_getprotobynumber_r`

From *d_getprotobynumber_r.U*:

This variable conditionally defines the `HAS_GETPROTOBYNUMBER_R` symbol, which indicates to the C program that the `getprotobynumber_r()` routine is available.

`d_getprotoent_r`

From *d_getprotoent_r.U*:

This variable conditionally defines the `HAS_GETPROTOENT_R` symbol, which indicates to the C program that the `getprotoent_r()` routine is available.

`d_getprotoprotos`

From *d_getprotoprotos.U*:

This variable conditionally defines the `HAS_GETPROTO_PROTOS` symbol, which indicates to

the C program that `<netdb.h>` supplies prototypes for the various `getproto*()` functions. See also *netdbtype.U* for probing for various `netdb` types.

`d_getprpwnam`

From *d_getprpwnam.U*:

This variable conditionally defines `HAS_GETPRPWNAM` if `getprpwnam()` is available to retrieve protected (shadow) password entries by name.

`d_getpwent`

From *d_getpwent.U*:

This variable conditionally defines the `HAS_GETPWENT` symbol, which indicates to the C program that the `getpwent()` routine is available for sequential access of the `passwd` database.

`d_getpwent_r`

From *d_getpwent_r.U*:

This variable conditionally defines the `HAS_GETPWENT_R` symbol, which indicates to the C program that the `getpwent_r()` routine is available.

`d_getpwnam_r`

From *d_getpwnam_r.U*:

This variable conditionally defines the `HAS_GETPWNAM_R` symbol, which indicates to the C program that the `getpwnam_r()` routine is available.

`d_getpwuid_r`

From *d_getpwuid_r.U*:

This variable conditionally defines the `HAS_GETPWUID_R` symbol, which indicates to the C program that the `getpwuid_r()` routine is available.

`d_getsbyname`

From *d_getsrvby.U*:

This variable conditionally defines the `HAS_GETSERVBYNAME` symbol, which indicates to the C program that the `getservbyname()` routine is available to look up services by their name.

`d_getsbyport`

From *d_getsrvby.U*:

This variable conditionally defines the `HAS_GETSERVBYPOR` symbol, which indicates to the C program that the `getservbyport()` routine is available to look up services by their port.

`d_getsent`

From *d_getsent.U*:

This variable conditionally defines `HAS_GETSERVENT` if `getservent()` is available to look up network services in some data base or another.

`d_getservbyname_r`

From *d_getservbyname_r.U*:

This variable conditionally defines the `HAS_GETSERVBYNAME_R` symbol, which indicates to the C program that the `getservbyname_r()` routine is available.

`d_getservbyport_r`

From *d_getservbyport_r.U*:

This variable conditionally defines the `HAS_GETSERVBYPOR_R` symbol, which indicates to the C program that the `getservbyport_r()` routine is available.

`d_getservent_r`

From *d_getservent_r.U*:

This variable conditionally defines the `HAS_GETSERVENT_R` symbol, which indicates to the C program that the `getservent_r()` routine is available.

`d_getservprotos`

From *d_getservprotos.U*:

This variable conditionally defines the `HAS_GETSERV_PROTOS` symbol, which indicates to the C program that `<netdb.h>` supplies prototypes for the various `getserv*()` functions. See also *netdbtype.U* for probing for various `netdb` types.

`d_getspnam`

From *d_getspnam.U*:

This variable conditionally defines `HAS_GETSPNAM` if `getspnam()` is available to retrieve SysV shadow password entries by name.

`d_getspnam_r`

From *d_getspnam_r.U*:

This variable conditionally defines the `HAS_GETSPNAM_R` symbol, which indicates to the C program that the `getspnam_r()` routine is available.

`d_gettimeod`

From *d_ftime.U*:

This variable conditionally defines the `HAS_GETTIMEOFDAY` symbol, which indicates that the `gettimeofday()` system call exists (to obtain a sub-second accuracy clock). You should probably include `<sys/resource.h>`.

`d_gmtime64`

From *d_timefuncs64.U*:

This variable conditionally defines the `HAS_GMTIME64` symbol, which indicates to the C program that the `gmtime64()` routine is available.

`d_gmtime_r`

From *d_gmtime_r.U*:

This variable conditionally defines the `HAS_GMTIME_R` symbol, which indicates to the C program that the `gmtime_r()` routine is available.

`d_gnulibc`

From *d_gnulibc.U*:

Defined if we're dealing with the GNU C Library.

`d_grpasswd`

From *i_grp.U*:

This variable conditionally defines `GRPASSWD`, which indicates that `struct group` in `<grp.h>` contains `gr_passwd`.

`d_hasmntopt`

From *d_hasmntopt.U*:

This variable conditionally defines the `HAS_HASMNTOPT` symbol, which indicates to the C program that the `hasmntopt()` routine is available to query the mount options of file systems.

`d_htonl`

From *d_htonl.U*:

This variable conditionally defines `HAS_HTONL` if `htonl()` and its friends are available to do network order byte swapping.

`d_hypot`

From *d_hypot.U*:

This variable conditionally defines `HAS_HYPOT` if `hypot` is available for numerically stable hypotenuse function.

`d_ilogb`

From *d_ilogb.U*:

This variable conditionally defines the `HAS_ILOGB` symbol, which indicates to the C program that the `ilogb()` routine is available for extracting the exponent of double `x` as a signed integer.

`d_ilogbl`

From *d_ilogbl.U*:

This variable conditionally defines the `HAS_ILOGBL` symbol, which indicates to the C program that the `ilogbl()` routine is available for extracting the exponent of long double `x` as a signed integer. If `scalbnl` is also present we can emulate `frexpl`.

`d_inc_version_list`

From *inc_version_list.U*:

This variable conditionally defines `PERL_INC_VERSION_LIST`. It is set to `undef` when `PERL_INC_VERSION_LIST` is empty.

`d_index`

From *d_strchr.U*:

This variable conditionally defines `HAS_INDEX` if `index()` and `rindex()` are available for string searching.

`d_inetaton`

From *d_inetaton.U*:

This variable conditionally defines the `HAS_INET_ATON` symbol, which indicates to the C program that the `inet_aton()` function is available to parse IP address dotted-quad strings.

`d_inetntop`

From *d_inetntop.U*:

This variable conditionally defines the `HAS_INETNTOP` symbol, which indicates to the C program that the `inet_ntop()` function is available.

`d_inetpton`

From *d_inetpton.U*:

This variable conditionally defines the `HAS_INETPTON` symbol, which indicates to the C program that the `inet_pton()` function is available.

`d_int64_t`

From *d_int64_t.U*:

This symbol will be defined if the C compiler supports `int64_t`.

`d_ip_mreq`

From *d_socket.U*:

This variable conditionally defines the `HAS_IP_MREQ` symbol, which indicates the availability

of a struct `ip_mreq`.

`d_ip_mreq_source`

From *d_socket.U*:

This variable conditionally defines the `HAS_IP_MREQ_SOURCE` symbol, which indicates the availability of a struct `ip_mreq_source`.

`d_ipv6_mreq`

From *d_socket.U*:

This variable conditionally defines the `HAS_IPV6_MREQ` symbol, which indicates the availability of a struct `ipv6_mreq`.

`d_ipv6_mreq_source`

From *d_socket.U*:

This variable conditionally defines the `HAS_IPV6_MREQ_SOURCE` symbol, which indicates the availability of a struct `ipv6_mreq_source`.

`d_isascii`

From *d_isascii.U*:

This variable conditionally defines the `HAS_ISASCII` constant, which indicates to the C program that `isascii()` is available.

`d_isblank`

From *d_isblank.U*:

This variable conditionally defines the `HAS_ISBLANK` constant, which indicates to the C program that `isblank()` is available.

`d_isfinite`

From *d_isfinite.U*:

This variable conditionally defines the `HAS_ISFINITE` symbol, which indicates to the C program that the `isfinite()` routine is available.

`d_isfinitel`

From *d_isfinitel.U*:

This variable conditionally defines the `HAS_ISFINITEL` symbol, which indicates to the C program that the `isfinitel()` routine is available.

`d_isinf`

From *d_isinf.U*:

This variable conditionally defines the `HAS_ISINF` symbol, which indicates to the C program that the `isinf()` routine is available.

`d_isinfl`

From *d_isinfl.U*:

This variable conditionally defines the `HAS_ISINFL` symbol, which indicates to the C program that the `isinfl()` routine is available.

`d_isless`

From *d_isless.U*:

This variable conditionally defines the `HAS_ISLESS` symbol, which indicates to the C program that the `isless()` routine is available.

`d_isnan`

From *d_isnan.U*:

This variable conditionally defines the `HAS_ISNAN` symbol, which indicates to the C program that the `isnan()` routine is available.

`d_isnanl`

From *d_isnanl.U*:

This variable conditionally defines the `HAS_ISNANL` symbol, which indicates to the C program that the `isnanl()` routine is available.

`d_isnormal`

From *d_isnormal.U*:

This variable conditionally defines the `HAS_ISNORMAL` symbol, which indicates to the C program that the `isnormal()` routine is available.

`d_j0`

From *d_j0.U*:

This variable conditionally defines the `HAS_J0` symbol, which indicates to the C program that the `j0()` routine is available.

`d_j0l`

From *d_j0l.U*:

This variable conditionally defines the `HAS_J0L` symbol, which indicates to the C program that the `j0l()` routine is available.

`d_killpg`

From *d_killpg.U*:

This variable conditionally defines the `HAS_KILLPG` symbol, which indicates to the C program that the `killpg()` routine is available to kill process groups.

`d_lc_monetary_2008`

From *d_lc_monetary_2008.U*:

This variable conditionally defines `HAS_LC_MONETARY_2008` if `libc` has the international currency locale rules from `POSIX 1003.1-2008`.

`d_lchown`

From *d_lchown.U*:

This variable conditionally defines the `HAS_LCHOWN` symbol, which indicates to the C program that the `lchown()` routine is available to operate on a symbolic link (instead of following the link).

`d_ldbl_dig`

From *d_ldbl_dig.U*:

This variable conditionally defines `d_ldbl_dig` if this system's header files provide `LDBL_DIG`, which is the number of significant digits in a long double precision number.

`d_ldexpl`

From *d_longdbl.U*:

This variable conditionally defines the `HAS_LDEXPL` symbol, which indicates to the C program that the `ldexpl()` routine is available.

`d_lgamma`

From *d_lgamma.U*:

This variable conditionally defines the `HAS_LGAMMA` symbol, which indicates to the C program that the `lgamma()` routine is available for the log gamma function. See also `d_tgamma` and `d_lgamma_r`.

`d_lgamma_r`

From *d_lgamma_r.U*:

This variable conditionally defines the `HAS_LGAMMA_R` symbol, which indicates to the C program that the `lgamma_r()` routine is available for the log gamma function, without using the global `signgam` variable.

`d_libm_lib_version`

From *d_libm_lib_version.U*:

This variable conditionally defines the `LIBM_LIB_VERSION` symbol, which indicates to the C program that *math.h* defines `_LIB_VERSION` being available in `libm`

`d_libname_unique`

From *so.U*:

This variable is defined if the target system insists on unique basenames for shared library files. This is currently true on Android, false everywhere else we know of. Defaults to `undef`.

`d_link`

From *d_link.U*:

This variable conditionally defines `HAS_LINK` if `link()` is available to create hard links.

`d_llrint`

From *d_llrint.U*:

This variable conditionally defines the `HAS_LLRINT` symbol, which indicates to the C program that the `llrint()` routine is available to return the long long value closest to a double (according to the current rounding mode).

`d_llrintl`

From *d_llrintl.U*:

This variable conditionally defines the `HAS_LLRINTL` symbol, which indicates to the C program that the `llrintl()` routine is available to return the long long value closest to a long double (according to the current rounding mode).

`d_llround`

From *d_llround.U*:

This variable conditionally defines the `HAS_LLROUND` symbol, which indicates to the C program that the `llround()` routine is available to return the long long value nearest to `x`.

`d_llroundl`

From *d_llroundl.U*:

This variable conditionally defines the `HAS_LLROUNDL` symbol, which indicates to the C program that the `llroundl()` routine is available to return the long long value nearest to `x` away from zero.

`d_localtime64`

From *d_timefuncs64.U*:

This variable conditionally defines the `HAS_LOCALTIME64` symbol, which indicates to the C program that the `localtime64()` routine is available.

`d_localtime_r`

From *d_localtime_r.U*:

This variable conditionally defines the `HAS_LOCALTIME_R` symbol, which indicates to the C program that the `localtime_r()` routine is available.

`d_localtime_r_needs_tzset`

From *d_localtime_r.U*:

This variable conditionally defines the `LOCALTIME_R_NEEDS_TZSET` symbol, which makes us call `tzset` before `localtime_r()`

`d_loconv`

From *d_loconv.U*:

This variable conditionally defines `HAS_LOCALECONV` if `localeconv()` is available for numeric and monetary formatting conventions.

`d_lockf`

From *d_lockf.U*:

This variable conditionally defines `HAS_LOCKF` if `lockf()` is available to do file locking.

`d_log1p`

From *d_log1p.U*:

This variable conditionally defines the `HAS_LOG1P` symbol, which indicates to the C program that the `log1p()` routine is available to compute $\log(1 + x)$ for values of x close to zero.

`d_log2`

From *d_log2.U*:

This variable conditionally defines the `HAS_LOG2` symbol, which indicates to the C program that the `log2()` routine is available to compute log base two.

`d_logb`

From *d_logb.U*:

This variable conditionally defines the `HAS_LOGB` symbol, which indicates to the C program that the `logb()` routine is available to extract the exponent of x .

`d_longdbl`

From *d_longdbl.U*:

This variable conditionally defines `HAS_LONG_DOUBLE` if the long double type is supported.

`d_longlong`

From *d_longlong.U*:

This variable conditionally defines `HAS_LONG_LONG` if the long long type is supported.

`d_lrint`

From *d_lrint.U*:

This variable conditionally defines the `HAS_LRINT` symbol, which indicates to the C program that the `lrint()` routine is available to return the integral value closest to a double (according to the current rounding mode).

`d_lrintl`

From *d_lrintl.U*:

This variable conditionally defines the `HAS_LRINTL` symbol, which indicates to the C program that the `lrintl()` routine is available to return the integral value closest to a long double (according to the current rounding mode).

`d_lround`

From *d_lround.U*:

This variable conditionally defines the `HAS_LROUND` symbol, which indicates to the C program that the `lround()` routine is available to return the integral value nearest to `x`.

`d_lroundl`

From *d_lroundl.U*:

This variable conditionally defines the `HAS_LROUNDL` symbol, which indicates to the C program that the `lroundl()` routine is available to return the integral value nearest to `x` away from zero.

`d_lseekproto`

From *d_lseekproto.U*:

This variable conditionally defines the `HAS_LSEEK_PROTO` symbol, which indicates to the C program that the system provides a prototype for the `lseek()` function. Otherwise, it is up to the program to supply one.

`d_lstat`

From *d_lstat.U*:

This variable conditionally defines `HAS_LSTAT` if `lstat()` is available to do file stats on symbolic links.

`d_madvise`

From *d_madvise.U*:

This variable conditionally defines `HAS_MADVISE` if `madvise()` is available to map a file into memory.

`d_malloc_good_size`

From *d_malloc_size.U*:

This symbol, if defined, indicates that the `malloc_good_size` routine is available for use.

`d_malloc_size`

From *d_malloc_size.U*:

This symbol, if defined, indicates that the `malloc_size` routine is available for use.

`d_mblen`

From *d_mblen.U*:

This variable conditionally defines the `HAS_MBLEN` symbol, which indicates to the C program that the `mblen()` routine is available to find the number of bytes in a multibyte character.

`d_mbstowcs`

From *d_mbstowcs.U*:

This variable conditionally defines the `HAS_MBSTOWCS` symbol, which indicates to the C program that the `mbstowcs()` routine is available to convert a multibyte string into a wide character string.

`d_mbtowc`

From *d_mbtowc.U*:

This variable conditionally defines the `HAS_MBTOWC` symbol, which indicates to the C program that the `mbtowc()` routine is available to convert multibyte to a wide character.

`d_memchr`

From *d_memchr.U*:

This variable conditionally defines the `HAS_MEMCHR` symbol, which indicates to the C program that the `memchr()` routine is available to locate characters within a C string.

`d_memcmp`

From *d_memcmp.U*:

This variable conditionally defines the `HAS_MEMCMP` symbol, which indicates to the C program that the `memcmp()` routine is available to compare blocks of memory.

`d_memcpy`

From *d_memcpy.U*:

This variable conditionally defines the `HAS_MEMCPY` symbol, which indicates to the C program that the `memcpy()` routine is available to copy blocks of memory.

`d_memmem`

From *d_memmem.U*:

This variable conditionally defines the `HAS_MEMMEM` symbol, which indicates to the C program that the `memmem()` routine is available to return a pointer to the start of the first occurrence of a substring in a memory area (or `NULL` if not found).

`d_memmove`

From *d_memmove.U*:

This variable conditionally defines the `HAS_MEMMOVE` symbol, which indicates to the C program that the `memmove()` routine is available to copy potentially overlapping blocks of memory.

`d_memset`

From *d_memset.U*:

This variable conditionally defines the `HAS_MEMSET` symbol, which indicates to the C program that the `memset()` routine is available to set blocks of memory.

`d_mkdir`

From *d_mkdir.U*:

This variable conditionally defines the `HAS_MKDIR` symbol, which indicates to the C program that the `mkdir()` routine is available to create *directories*.

`d_mkdtemp`

From *d_mkdtemp.U*:

This variable conditionally defines the `HAS_MKDTEMP` symbol, which indicates to the C program that the `mkdtemp()` routine is available to exclusively create a uniquely named temporary directory.

`d_mkfifo`

From *d_mkfifo.U*:

This variable conditionally defines the `HAS_MKFIFO` symbol, which indicates to the C program that the `mkfifo()` routine is available.

`d_mkstemp`

From *d_mkstemp.U*:

This variable conditionally defines the `HAS_MKSTEMP` symbol, which indicates to the C program that the `mkstemp()` routine is available to exclusively create and open a uniquely named temporary file.

`d_mkstemps`

From *d_mkstemps.U*:

This variable conditionally defines the `HAS_MKSTEMPS` symbol, which indicates to the C program that the `mkstemps()` routine is available to exclusively create and open a uniquely named (with a suffix) temporary file.

`d_mktime`

From *d_mktime.U*:

This variable conditionally defines the `HAS_MKTIME` symbol, which indicates to the C program that the `mktime()` routine is available.

`d_mktime64`

From *d_timefuncs64.U*:

This variable conditionally defines the `HAS_MKTIME64` symbol, which indicates to the C program that the `mktime64 ()` routine is available.

`d_mmap`

From *d_mmap.U*:

This variable conditionally defines `HAS_MMAP` if `mmap()` is available to map a file into memory.

`d_modfl`

From *d_modfl.U*:

This variable conditionally defines the `HAS_MODFL` symbol, which indicates to the C program that the `modfl()` routine is available.

`d_modflproto`

From *d_modfl.U*:

This symbol, if defined, indicates that the system provides a prototype for the `modfl()` function. Otherwise, it is up to the program to supply one. C99 says it should be `long double modfl(long double, long double *)`;

`d_mprotect`

From *d_mprotect.U*:

This variable conditionally defines `HAS_MPROTECT` if `mprotect()` is available to modify the access protection of a memory mapped file.

`d_msg`

From *d_msg.U*:

This variable conditionally defines the `HAS_MSG` symbol, which indicates that the entire `msg*(2)` library is present.

`d_msg_ctrunc`

From *d_socket.U*:

This variable conditionally defines the `HAS_MSG_CTRUNC` symbol, which indicates that the `MSG_CTRUNC` is available. `#ifdef` is not enough because it may be an enum, glibc has been known to do this.

`d_msg_dontroute`

From *d_socket.U*:

This variable conditionally defines the `HAS_MSG_DONTROUTE` symbol, which indicates that the `MSG_DONTROUTE` is available. `#ifdef` is not enough because it may be an enum, glibc has been known to do this.

`d_msg_oob`

From *d_socket.U*:

This variable conditionally defines the `HAS_MSG_OOB` symbol, which indicates that the `MSG_OOB` is available. `#ifdef` is not enough because it may be an enum, glibc has been known to do this.

`d_msg_peek`

From *d_socket.U*:

This variable conditionally defines the `HAS_MSG_PEEK` symbol, which indicates that the `MSG_PEEK` is available. `#ifdef` is not enough because it may be an enum, glibc has been known to do this.

`d_msg_proxy`

From *d_socket.U*:

This variable conditionally defines the `HAS_MSG_PROXY` symbol, which indicates that the `MSG_PROXY` is available. `#ifdef` is not enough because it may be an enum, glibc has been known to do this.

`d_msgctl`

From *d_msgctl.U*:

This variable conditionally defines the `HAS_MSGCTL` symbol, which indicates to the C program that the `msgctl()` routine is available.

`d_msgget`

From *d_msgget.U*:

This variable conditionally defines the `HAS_MSGGET` symbol, which indicates to the C program that the `msgget()` routine is available.

`d_msghdr_s`

From *d_msghdr_s.U*:

This variable conditionally defines the `HAS_STRUCT_MSGHDR` symbol, which indicates that the `struct msghdr` is supported.

`d_msgrcv`

From *d_msgrcv.U*:

This variable conditionally defines the `HAS_MSGRCV` symbol, which indicates to the C program that the `msgrcv()` routine is available.

`d_msgsnd`

From *d_msgsnd.U*:

This variable conditionally defines the `HAS_MSGSND` symbol, which indicates to the C program that the `msgsnd()` routine is available.

`d_msync`

From *d_msync.U*:

This variable conditionally defines `HAS_MSYNC` if `msync()` is available to synchronize a mapped file.

`d_munmap`

From *d_munmap.U*:

This variable conditionally defines `HAS_MUNMAP` if `munmap()` is available to unmap a region mapped by `mmap()`.

`d_mymalloc`

From *mallocsrc.U*:

This variable conditionally defines `MYMALLOC` in case other parts of the source want to take special action if `MYMALLOC` is used. This may include different sorts of profiling or error detection.

`d_nan`

From *d_nan.U*:

This variable conditionally defines `HAS_NAN` if `nan()` is available to generate NaN.

`d_ndbm`

From *i_ndbm.U*:

This variable conditionally defines the `HAS_NDBM` symbol, which indicates that both the *ndbm.h* include file and an appropriate ndbm library exist. Consult the different *i_*ndbm* variables to find out the actual include location. Sometimes, a system has the header file but not the library. This variable will only be set if the system has both.

`d_ndbm_h_uses_prototypes`

From *i_ndbm.U*:

This variable conditionally defines the `NDBM_H_USES_PROTOTYPES` symbol, which indicates that the *ndbm.h* include file uses real ANSI C prototypes instead of K&R style function declarations. K&R style declarations are unsupported in C++, so the include file requires special handling when using a C++ compiler and this variable is undefined. Consult the different *d_*ndbm_h_uses_prototypes* variables to get the same information for alternative *ndbm.h* include files.

`d_nearbyint`

From *d_nearbyint.U*:

This variable conditionally defines `HAS_NEARBYINT` if `nearbyint()` is available to return the integral value closest to (according to the current rounding mode) to *x*.

`d_newlocale`

From *d_newlocale.U*:

This variable conditionally defines the `HAS_NEWLOCALE` symbol, which indicates to the C program that the `newlocale()` routine is available to return a new locale object or modify an existing locale object.

`d_nextafter`

From *d_nextafter.U*:

This variable conditionally defines `HAS_NEXTAFTER` if `nextafter()` is available to return the next machine representable double from *x* in direction *y*.

`d_nexttoward`

From *d_nexttoward.U*:

This variable conditionally defines `HAS_NEXTTOWARD` if `nexttoward()` is available to return the next machine representable long double from *x* in direction *y*.

`d_nice`

From *d_nice.U*:

This variable conditionally defines the `HAS_NICE` symbol, which indicates to the C program that the `nice()` routine is available.

`d_nl_langinfo`

From *d_nl_langinfo.U*:

This variable conditionally defines the `HAS_NL_LANGINFO` symbol, which indicates to the C program that the `nl_langinfo()` routine is available.

`d_nv_preserves_uv`

From *perlsv.U*:

This variable indicates whether a variable of type `nvtype` can preserve all the bits a variable of type `uvtype`.

`d_nv_zero_is_allbits_zero`

From *perlsv.U*:

This variable indicates whether a variable of type `nvtype` stores 0.0 in memory as all bits zero.

`d_off64_t`

From *d_off64_t.U*:

This symbol will be defined if the C compiler supports `off64_t`.

`d_old_pthread_create_joinable`

From *d_pthreadtrj.U*:

This variable conditionally defines `pthread_create_joinable`. undef if *pthread.h* defines `PTHREAD_CREATE_JOINABLE`.

`d_oldpthreads`

From *usethreads.U*:

This variable conditionally defines the `OLD_PTHREADS_API` symbol, and indicates that Perl should be built to use the old draft `POSIX` threads API. This is only potentially meaningful if `usethreads` is set.

`d_oldsocket`

From *d_socket.U*:

This variable conditionally defines the `OLD_SOCKET` symbol, which indicates that the BSD socket interface is based on 4.1c and not 4.2.

`d_open3`

From *d_open3.U*:

This variable conditionally defines the `HAS_OPEN3` manifest constant, which indicates to the C program that the 3 argument version of the `open(2)` function is available.

`d_pathconf`

From *d_pathconf.U*:

This variable conditionally defines the `HAS_PATHCONF` symbol, which indicates to the C program that the `pathconf()` routine is available to determine file-system related limits and options associated with a given filename.

`d_pause`

From *d_pause.U*:

This variable conditionally defines the `HAS_PAUSE` symbol, which indicates to the C program that the `pause()` routine is available to suspend a process until a signal is received.

`d_perl_otherlibdirs`

From *otherlibdirs.U*:

This variable conditionally defines `PERL_OTHERLIBDIRS`, which contains a colon-separated

set of paths for the perl binary to include in @INC. See also otherlibdirs.

`d_phostname`

From *d_gethname.U*:

This variable conditionally defines the `HAS_PHOSTNAME` symbol, which contains the shell command which, when fed to `popen()`, may be used to derive the host name.

`d_pipe`

From *d_pipe.U*:

This variable conditionally defines the `HAS_PIPE` symbol, which indicates to the C program that the `pipe()` routine is available to create an inter-process channel.

`d_poll`

From *d_poll.U*:

This variable conditionally defines the `HAS_POLL` symbol, which indicates to the C program that the `poll()` routine is available to poll active file descriptors.

`d_portable`

From *d_portable.U*:

This variable conditionally defines the `PORTABLE` symbol, which indicates to the C program that it should not assume that it is running on the machine it was compiled on.

`d_prctl`

From *d_prctl.U*:

This variable conditionally defines the `HAS_PRCTL` symbol, which indicates to the C program that the `prctl()` routine is available. Note that there are at least two `prctl` variants: Linux and Irix. While they are somewhat similar, they are incompatible.

`d_prctl_set_name`

From *d_prctl.U*:

This variable conditionally defines the `HAS_PRCTL_SET_NAME` symbol, which indicates to the C program that the `prctl()` routine supports the `PR_SET_NAME` option.

`d_PRId64`

From *quadfio.U*:

This variable conditionally defines the `PERL_PRId64` symbol, which indicates that `stdio` has a symbol to print 64-bit decimal numbers.

`d_PRIeldbl`

From *longdblfiio.U*:

This variable conditionally defines the `PERL_PRIeldbl` symbol, which indicates that `stdio` has a symbol to print long doubles.

`d_PRIEUldbl`

From *longdblfiio.U*:

This variable conditionally defines the `PERL_PRIldbl` symbol, which indicates that `stdio` has a symbol to print long doubles. The `U` in the name is to separate this from `d_PRIeldbl` so that even case-blind systems can see the difference.

`d_PRIfldbl`

From *longdblfiio.U*:

This variable conditionally defines the `PERL_PRIldbl` symbol, which indicates that `stdio` has a symbol to print long doubles.

d_PRIFUldbl

From *longdblfiio.U*:

This variable conditionally defines the PERL_PRIfldbl symbol, which indicates that stdio has a symbol to print long doubles. The `U` in the name is to separate this from d_PRIfldbl so that even case-blind systems can see the difference.

d_PRIgldbl

From *longdblfiio.U*:

This variable conditionally defines the PERL_PRIfldbl symbol, which indicates that stdio has a symbol to print long doubles.

d_PRIGUldbl

From *longdblfiio.U*:

This variable conditionally defines the PERL_PRIfldbl symbol, which indicates that stdio has a symbol to print long doubles. The `U` in the name is to separate this from d_PRIgldbl so that even case-blind systems can see the difference.

d_PRIi64

From *quadfiio.U*:

This variable conditionally defines the PERL_PRIi64 symbol, which indicates that stdio has a symbol to print 64-bit decimal numbers.

d_printf_format_null

From *d_attribut.U*:

This variable conditionally defines PRINTF_FORMAT_NULL_OK, which indicates the C compiler allows printf-like formats to be null.

d_PRIo64

From *quadfiio.U*:

This variable conditionally defines the PERL_PRIo64 symbol, which indicates that stdio has a symbol to print 64-bit octal numbers.

d_PRIu64

From *quadfiio.U*:

This variable conditionally defines the PERL_PRIu64 symbol, which indicates that stdio has a symbol to print 64-bit unsigned decimal numbers.

d_PRIx64

From *quadfiio.U*:

This variable conditionally defines the PERL_PRIx64 symbol, which indicates that stdio has a symbol to print 64-bit hexadecimal numbers.

d_PRIXU64

From *quadfiio.U*:

This variable conditionally defines the PERL_PRIXU64 symbol, which indicates that stdio has a symbol to print 64-bit hExADECImaI numbers. The `U` in the name is to separate this from d_PRIx64 so that even case-blind systems can see the difference.

d_proclselfexe

From *d_proclselfexe.U*:

Defined if \$proclselfexe is symlink to the absolute pathname of the executing program.

d_pseudofork

From *d_vfork.U*:

This variable conditionally defines the `HAS_PSEUDOFORK` symbol, which indicates that an emulation of the `fork` routine is available.

`d_pthread_atfork`

From *d_pthread_atfork.U*:

This variable conditionally defines the `HAS_PTHREAD_ATFORK` symbol, which indicates to the C program that the `pthread_atfork()` routine is available.

`d_pthread_attr_setscope`

From *d_pthread_attr_ss.U*:

This variable conditionally defines `HAS_PTHREAD_ATTR_SETSCOPE` if `pthread_attr_setscope()` is available to set the contention scope attribute of a thread attribute object.

`d_pthread_yield`

From *d_pthread_y.U*:

This variable conditionally defines the `HAS_PTHREAD_YIELD` symbol if the `pthread_yield` routine is available to yield the execution of the current thread.

`d_ptrdiff_t`

From *d_ptrdiff_t.U*:

This symbol will be defined if the C compiler supports `ptrdiff_t`.

`d_pwage`

From *i_pwd.U*:

This variable conditionally defines `PWAGE`, which indicates that struct `passwd` contains `pw_age`.

`d_pwchange`

From *i_pwd.U*:

This variable conditionally defines `PWCHANGE`, which indicates that struct `passwd` contains `pw_change`.

`d_pwclass`

From *i_pwd.U*:

This variable conditionally defines `PWCLASS`, which indicates that struct `passwd` contains `pw_class`.

`d_pwcomment`

From *i_pwd.U*:

This variable conditionally defines `PWCOMMENT`, which indicates that struct `passwd` contains `pw_comment`.

`d_pwexpire`

From *i_pwd.U*:

This variable conditionally defines `PWEXPIRE`, which indicates that struct `passwd` contains `pw_expire`.

`d_pwgecos`

From *i_pwd.U*:

This variable conditionally defines `PWGECOS`, which indicates that struct `passwd` contains

`d_pwpasswd`

From *i_pwd.U*:

This variable conditionally defines `PWPASSWD`, which indicates that struct `passwd` contains `pw_passwd`.

`d_pwquota`

From *i_pwd.U*:

This variable conditionally defines `PWQUOTA`, which indicates that struct `passwd` contains `pw_quota`.

`d_qgcvt`

From *d_qgcvt.U*:

This variable conditionally defines the `HAS_QGCVT` symbol, which indicates to the C program that the `qgcvt()` routine is available.

`d_quad`

From *quadtype.U*:

This variable, if defined, tells that there's a 64-bit integer type, `quadtype`.

`d_random_r`

From *d_random_r.U*:

This variable conditionally defines the `HAS_RANDOM_R` symbol, which indicates to the C program that the `random_r()` routine is available.

`d_re_comp`

From *d_regcmp.U*:

This variable conditionally defines the `HAS_RECOMP` symbol, which indicates to the C program that the `re_comp()` routine is available for regular pattern matching (usually on BSD). If so, it is likely that `re_exec()` exists.

`d_readdir`

From *d_readdir.U*:

This variable conditionally defines `HAS_READDIR` if `readdir()` is available to read directory entries.

`d_readdir64_r`

From *d_readdir64_r.U*:

This variable conditionally defines the `HAS_READDIR64_R` symbol, which indicates to the C program that the `readdir64_r()` routine is available.

`d_readdir_r`

From *d_readdir_r.U*:

This variable conditionally defines the `HAS_READDIR_R` symbol, which indicates to the C program that the `readdir_r()` routine is available.

`d_readlink`

From *d_readlink.U*:

This variable conditionally defines the `HAS_READLINK` symbol, which indicates to the C program that the `readlink()` routine is available to read the value of a symbolic link.

`d_readv`

From *d_readv.U*:

This variable conditionally defines the `HAS_READV` symbol, which indicates to the C program that the `readv()` routine is available.

`d_recvmsg`

From *d_recvmsg.U*:

This variable conditionally defines the `HAS_RECVMSG` symbol, which indicates to the C program that the `recvmsg()` routine is available.

`d_regcmp`

From *d_regcmp.U*:

This variable conditionally defines the `HAS_REGCMP` symbol, which indicates to the C program that the `regcmp()` routine is available for regular pattern matching (usually on System V).

`d_regcomp`

From *d_regcomp.U*:

This variable conditionally defines the `HAS_REGCOMP` symbol, which indicates to the C program that the `regcomp()` routine is available for regular pattern matching (usually on *POSIX.2* conforming systems).

`d_remainder`

From *d_remainder.U*:

This variable conditionally defines the `HAS_REMAINDER` symbol, which indicates to the C program that the `remainder()` routine is available.

`d_remquo`

From *d_remquo.U*:

This variable conditionally defines the `HAS_REMQUO` symbol, which indicates to the C program that the `remquo()` routine is available.

`d_rename`

From *d_rename.U*:

This variable conditionally defines the `HAS_RENAME` symbol, which indicates to the C program that the `rename()` routine is available to rename files.

`d_rewinddir`

From *d_readdir.U*:

This variable conditionally defines `HAS_REWINDDIR` if `rewinddir()` is available.

`d_rint`

From *d_rint.U*:

This variable conditionally defines the `HAS_RINT` symbol, which indicates to the C program that the `rint()` routine is available.

`d_rmdir`

From *d_rmdir.U*:

This variable conditionally defines `HAS_RMDIR` if `rmdir()` is available to remove directories.

`d_round`

From *d_round.U*:

This variable conditionally defines the `HAS_ROUND` symbol, which indicates to the C program that the `round()` routine is available.

`d_safebcpy`

From *d_safebcopy.U*:

This variable conditionally defines the `HAS_SAFE_BCOPY` symbol if the `bcopy()` routine can do overlapping copies. Normally, you should probably use `memmove()`.

`d_safemcpy`

From *d_safemcpy.U*:

This variable conditionally defines the `HAS_SAFE_MEMCPY` symbol if the `memcpy()` routine can do overlapping copies. For overlapping copies, `memmove()` should be used, if available.

`d_sanemcmp`

From *d_sanemcmp.U*:

This variable conditionally defines the `HAS_SANE_MEMCMP` symbol if the `memcpy()` routine is available and can be used to compare relative magnitudes of chars with their high bits set.

`d_sbrkproto`

From *d_sbrkproto.U*:

This variable conditionally defines the `HAS_SBRK_PROTO` symbol, which indicates to the C program that the system provides a prototype for the `sbrk()` function. Otherwise, it is up to the program to supply one.

`d_scalbn`

From *d_scalbn.U*:

This variable conditionally defines the `HAS_SCALBN` symbol, which indicates to the C program that the `scalbn()` routine is available.

`d_scalbnl`

From *d_scalbnl.U*:

This variable conditionally defines the `HAS_SCALBNL` symbol, which indicates to the C program that the `scalbnl()` routine is available. If `ilogbl` is also present we can emulate `frexpl`.

`d_sched_yield`

From *d_pthread_y.U*:

This variable conditionally defines the `HAS_SCHED_YIELD` symbol if the `sched_yield` routine is available to yield the execution of the current thread.

`d_scm_rights`

From *d_socket.U*:

This variable conditionally defines the `HAS_SCM_RIGHTS` symbol, which indicates that the `SCM_RIGHTS` is available. `#ifdef` is not enough because it may be an enum, glibc has been known to do this.

`d_SCNfldbl`

From *longdblflto.U*:

This variable conditionally defines the `PERL_PRIfldbl` symbol, which indicates that `stdio` has a symbol to scan long doubles.

`d_seekdir`

From *d_readdir.U*:

This variable conditionally defines `HAS_SEEKDIR` if `seekdir()` is available.

`d_select`

From *d_select.U*:

This variable conditionally defines `HAS_SELECT` if `select()` is available to select active file descriptors. A `<sys/time.h>` inclusion may be necessary for the timeout field.

`d_sem`

From *d_sem.U*:

This variable conditionally defines the `HAS_SEM` symbol, which indicates that the entire `sem*(2)` library is present.

`d_semctl`

From *d_semctl.U*:

This variable conditionally defines the `HAS_SEMCTL` symbol, which indicates to the C program that the `semctl()` routine is available.

`d_semctl_semid_ds`

From *d_union_semun.U*:

This variable conditionally defines `USE_SEMCTL_SEMID_DS`, which indicates that struct `semid_ds *` is to be used for `semctl IPC_STAT`.

`d_semctl_semun`

From *d_union_semun.U*:

This variable conditionally defines `USE_SEMCTL_SEMUN`, which indicates that union `semun` is to be used for `semctl IPC_STAT`.

`d_semget`

From *d_semget.U*:

This variable conditionally defines the `HAS_SEMGET` symbol, which indicates to the C program that the `semget()` routine is available.

`d_semop`

From *d_semop.U*:

This variable conditionally defines the `HAS_SEMOP` symbol, which indicates to the C program that the `semop()` routine is available.

`d_sendmsg`

From *d_sendmsg.U*:

This variable conditionally defines the `HAS_SENDMSG` symbol, which indicates to the C program that the `sendmsg()` routine is available.

`d_setegid`

From *d_setegid.U*:

This variable conditionally defines the `HAS_SETEGID` symbol, which indicates to the C program that the `setegid()` routine is available to change the effective gid of the current program.

`d_seteuid`

From *d_seteuid.U*:

This variable conditionally defines the `HAS_SETEUID` symbol, which indicates to the C program that the `seteuid()` routine is available to change the effective uid of the current program.

`d_setgrent`

From *d_setgrent.U*:

This variable conditionally defines the `HAS_SETGRENT` symbol, which indicates to the C

program that the `setgrent()` routine is available for initializing sequential access to the group database.

`d_setgrent_r`

From *d_setgrent_r.U*:

This variable conditionally defines the `HAS_SETGRENT_R` symbol, which indicates to the C program that the `setgrent_r()` routine is available.

`d_setgrps`

From *d_setgrps.U*:

This variable conditionally defines the `HAS_SETGROUPS` symbol, which indicates to the C program that the `setgroups()` routine is available to set the list of process groups.

`d_sethent`

From *d_sethent.U*:

This variable conditionally defines `HAS_SETHOSTENT` if `sethostent()` is available.

`d_sethostent_r`

From *d_sethostent_r.U*:

This variable conditionally defines the `HAS_SETHOSTENT_R` symbol, which indicates to the C program that the `sethostent_r()` routine is available.

`d_setitimer`

From *d_setitimer.U*:

This variable conditionally defines the `HAS_SETITIMER` symbol, which indicates to the C program that the `setitimer()` routine is available.

`d_setlinebuf`

From *d_setlnbuf.U*:

This variable conditionally defines the `HAS_SETLINEBUF` symbol, which indicates to the C program that the `setlinebuf()` routine is available to change `stderr` or `stdout` from block-buffered or unbuffered to a line-buffered mode.

`d_setlocale`

From *d_setlocale.U*:

This variable conditionally defines `HAS_SETLOCALE` if `setlocale()` is available to handle locale-specific ctype implementations.

`d_setlocale_r`

From *d_setlocale_r.U*:

This variable conditionally defines the `HAS_SETLOCALE_R` symbol, which indicates to the C program that the `setlocale_r()` routine is available.

`d_setnetent`

From *d_setnetent.U*:

This variable conditionally defines `HAS_SETNETENT` if `setnetent()` is available.

`d_setnetent_r`

From *d_setnetent_r.U*:

This variable conditionally defines the `HAS_SETNETENT_R` symbol, which indicates to the C program that the `setnetent_r()` routine is available.

`d_setpent`

From *d_setpent.U*:

This variable conditionally defines `HAS_SETPROTOENT` if `setprotoent()` is available.

`d_setpgid`

From *d_setpgid.U*:

This variable conditionally defines the `HAS_SETPGID` symbol if the `setpgid(pid, gpid)` function is available to set process group ID.

`d_setpgrp`

From *d_setpgrp.U*:

This variable conditionally defines `HAS_SETPGRP` if `setpgrp()` is available to set the current process group.

`d_setpgrp2`

From *d_setpgrp2.U*:

This variable conditionally defines the `HAS_SETPGRP2` symbol, which indicates to the C program that the `setpgrp2()` (as in *DG/ux*) routine is available to set the current process group.

`d_setprior`

From *d_setprior.U*:

This variable conditionally defines `HAS_SETPRIORITY` if `setpriority()` is available to set a process's priority.

`d_setproctitle`

From *d_setproctitle.U*:

This variable conditionally defines the `HAS_SETPROCTITLE` symbol, which indicates to the C program that the `setproctitle()` routine is available.

`d_setprotoent_r`

From *d_setprotoent_r.U*:

This variable conditionally defines the `HAS_SETPROTOENT_R` symbol, which indicates to the C program that the `setprotoent_r()` routine is available.

`d_setpwent`

From *d_setpwent.U*:

This variable conditionally defines the `HAS_SETPWENT` symbol, which indicates to the C program that the `setpwent()` routine is available for initializing sequential access to the passwd database.

`d_setpwent_r`

From *d_setpwent_r.U*:

This variable conditionally defines the `HAS_SETPWENT_R` symbol, which indicates to the C program that the `setpwent_r()` routine is available.

`d_setregid`

From *d_setregid.U*:

This variable conditionally defines `HAS_SETREGID` if `setregid()` is available to change the real and effective gid of the current process.

`d_setresgid`

From *d_setresgid.U*:

This variable conditionally defines `HAS_SETRESGID` if `setresgid()` is available to change the

real, effective and saved gid of the current process.

`d_setresuid`

From *d_setreuid.U*:

This variable conditionally defines `HAS_SETREUID` if `setresuid()` is available to change the real, effective and saved uid of the current process.

`d_setreuid`

From *d_setreuid.U*:

This variable conditionally defines `HAS_SETREUID` if `setreuid()` is available to change the real and effective uid of the current process.

`d_setrgid`

From *d_setrgid.U*:

This variable conditionally defines the `HAS_SETRGID` symbol, which indicates to the C program that the `setrgid()` routine is available to change the real gid of the current program.

`d_setruid`

From *d_setruid.U*:

This variable conditionally defines the `HAS_SETRUID` symbol, which indicates to the C program that the `setruid()` routine is available to change the real uid of the current program.

`d_setsent`

From *d_setsent.U*:

This variable conditionally defines `HAS_SETSERVENT` if `setservernt()` is available.

`d_setservernt_r`

From *d_setservernt_r.U*:

This variable conditionally defines the `HAS_SETSERVENT_R` symbol, which indicates to the C program that the `setservernt_r()` routine is available.

`d_setsid`

From *d_setsid.U*:

This variable conditionally defines `HAS_SETSID` if `setsid()` is available to set the process group ID.

`d_setvbuf`

From *d_setvbuf.U*:

This variable conditionally defines the `HAS_SETVBUF` symbol, which indicates to the C program that the `setvbuf()` routine is available to change buffering on an open stdio stream.

`d_shm`

From *d_shm.U*:

This variable conditionally defines the `HAS_SHM` symbol, which indicates that the entire `shm*(2)` library is present.

`d_shmat`

From *d_shmat.U*:

This variable conditionally defines the `HAS_SHMAT` symbol, which indicates to the C program that the `shmat()` routine is available.

`d_shmatprototype`

From *d_shmat.U*:

This variable conditionally defines the `HAS_SHMAT_PROTOTYPE` symbol, which indicates that `sys/shm.h` has a prototype for `shmat`.

`d_shmctl`

From *d_shmctl.U*:

This variable conditionally defines the `HAS_SHMCTL` symbol, which indicates to the C program that the `shmctl()` routine is available.

`d_shmdt`

From *d_shmdt.U*:

This variable conditionally defines the `HAS_SHMDT` symbol, which indicates to the C program that the `shmdt()` routine is available.

`d_shmget`

From *d_shmget.U*:

This variable conditionally defines the `HAS_SHMGET` symbol, which indicates to the C program that the `shmget()` routine is available.

`d_sigaction`

From *d_sigaction.U*:

This variable conditionally defines the `HAS_SIGACTION` symbol, which indicates that the `Vr4 sigaction()` routine is available.

`d_siginfo_si_addr`

From *d_siginfo.si.U*:

This variable conditionally defines the `HAS_SIGINFO_SI_ADDR` symbol, which indicates that the `siginfo_t` struct has the `si_addr` member.

`d_siginfo_si_band`

From *d_siginfo.si.U*:

This variable conditionally defines the `HAS_SIGINFO_SI_BAND` symbol, which indicates that the `siginfo_t` struct has the `si_band` member.

`d_siginfo_si_errno`

From *d_siginfo.si.U*:

This variable conditionally defines the `HAS_SIGINFO_SI_ERRNO` symbol, which indicates that the `siginfo_t` struct has the `si_errno` member.

`d_siginfo_si_fd`

From *d_siginfo.si.U*:

This variable conditionally defines the `HAS_SIGINFO_SI_FD` symbol, which indicates that the `siginfo_t` struct has the `si_fd` member.

`d_siginfo_si_pid`

From *d_siginfo.si.U*:

This variable conditionally defines the `HAS_SIGINFO_SI_PID` symbol, which indicates that the `siginfo_t` struct has the `si_pid` member.

`d_siginfo_si_status`

From *d_siginfo.si.U*:

This variable conditionally defines the `HAS_SIGINFO_SI_STATUS` symbol, which indicates that the `siginfo_t` struct has the `si_status` member.

`d_siginfo_si_uid`

From *d_siginfo.si.U*:

This variable conditionally defines the `HAS_SIGINFO_SI_UID` symbol, which indicates that the `siginfo_t` struct has the `si_uid` member.

`d_siginfo_si_value`

From *d_siginfo.si.U*:

This variable conditionally defines the `HAS_SIGINFO_SI_VALUE` symbol, which indicates that the `siginfo_t` struct has the `si_value` member.

`d_signbit`

From *d_signbit.U*:

This variable conditionally defines the `HAS_SIGNBIT` symbol, which indicates to the C program that the `signbit()` routine is available and safe to use with perl's intern NV type.

`d_sigprocmask`

From *d_sigprocmask.U*:

This variable conditionally defines `HAS_SIGPROCMASK` if `sigprocmask()` is available to examine or change the signal mask of the calling process.

`d_sigsetjmp`

From *d_sigsetjmp.U*:

This variable conditionally defines the `HAS_SIGSETJMP` symbol, which indicates that the `sigsetjmp()` routine is available to call `setjmp()` and optionally save the process's signal mask.

`d_sin6_scope_id`

From *d_socket.U*:

This variable conditionally defines the `HAS_SIN6_SCOPE_ID` symbol, which indicates that a struct `sockaddr_in6` structure has the `sin6_scope_id` member.

`d_ssearch`

From *ssearch.U*:

This variable conditionally defines `SSEARCH` to hold the pathname of architecture-dependent library files for `$package`. If `$ssearch` is the same as `$archlib`, then this is set to `undef`.

`d_snprintf`

From *d_snprintf.U*:

This variable conditionally defines the `HAS_SNPRINTF` symbol, which indicates to the C program that the `snprintf()` library function is available.

`d_sockaddr_in6`

From *d_socket.U*:

This variable conditionally defines the `HAS_SOCKADDR_IN6` symbol, which indicates the availability of a struct `sockaddr_in6`.

`d_sockaddr_sa_len`

From *d_socket.U*:

This variable conditionally defines the `HAS_SOCKADDR_SA_LEN` symbol, which indicates that a struct `sockaddr` structure has the `sa_len` member.

`d_sockatmark`

From *d_sockatmark.U*:

This variable conditionally defines the `HAS_SOCKETMARK` symbol, which indicates to the C program that the `socketmark()` routine is available.

`d_socketmarkproto`

From *d_socketmarkproto.U*:

This variable conditionally defines the `HAS_SOCKETMARK_PROTO` symbol, which indicates to the C program that the system provides a prototype for the `socketmark()` function. Otherwise, it is up to the program to supply one.

`d_socket`

From *d_socket.U*:

This variable conditionally defines `HAS_SOCKET`, which indicates that the BSD socket interface is supported.

`d_socklen_t`

From *d_socklen_t.U*:

This symbol will be defined if the C compiler supports `socklen_t`.

`d_socketpair`

From *d_socket.U*:

This variable conditionally defines the `HAS_SOCKETPAIR` symbol, which indicates that the BSD `socketpair()` is supported.

`d_socks5_init`

From *d_socks5_init.U*:

This variable conditionally defines the `HAS_SOCKS5_INIT` symbol, which indicates to the C program that the `socks5_init()` routine is available.

`d_sprintf_returns_strlen`

From *d_sprintf_len.U*:

This variable defines whether `sprintf` returns the length of the string (as per the `ANSI` spec). Some C libraries retain compatibility with pre-`ANSI` C and return a pointer to the passed in buffer; for these this variable will be `undef`.

`d_sqrtl`

From *d_sqrtl.U*:

This variable conditionally defines the `HAS_SQRTL` symbol, which indicates to the C program that the `sqrtl()` routine is available.

`d_srand48_r`

From *d_srand48_r.U*:

This variable conditionally defines the `HAS_SRAND48_R` symbol, which indicates to the C program that the `srand48_r()` routine is available.

`d_srandom_r`

From *d_srandom_r.U*:

This variable conditionally defines the `HAS_SRANDOM_R` symbol, which indicates to the C program that the `srandom_r()` routine is available.

`d_sresgproto`

From *d_sresgproto.U*:

This variable conditionally defines the `HAS_SETRESGID_PROTO` symbol, which indicates to the C program that the system provides a prototype for the `setresgid()` function. Otherwise, it

is up to the program to supply one.

`d_sresuproto`

From *d_sresuproto.U*:

This variable conditionally defines the `HAS_SETRESUID_PROTO` symbol, which indicates to the C program that the system provides a prototype for the `setresuid()` function. Otherwise, it is up to the program to supply one.

`d_stat`

From *d_stat.U*:

This variable conditionally defines `HAS_STAT` if `stat()` is available to get file status.

`d_statblks`

From *d_statblks.U*:

This variable conditionally defines `USE_STAT_BLOCKS` if this system has a `stat` structure declaring `st_blksize` and `st_blocks`.

`d_statfs_f_flags`

From *d_statfs_f_flags.U*:

This variable conditionally defines the `HAS_STRUCT_STATFS_F_FLAGS` symbol, which indicates to `struct statfs` from has `f_flags` member. This kind of `struct statfs` is coming from *sys/mount.h* (BSD), not from *sys/statfs.h* (SYSV).

`d_statfs_s`

From *d_statfs_s.U*:

This variable conditionally defines the `HAS_STRUCT_STATFS` symbol, which indicates that the `struct statfs` is supported.

`d_static_inline`

From *d_static_inline.U*:

This variable conditionally defines the `HAS_STATIC_INLINE` symbol, which indicates that the C compiler supports C99-style static inline. That is, the function can't be called from another translation unit.

`d_statvfs`

From *d_statvfs.U*:

This variable conditionally defines the `HAS_STATVFS` symbol, which indicates to the C program that the `statvfs()` routine is available.

`d_stdio_cnt_lval`

From *d_stdstdio.U*:

This variable conditionally defines `STDIO_CNT_LVALUE` if the `FILE_cnt` macro can be used as an lvalue.

`d_stdio_ptr_lval`

From *d_stdstdio.U*:

This variable conditionally defines `STDIO_PTR_LVALUE` if the `FILE_ptr` macro can be used as an lvalue.

`d_stdio_ptr_lval_nochange_cnt`

From *d_stdstdio.U*:

This symbol is defined if using the `FILE_ptr` macro as an lvalue to increase the pointer by `n` leaves `File_cnt(fp)` unchanged.

`d_stdio_ptr_lval_sets_cnt`

From *d_stdstdio.U*:

This symbol is defined if using the `FILE_ptr` macro as an lvalue to increase the pointer by `n` has the side effect of decreasing the value of `File_cnt(fp)` by `n`.

`d_stdio_stream_array`

From *stdio_streams.U*:

This variable tells whether there is an array holding the stdio streams.

`d_stdio_base`

From *d_stdstdio.U*:

This variable conditionally defines `USE_STDIO_BASE` if this system has a `FILE` structure declaring a usable `_base` field (or equivalent) in *stdio.h*.

`d_stdstdio`

From *d_stdstdio.U*:

This variable conditionally defines `USE_STDIO_PTR` if this system has a `FILE` structure declaring usable `_ptr` and `_cnt` fields (or equivalent) in *stdio.h*.

`d_strchr`

From *d_strchr.U*:

This variable conditionally defines `HAS_STRCHR` if `strchr()` and `strrchr()` are available for string searching.

`d_strcoll`

From *d_strcoll.U*:

This variable conditionally defines `HAS_STRCOLL` if `strcoll()` is available to compare strings using collating information.

`d_structcopy`

From *d_structcopy.U*:

This variable conditionally defines the `USE_STRUCT_COPY` symbol, which indicates to the C program that this C compiler knows how to copy structures.

`d_strerrorm`

From *d_strerror.U*:

This variable holds what `Strerror` is defined as to translate an error code condition into an error message string. It could be `strerror` or a more complex macro emulating `strerror` with `sys_errlist[]`, or the `unknown` string when both `strerror` and `sys_errlist` are missing.

`d_strerror`

From *d_strerror.U*:

This variable conditionally defines `HAS_STRERROR` if `strerror()` is available to translate error numbers to strings.

`d_strerror_r`

From *d_strerror_r.U*:

This variable conditionally defines the `HAS_STRERROR_R` symbol, which indicates to the C program that the `strerror_r()` routine is available.

`d_strftime`

From *d_strftime.U*:

This variable conditionally defines the `HAS_STRFTIME` symbol, which indicates to the C program that the `strftime()` routine is available.

`d_strlcat`

From *d_strlcat.U*:

This variable conditionally defines the `HAS_STRLCAT` symbol, which indicates to the C program that the `strlcat()` routine is available.

`d_strncpy`

From *d_strncpy.U*:

This variable conditionally defines the `HAS_STRNCPY` symbol, which indicates to the C program that the `strncpy()` routine is available.

`d_strtod`

From *d_strtod.U*:

This variable conditionally defines the `HAS_STRTOD` symbol, which indicates to the C program that the `strtod()` routine is available to provide better numeric string conversion than `atof()`.

`d_strtol`

From *d_strtol.U*:

This variable conditionally defines the `HAS_STRTOL` symbol, which indicates to the C program that the `strtol()` routine is available to provide better numeric string conversion than `atoi()` and friends.

`d_strtold`

From *d_strtold.U*:

This variable conditionally defines the `HAS_STRTOLD` symbol, which indicates to the C program that the `strtold()` routine is available.

`d_strtoll`

From *d_strtoll.U*:

This variable conditionally defines the `HAS_STRTOLL` symbol, which indicates to the C program that the `strtoll()` routine is available.

`d_strtoq`

From *d_strtoq.U*:

This variable conditionally defines the `HAS_STRTOQ` symbol, which indicates to the C program that the `strtoq()` routine is available.

`d_strtoul`

From *d_strtoul.U*:

This variable conditionally defines the `HAS_STRTOUL` symbol, which indicates to the C program that the `strtoul()` routine is available to provide conversion of strings to unsigned long.

`d_strtoull`

From *d_strtoull.U*:

This variable conditionally defines the `HAS_STRTOULL` symbol, which indicates to the C program that the `strtoull()` routine is available.

`d_strtouq`

From *d_strtouq.U*:

This variable conditionally defines the `HAS_STRTOUQ` symbol, which indicates to the C program that the `strtouq()` routine is available.

`d_strxfrm`

From *d_strxfrm.U*:

This variable conditionally defines `HAS_STRXFRM` if `strxfrm()` is available to transform strings.

`d_suidsafe`

From *d_dosuid.U*:

This variable conditionally defines `SETUID_SCRIPTS_ARE_SECURE_NOW` if `setuid` scripts can be secure. This test looks in `/dev/fd/`.

`d_symlink`

From *d_symlink.U*:

This variable conditionally defines the `HAS_SYMLINK` symbol, which indicates to the C program that the `symlink()` routine is available to create symbolic links.

`d_syscall`

From *d_syscall.U*:

This variable conditionally defines `HAS_SYSCALL` if `syscall()` is available call arbitrary system calls.

`d_syscallproto`

From *d_syscallproto.U*:

This variable conditionally defines the `HAS_SYSCALL_PROTO` symbol, which indicates to the C program that the system provides a prototype for the `syscall()` function. Otherwise, it is up to the program to supply one.

`d_sysconf`

From *d_sysconf.U*:

This variable conditionally defines the `HAS_SYSCONF` symbol, which indicates to the C program that the `sysconf()` routine is available to determine system related limits and options.

`d_syserrnlst`

From *d_strerror.U*:

This variable conditionally defines `HAS_SYS_ERRNOLIST` if `sys_errnolist[]` is available to translate error numbers to the symbolic name.

`d_syserrrlst`

From *d_strerror.U*:

This variable conditionally defines `HAS_SYS_ERRRLIST` if `sys_errlist[]` is available to translate error numbers to strings.

`d_system`

From *d_system.U*:

This variable conditionally defines `HAS_SYSTEM` if `system()` is available to issue a shell command.

`d_tcgetpgrp`

From *d_tcgetpgrp.U*:

This variable conditionally defines the `HAS_TCGETPGRP` symbol, which indicates to the C program that the `tcgetpgrp()` routine is available. to get foreground process group `ID`.

`d_tcsetpgrp`

From *d_tcsetpgrp.U*:

This variable conditionally defines the `HAS_TCSETPGRP` symbol, which indicates to the C program that the `tcsetpgrp()` routine is available to set foreground process group ID.

`d_telldir`

From *d_readdir.U*:

This variable conditionally defines `HAS_TELLDIR` if `telldir()` is available.

`d_telldirproto`

From *d_telldirproto.U*:

This variable conditionally defines the `HAS_TELLDIR_PROTO` symbol, which indicates to the C program that the system provides a prototype for the `telldir()` function. Otherwise, it is up to the program to supply one.

`d_tgamma`

From *d_tgamma.U*:

This variable conditionally defines the `HAS_TGAMMA` symbol, which indicates to the C program that the `tgamma()` routine is available for the gamma function. See also `d_lgamma`.

`d_time`

From *d_time.U*:

This variable conditionally defines the `HAS_TIME` symbol, which indicates that the `time()` routine exists. The `time()` routine is normally provided on UNIX systems.

`d_timegm`

From *d_timegm.U*:

This variable conditionally defines the `HAS_TIMEGM` symbol, which indicates to the C program that the `timegm()` routine is available.

`d_times`

From *d_times.U*:

This variable conditionally defines the `HAS_TIMES` symbol, which indicates that the `times()` routine exists. The `times()` routine is normally provided on UNIX systems. You may have to include `<sys/times.h>`.

`d_tm_tm_gmtimeoff`

From *i_time.U*:

This variable conditionally defines `HAS_TM_TM_GMTOFF`, which indicates to the C program that the struct `tm` has the `tm_gmtimeoff` field.

`d_tm_tm_zone`

From *i_time.U*:

This variable conditionally defines `HAS_TM_TM_ZONE`, which indicates to the C program that the struct `tm` has the `tm_zone` field.

`d_tmpnam_r`

From *d_tmpnam_r.U*:

This variable conditionally defines the `HAS_TMPNAM_R` symbol, which indicates to the C program that the `tmpnam_r()` routine is available.

`d_trunc`

From *d_trunc.U*:

This variable conditionally defines the `HAS_TRUNC` symbol, which indicates to the C program that the `trunc()` routine is available to round doubles towards zero.

`d_truncate`

From *d_truncate.U*:

This variable conditionally defines `HAS_TRUNCATE` if `truncate()` is available to truncate files.

`d_truncl`

From *d_truncl.U*:

This variable conditionally defines the `HAS_TRUNCL` symbol, which indicates to the C program that the `truncl()` routine is available to round long doubles towards zero. If `copysignl` is also present, we can emulate `modfl`.

`d_ttyname_r`

From *d_ttyname_r.U*:

This variable conditionally defines the `HAS_TTYNAME_R` symbol, which indicates to the C program that the `ttyname_r()` routine is available.

`d_tzname`

From *d_tzname.U*:

This variable conditionally defines `HAS_TZNAME` if `tzname[]` is available to access timezone names.

`d_u32align`

From *d_u32align.U*:

This variable tells whether you must access character data through U32-aligned pointers.

`d_ualarm`

From *d_ualarm.U*:

This variable conditionally defines the `HAS_UALARM` symbol, which indicates to the C program that the `ualarm()` routine is available.

`d_umask`

From *d_umask.U*:

This variable conditionally defines the `HAS_UMASK` symbol, which indicates to the C program that the `umask()` routine is available. to set and get the value of the file creation mask.

`d_uname`

From *d_gethname.U*:

This variable conditionally defines the `HAS_UNAME` symbol, which indicates to the C program that the `uname()` routine may be used to derive the host name.

`d_union_semun`

From *d_union_semun.U*:

This variable conditionally defines `HAS_UNION_SEMUN` if the union `semun` is defined by including `<sys/sem.h>`.

`d_unordered`

From *d_unordered.U*:

This variable conditionally defines the `HAS_UNORDERED` symbol, which indicates to the C program that the `unordered()` routine is available.

`d_unsetenv`

From *d_unsetenv.U*:

This variable conditionally defines the `HAS_UNSETENV` symbol, which indicates to the C

program that the `unsetenv()` routine is available.

`d_uselocale`

From *d_newlocale.U*:

This variable conditionally defines the `HAS_USELOCALE` symbol, which indicates to the C program that the `uselocale()` routine is available to set the current locale for the calling thread.

`d_usleep`

From *d_usleep.U*:

This variable conditionally defines `HAS_USLEEP` if `usleep()` is available to do high granularity sleeps.

`d_usleepproto`

From *d_usleepproto.U*:

This variable conditionally defines the `HAS_USLEEP_PROTO` symbol, which indicates to the C program that the system provides a prototype for the `usleep()` function. Otherwise, it is up to the program to supply one.

`d_ustat`

From *d_ustat.U*:

This variable conditionally defines `HAS_USTAT` if `ustat()` is available to query file system statistics by `dev_t`.

`d_vendorarch`

From *vendorarch.U*:

This variable conditionally defined `PERL_VENDORARCH`.

`d_vendorbin`

From *vendorbin.U*:

This variable conditionally defines `PERL_VENDORBIN`.

`d_vendorlib`

From *vendorlib.U*:

This variable conditionally defines `PERL_VENDORLIB`.

`d_vendorscript`

From *vendorscript.U*:

This variable conditionally defines `PERL_VENDORSRIPT`.

`d_vfork`

From *d_vfork.U*:

This variable conditionally defines the `HAS_VFORK` symbol, which indicates the `vfork()` routine is available.

`d_void_closedir`

From *d_closedir.U*:

This variable conditionally defines `VOID_CLOSEDIR` if `closedir()` does not return a value.

`d_voidsig`

From *d_voidsig.U*:

This variable conditionally defines `VOIDSIG` if this system declares `"void (*signal(...))()"` in *signal.h*. The old way was to declare it as `"int (*signal(...))()"`.

`d_voidtty`

From *d_sysioctl.U*:

This variable conditionally defines `USE_IOCTL` to indicate that the `ioctl()` call with `TIOCNOTTY` should be used to void tty association. Otherwise (on `USG` probably), it is enough to close the standard file descriptors and do a `setpgrp()`.

`d_volatile`

From *d_volatile.U*:

This variable conditionally defines the `HASVOLATILE` symbol, which indicates to the C program that this C compiler knows about the volatile declaration.

`d_vprintf`

From *d_vprintf.U*:

This variable conditionally defines the `HAS_VPRINTF` symbol, which indicates to the C program that the `vprintf()` routine is available to `printf` with a pointer to an argument list.

`d_vsnprintf`

From *d_snprintf.U*:

This variable conditionally defines the `HAS_VSNPRINTF` symbol, which indicates to the C program that the `vsprintf()` library function is available.

`d_wait4`

From *d_wait4.U*:

This variable conditionally defines the `HAS_WAIT4` symbol, which indicates the `wait4()` routine is available.

`d_waitpid`

From *d_waitpid.U*:

This variable conditionally defines `HAS_WAITPID` if `waitpid()` is available to wait for child process.

`d_wcscmp`

From *d_wcscmp.U*:

This variable conditionally defines the `HAS_WCSCMP` symbol if the `wcscmp()` routine is available and can be used to compare wide character strings.

`d_wcstombs`

From *d_wcstombs.U*:

This variable conditionally defines the `HAS_WCSTOMBS` symbol, which indicates to the C program that the `wcstombs()` routine is available to convert wide character strings to multibyte strings.

`d_wcsxfrm`

From *d_wcsxfrm.U*:

This variable conditionally defines the `HAS_WCSXFRM` symbol if the `wcsxfrm()` routine is available and can be used to compare wide character strings.

`d_wctomb`

From *d_wctomb.U*:

This variable conditionally defines the `HAS_WCTOMB` symbol, which indicates to the C program that the `wctomb()` routine is available to convert a wide character to a multibyte.

`d_writev`

From *d_writev.U*:

This variable conditionally defines the `HAS_WRITEV` symbol, which indicates to the C program that the `writev()` routine is available.

`d_xenix`

From *Guess.U*:

This variable conditionally defines the symbol `XENIX`, which alerts the C program that it runs under Xenix.

`date`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the date program. After Configure runs, the value is reset to a plain `date` and is not useful.

`db_hashtype`

From *i_db.U*:

This variable contains the type of the hash structure element in the `<db.h>` header file. In older versions of DB, it was `int`, while in newer ones it is `u_int32_t`.

`db_prefixtype`

From *i_db.U*:

This variable contains the type of the prefix structure element in the `<db.h>` header file. In older versions of DB, it was `int`, while in newer ones it is `size_t`.

`db_version_major`

From *i_db.U*:

This variable contains the major version number of Berkeley DB found in the `<db.h>` header file.

`db_version_minor`

From *i_db.U*:

This variable contains the minor version number of Berkeley DB found in the `<db.h>` header file. For DB version 1 this is always 0.

`db_version_patch`

From *i_db.U*:

This variable contains the patch version number of Berkeley DB found in the `<db.h>` header file. For DB version 1 this is always 0.

`direntrytype`

From *i_dirent.U*:

This symbol is set to `struct direct` or `struct dirent` depending on whether `dirent` is available or not. You should use this pseudo type to portably declare your directory entries.

`dlextr`

From *dlextr.U*:

This variable contains the extension that is to be used for the dynamically loaded modules that perl generates.

`dlsrcl`

From *dlsrcl.U*:

This variable contains the name of the dynamic loading file that will be used with the package.

`doubleinfbytes`

From *infnan.U*:

This variable contains comma-separated list of hexadecimal bytes for the double precision infinity.

`doublekind`

From *longdblfluo.U*:

This variable, if defined, encodes the type of a double: 1 = IEEE 754 32-bit little endian, 2 = IEEE 754 32-bit big endian, 3 = IEEE 754 64-bit little endian, 4 = IEEE 754 64-bit big endian, 5 = IEEE 754 128-bit little endian, 6 = IEEE 754 128-bit big endian, 7 = IEEE 754 64-bit mixed endian le-be, 8 = IEEE 754 64-bit mixed endian be-le, -1 = unknown format.

`doublemantbits`

From *mantbits.U*:

This symbol, if defined, tells how many mantissa bits there are in double precision floating point format. Note that this is usually `DBL_MANT_DIG` minus one, since with the standard IEEE 754 formats `DBL_MANT_DIG` includes the implicit bit which doesn't really exist.

`doublenanbytes`

From *infnan.U*:

This variable contains comma-separated list of hexadecimal bytes for the double precision not-a-number.

`doublesize`

From *doublesize.U*:

This variable contains the value of the `DOUBLESIZE` symbol, which indicates to the C program how many bytes there are in a double.

`drand01`

From *randfunc.U*:

Indicates the macro to be used to generate normalized random numbers. Uses `randfunc`, often divided by `(double) (((unsigned long) 1 << randbits))` in order to normalize the result. In C programs, the macro `Drand01` is mapped to `drand01`.

`drand48_r_proto`

From *d_drand48_r.U*:

This variable encodes the prototype of `drand48_r`. It is zero if `d_drand48_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_drand48_r` is defined.

`dtrace`

From *usedtrace.U*:

This variable holds the location of the `dtrace` executable.

`dtraceobject`

From *dtraceobject.U*:

Whether we need to build an object file with the `dtrace` tool.

`dynamic_ext`

From *Extensions.U*:

This variable holds a list of XS extension files we want to link dynamically into the package. It is used by `Makefile`.

e

`eagain`

From *nblock_io.U*:

This variable bears the symbolic `errno` code set by `read()` when no data is present on the file and non-blocking I/O was enabled (otherwise, `read()` blocks naturally).

`ebcdic`

From *ebcdic.U*:

This variable conditionally defines `EBCDIC` if this system uses `EBCDIC` encoding.

`echo`

From *Loc.U*:

This variable is used internally by `Configure` to determine the full pathname (if any) of the `echo` program. After `Configure` runs, the value is reset to a plain `echo` and is not useful.

`egrep`

From *Loc.U*:

This variable is used internally by `Configure` to determine the full pathname (if any) of the `egrep` program. After `Configure` runs, the value is reset to a plain `egrep` and is not useful.

`emacs`

From *Loc.U*:

This variable is defined but not used by `Configure`. The value is the empty string and is not useful.

`endgrent_r_proto`

From *d_endgrent_r.U*:

This variable encodes the prototype of `endgrent_r`. It is zero if `d_endgrent_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_endgrent_r` is defined.

`endhostent_r_proto`

From *d_endhostent_r.U*:

This variable encodes the prototype of `endhostent_r`. It is zero if `d_endhostent_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_endhostent_r` is defined.

`endnetent_r_proto`

From *d_endnetent_r.U*:

This variable encodes the prototype of `endnetent_r`. It is zero if `d_endnetent_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_endnetent_r` is defined.

`endprotoent_r_proto`

From *d_endprotoent_r.U*:

This variable encodes the prototype of `endprotoent_r`. It is zero if `d_endprotoent_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_endprotoent_r` is defined.

`endpwent_r_proto`

From *d_endpwent_r.U*:

This variable encodes the prototype of `endpwent_r`. It is zero if `d_endpwent_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_endpwent_r` is defined.

`endservent_r_proto`

From *d_endservent_r.U*:

This variable encodes the prototype of `endservent_r`. It is zero if `d_endservent_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_endservent_r` is defined.

`eunicefix`

From *Init.U*:

When running under Eunice this variable contains a command which will convert a shell script to the proper form of text file for it to be executable by the shell. On other systems it is a no-op.

`exe_ext`

From *Unix.U*:

This is an old synonym for `_exe`.

`expr`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the `expr` program. After Configure runs, the value is reset to a plain `expr` and is not useful.

`extensions`

From *Extensions.U*:

This variable holds a list of all extension files (both `xs` and non-`xs`) installed with the package. It is propagated to *Config.pm* and is typically used to test whether a particular extension is available.

`extern_C`

From *Csym.U*:

ANSI C requires `extern` where C++ requires 'extern C'. This variable can be used in Configure to do the right thing.

`extras`

From *Extras.U*:

This variable holds a list of extra modules to install.

f

`fflushall`

From *fflushall.U*:

This symbol, if defined, tells that to flush all pending stdio output one must loop through all the stdio file handles stored in an array and fflush them. Note that if `fflushNULL` is defined, `fflushall` will not even be probed for and will be left undefined.

`fflushNULL`

From *fflushall.U*:

This symbol, if defined, tells that `fflush(NULL)` correctly flushes all pending stdio output without side effects. In particular, on some platforms calling `fflush(NULL)` *still* corrupts `STDIN` if it is a pipe.

`find`

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

`firstmakefile`

From *Unix.U*:

This variable defines the first file searched by make. On unix, it is makefile (then Makefile). On case-insensitive systems, it might be something else. This is only used to deal with convoluted make depend tricks.

`flex`

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

`fpossize`

From *fpossize.U*:

This variable contains the size of a fpos_t in bytes.

`fpostype`

From *fpostype.U*:

This variable defines Fpos_t to be something like fpos_t, long, uint, or whatever type is used to declare file positions in libc.

`freetype`

From *malloclsrc.U*:

This variable contains the return type of free(). It is usually void, but occasionally int.

`from`

From *Cross.U*:

This variable contains the command used by Configure to copy files from the target host. Useful and available only during Perl build. The string : if not cross-compiling.

`full_ar`

From *Loc_ar.U*:

This variable contains the full pathname to `ar`, whether or not the user has specified `portability`. This is only used in the *Makefile.SH*.

`full_csh`

From *d_csh.U*:

This variable contains the full pathname to `csh`, whether or not the user has specified `portability`. This is only used in the compiled C program, and we assume that all systems which can share this executable will have the same full pathname to `csh`.

`full_sed`

From *Loc_sed.U*:

This variable contains the full pathname to `sed`, whether or not the user has specified `portability`. This is only used in the compiled C program, and we assume that all systems which can share this executable will have the same full pathname to `sed`.

9

`gccansipedantic`

From *gccvers.U*:

If GNU cc (gcc) is used, this variable will enable (if set) the `-ansi` and `-pedantic` ccflags for building core files (through `cflags` script). (See *Porting/pumpkin.pod* for full description).

`gccosandvers`

From *gccvers.U*:

If GNU cc (gcc) is used, this variable holds the operating system and version used to compile gcc. It is set to " if not gcc, or if nothing useful can be parsed as the os version.

`gccversion`

From *gccvers.U*:

If GNU cc (gcc) is used, this variable holds 1 or 2 to indicate whether the compiler is version 1 or 2. This is used in setting some of the default cflags. It is set to " if not gcc.

`getgrent_r_proto`

From *d_getgrent_r.U*:

This variable encodes the prototype of `getgrent_r`. It is zero if `d_getgrent_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_getgrent_r` is defined.

`getgrgid_r_proto`

From *d_getgrgid_r.U*:

This variable encodes the prototype of `getgrgid_r`. It is zero if `d_getgrgid_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_getgrgid_r` is defined.

`getgrnam_r_proto`

From *d_getgrnam_r.U*:

This variable encodes the prototype of `getgrnam_r`. It is zero if `d_getgrnam_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_getgrnam_r` is defined.

`gethostbyaddr_r_proto`

From *d_gethostbyaddr_r.U*:

This variable encodes the prototype of `gethostbyaddr_r`. It is zero if `d_gethostbyaddr_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_gethostbyaddr_r` is defined.

`gethostbyname_r_proto`

From *d_gethostbyname_r.U*:

This variable encodes the prototype of `gethostbyname_r`. It is zero if `d_gethostbyname_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_gethostbyname_r` is defined.

`gethostent_r_proto`

From *d_gethostent_r.U*:

This variable encodes the prototype of `gethostent_r`. It is zero if `d_gethostent_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_gethostent_r` is defined.

`getlogin_r_proto`

From *d_getlogin_r.U*:

This variable encodes the prototype of `getlogin_r`. It is zero if `d_getlogin_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_getlogin_r` is defined.

`getnetbyaddr_r_proto`

From *d_getnetbyaddr_r.U*:

This variable encodes the prototype of `getnetbyaddr_r`. It is zero if `d_getnetbyaddr_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_getnetbyaddr_r` is defined.

`getnetbyname_r_proto`

From *d_getnetbyname_r.U*:

This variable encodes the prototype of `getnetbyname_r`. It is zero if `d_getnetbyname_r` is

undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_getnetbyname_r` is defined.

`getnetent_r_proto`

From *d_getnetent_r.U*:

This variable encodes the prototype of `getnetent_r`. It is zero if `d_getnetent_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_getnetent_r` is defined.

`getprotobyname_r_proto`

From *d_getprotobyname_r.U*:

This variable encodes the prototype of `getprotobyname_r`. It is zero if `d_getprotobyname_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_getprotobyname_r` is defined.

`getprotobynumber_r_proto`

From *d_getprotobynumber_r.U*:

This variable encodes the prototype of `getprotobynumber_r`. It is zero if `d_getprotobynumber_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_getprotobynumber_r` is defined.

`getprotoent_r_proto`

From *d_getprotoent_r.U*:

This variable encodes the prototype of `getprotoent_r`. It is zero if `d_getprotoent_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_getprotoent_r` is defined.

`getpwent_r_proto`

From *d_getpwent_r.U*:

This variable encodes the prototype of `getpwent_r`. It is zero if `d_getpwent_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_getpwent_r` is defined.

`getpwnam_r_proto`

From *d_getpwnam_r.U*:

This variable encodes the prototype of `getpwnam_r`. It is zero if `d_getpwnam_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_getpwnam_r` is defined.

`getpwuid_r_proto`

From *d_getpwuid_r.U*:

This variable encodes the prototype of `getpwuid_r`. It is zero if `d_getpwuid_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_getpwuid_r` is defined.

`getservbyname_r_proto`

From *d_getservbyname_r.U*:

This variable encodes the prototype of `getservbyname_r`. It is zero if `d_getservbyname_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_getservbyname_r` is defined.

`getservbyport_r_proto`

From *d_getservbyport_r.U*:

This variable encodes the prototype of `getservbyport_r`. It is zero if `d_getservbyport_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_getservbyport_r` is defined.

`getservent_r_proto`

From *d_getservent_r.U*:

This variable encodes the prototype of `getservent_r`. It is zero if `d_getservent_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_getservent_r` is defined.

`getspnam_r_proto`

From *d_getspnam_r.U*:

This variable encodes the prototype of `getspnam_r`. It is zero if `d_getspnam_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_getspnam_r` is defined.

`gidformat`

From *gidf.U*:

This variable contains the format string used for printing a `Gid_t`.

`gidsign`

From *gidsign.U*:

This variable contains the signedness of a `gidtype`. 1 for unsigned, -1 for signed.

`gidsize`

From *gidsize.U*:

This variable contains the size of a `gidtype` in bytes.

`gidtype`

From *gidtype.U*:

This variable defines `Gid_t` to be something like `gid_t`, `int`, `ushort`, or whatever type is used to declare the return type of `getgid()`. Typically, it is the type of group ids in the kernel.

`glibpth`

From *libpth.U*:

This variable holds the general path (space-separated) used to find libraries. It may contain directories that do not exist on this platform, `libpth` is the cleaned-up version.

`gmake`

From *Loc.U*:

This variable is used internally by `Configure` to determine the full pathname (if any) of the `gmake` program. After `Configure` runs, the value is reset to a plain `gmake` and is not useful.

`gmtime_r_proto`

From *d_gmtime_r.U*:

This variable encodes the prototype of `gmtime_r`. It is zero if `d_gmtime_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_gmtime_r` is defined.

`gnulibc_version`

From *d_gnulibc.U*:

This variable contains the version number of the GNU C library. It is usually something like 2.2.5. It is a plain " if this is not the GNU C library, or if the version is unknown.

`grep`

From *Loc.U*:

This variable is used internally by `Configure` to determine the full pathname (if any) of the `grep` program. After `Configure` runs, the value is reset to a plain `grep` and is not useful.

`groupcat`

From *nis.U*:

This variable contains a command that produces the text of the `/etc/group` file. This is normally `"cat /etc/group"`, but can be `"ypcat group"` when `NIS` is used. On some systems, such as `os390`, there may be no equivalent command, in which case this variable is unset.

`groupstype`

From *groupstype.U*:

This variable defines `Groups_t` to be something like `gid_t`, `int`, `ushort`, or whatever type is used for the second argument to `getgroups()` and `setgroups()`. Usually, this is the same as `gidtype` (`gid_t`), but sometimes it isn't.

`gzip`

From *Loc.U*:

This variable is used internally by `Configure` to determine the full pathname (if any) of the `gzip` program. After `Configure` runs, the value is reset to a plain `gzip` and is not useful.

h

`h_fcntl`

From *h_fcntl.U*:

This variable gets set in various places to tell `i_fcntl` that `<fcntl.h>` should be included.

`h_sysfile`

From *h_sysfile.U*:

This variable gets set in various places to tell `i_sys_file` that `<sys/file.h>` should be included.

`hint`

From *Oldconfig.U*:

Gives the type of hints used for previous answers. May be one of `default`, `recommended` or `previous`.

`hostcat`

From *nis.U*:

This variable contains a command that produces the text of the `/etc/hosts` file. This is normally `"cat /etc/hosts"`, but can be `"ypcat hosts"` when `NIS` is used. On some systems, such as `os390`, there may be no equivalent command, in which case this variable is unset.

`hostgenerate`

From *Cross.U*:

This variable contains the path to a `generate_uudmap` binary that can be run on the host `os` when cross-compiling. Useful and available only during Perl build. Empty string `"` if not cross-compiling.

`hostosname`

From *Cross.U*:

This variable contains the original value of `$_` for `hostperl` when cross-compiling. This is useful to pick the proper tools when running build code in the host. Empty string `"` if not cross-compiling.

`hostperl`

From *Cross.U*:

This variable contains the path to a `miniperl` binary that can be run on the host `os` when cross-compiling. Useful and available only during Perl build. Empty string `"` if not cross-compiling.

html1dir

From *html1dir.U*:

This variable contains the name of the directory in which html source pages are to be put. This directory is for pages that describe whole programs, not libraries or modules. It is intended to correspond roughly to section 1 of the Unix manuals.

html1direxp

From *html1dir.U*:

This variable is the same as the html1dir variable, but is filename expanded at configuration time, for convenient use in makefiles.

html3dir

From *html3dir.U*:

This variable contains the name of the directory in which html source pages are to be put. This directory is for pages that describe libraries or modules. It is intended to correspond roughly to section 3 of the Unix manuals.

html3direxp

From *html3dir.U*:

This variable is the same as the html3dir variable, but is filename expanded at configuration time, for convenient use in makefiles.

i

i16size

From *perl.v.U*:

This variable is the size of an I16 in bytes.

i16type

From *perl.v.U*:

This variable contains the C type used for Perl's I16.

i32size

From *perl.v.U*:

This variable is the size of an I32 in bytes.

i32type

From *perl.v.U*:

This variable contains the C type used for Perl's I32.

i64size

From *perl.v.U*:

This variable is the size of an I64 in bytes.

i64type

From *perl.v.U*:

This variable contains the C type used for Perl's I64.

i8size

From *perl.v.U*:

This variable is the size of an I8 in bytes.

i8type

From *perl5v.U*:

This variable contains the C type used for Perl's I8.

`i_arpainet`

From *i_arpainet.U*:

This variable conditionally defines the `I_ARPA_INET` symbol, and indicates whether a C program should include `<arpa/inet.h>`.

`i_assert`

From *i_assert.U*:

This variable conditionally defines the `I_ASSERT` symbol, which indicates to the C program that `<assert.h>` exists and could be included.

`i_bfd`

From *i_bfd.U*:

This variable conditionally defines the `I_BFD` symbol, and indicates whether a C program can include `<bfd.h>`.

`i_bsdioc1`

From *i_sysioc1.U*:

This variable conditionally defines the `I_SYS_BSDIOCTL` symbol, which indicates to the C program that `<sys/bsdioc1.h>` exists and should be included.

`i_crypt`

From *i_crypt.U*:

This variable conditionally defines the `I_CRYPT` symbol, and indicates whether a C program should include `<crypt.h>`.

`i_db`

From *i_db.U*:

This variable conditionally defines the `I_DB` symbol, and indicates whether a C program may include Berkeley's DB include file `<db.h>`.

`i_dbm`

From *i_dbm.U*:

This variable conditionally defines the `I_DBM` symbol, which indicates to the C program that `<dbm.h>` exists and should be included.

`i_dirent`

From *i_dirent.U*:

This variable conditionally defines `I_DIRENT`, which indicates to the C program that it should include `<dirent.h>`.

`i_dlfcn`

From *i_dlfcn.U*:

This variable conditionally defines the `I_DLFCN` symbol, which indicates to the C program that `<dlfcn.h>` exists and should be included.

`i_execinfo`

From *i_execinfo.U*:

This variable conditionally defines the `I_EXECINFO` symbol, and indicates whether a C program may include `<execinfo.h>`, for `backtrace()` support.

`i_fcntl`

From *i_fcntl.U*:

This variable controls the value of `I_FCNTL` (which tells the C program to include `<fcntl.h>`).

`i_fenv`

From *i_fenv.U*:

This variable conditionally defines the `I_FENV` symbol, which indicates to the C program that `<fenv.h>` exists and should be included.

`i_float`

From *i_float.U*:

This variable conditionally defines the `I_FLOAT` symbol, and indicates whether a C program may include `<float.h>` to get symbols like `DBL_MAX` or `DBL_MIN`, i.e. machine dependent floating point values.

`i_fp`

From *i_fp.U*:

This variable conditionally defines the `I_FP` symbol, and indicates whether a C program should include `<fp.h>`.

`i_fp_class`

From *i_fp_class.U*:

This variable conditionally defines the `I_FP_CLASS` symbol, and indicates whether a C program should include `<fp_class.h>`.

`i_gdbm`

From *i_gdbm.U*:

This variable conditionally defines the `I_GDBM` symbol, which indicates to the C program that `<gdbm.h>` exists and should be included.

`i_gdbm_ndbm`

From *i_ndbm.U*:

This variable conditionally defines the `I_GDBM_NDBM` symbol, which indicates to the C program that `<gdbm-ndbm.h>` exists and should be included. This is the location of the *ndbm.h* compatibility file in Debian 4.0.

`i_gdbmndbm`

From *i_ndbm.U*:

This variable conditionally defines the `I_GDBMNDBM` symbol, which indicates to the C program that `<gdbm/ndbm.h>` exists and should be included. This was the location of the *ndbm.h* compatibility file in RedHat 7.1.

`i_grp`

From *i_grp.U*:

This variable conditionally defines the `I_GRP` symbol, and indicates whether a C program should include `<grp.h>`.

`i_ieeefp`

From *i_ieeefp.U*:

This variable conditionally defines the `I_IEEEFP` symbol, and indicates whether a C program should include `<ieeefp.h>`.

`i_inttypes`

From *i_inttypes.U*:

This variable conditionally defines the `I_INTTYPES` symbol, and indicates whether a C program should include `<inttypes.h>`.

`i_langinfo`

From *i_langinfo.U*:

This variable conditionally defines the `I_LANGINFO` symbol, and indicates whether a C program should include `<langinfo.h>`.

`i_libutil`

From *i_libutil.U*:

This variable conditionally defines the `I_LIBUTIL` symbol, and indicates whether a C program should include `<libutil.h>`.

`i_limits`

From *i_limits.U*:

This variable conditionally defines the `I_LIMITS` symbol, and indicates whether a C program may include `<limits.h>` to get symbols like `WORD_BIT` and friends.

`i_locale`

From *i_locale.U*:

This variable conditionally defines the `I_LOCALE` symbol, and indicates whether a C program should include `<locale.h>`.

`i_machcthr`

From *i_machcthr.U*:

This variable conditionally defines the `I_MACH_CTHREADS` symbol, and indicates whether a C program should include `<mach/cthrads.h>`.

`i_malloc`

From *i_malloc.U*:

This variable conditionally defines the `I_MALLOC` symbol, and indicates whether a C program should include `<malloc.h>`.

`i_mallocmalloc`

From *i_mallocmalloc.U*:

This variable conditionally defines the `I_MALLOCMALLOC` symbol, and indicates whether a C program should include `<malloc/malloc.h>`.

`i_math`

From *i_math.U*:

This variable conditionally defines the `I_MATH` symbol, and indicates whether a C program may include `<math.h>`.

`i_memory`

From *i_memory.U*:

This variable conditionally defines the `I_MEMORY` symbol, and indicates whether a C program should include `<memory.h>`.

`i_mntent`

From *i_mntent.U*:

This variable conditionally defines the `I_MNTENT` symbol, and indicates whether a C program

should include `<mntent.h>`.

`i_ndbm`

From *i_ndbm.U*:

This variable conditionally defines the `I_NDBM` symbol, which indicates to the C program that `<ndbm.h>` exists and should be included.

`i_netdb`

From *i_netdb.U*:

This variable conditionally defines the `I_NETDB` symbol, and indicates whether a C program should include `<netdb.h>`.

`i_neterrno`

From *i_netermo.U*:

This variable conditionally defines the `I_NET_ERRNO` symbol, which indicates to the C program that `<net/errno.h>` exists and should be included.

`i_netinettcp`

From *i_netinettcp.U*:

This variable conditionally defines the `I_NETINET_TCP` symbol, and indicates whether a C program should include `<netinet/tcp.h>`.

`i_niin`

From *i_niin.U*:

This variable conditionally defines `I_NETINET_IN`, which indicates to the C program that it should include `<netinet/in.h>`. Otherwise, you may try `<sys/in.h>`.

`i_poll`

From *i_poll.U*:

This variable conditionally defines the `I_POLL` symbol, and indicates whether a C program should include `<poll.h>`.

`i_prot`

From *i_prot.U*:

This variable conditionally defines the `I_PROT` symbol, and indicates whether a C program should include `<prot.h>`.

`i_pthread`

From *i_pthread.U*:

This variable conditionally defines the `I_PTHREAD` symbol, and indicates whether a C program should include `<pthread.h>`.

`i_pwd`

From *i_pwd.U*:

This variable conditionally defines `I_PWD`, which indicates to the C program that it should include `<pwd.h>`.

`i_quadmath`

From *i_quadmath.U*:

This variable conditionally defines `I_QUADMATH`, which indicates to the C program that it should include `<quadmath.h>`.

`i_rpcsvcdbm`

From *i_dbm.U*:

This variable conditionally defines the `I_RPCSVCS_DBM` symbol, which indicates to the C program that `<rpcsvc/dbm.h>` exists and should be included. Some System V systems might need this instead of `<dbm.h>`.

`i_sgtty`

From *i_termio.U*:

This variable conditionally defines the `I_SGTTY` symbol, which indicates to the C program that it should include `<sgtty.h>` rather than `<termio.h>`.

`i_shadow`

From *i_shadow.U*:

This variable conditionally defines the `I_SHADOW` symbol, and indicates whether a C program should include `<shadow.h>`.

`i_socks`

From *i_socks.U*:

This variable conditionally defines the `I SOCKS` symbol, and indicates whether a C program should include `<socks.h>`.

`i_stdarg`

From *i_varhdr.U*:

This variable conditionally defines the `I_STDARG` symbol, which indicates to the C program that `<stdarg.h>` exists and should be included.

`i_stdbool`

From *i_stdbool.U*:

This variable conditionally defines the `I_STDBOOL` symbol, which indicates to the C program that `<stdbool.h>` exists and should be included.

`i_stddef`

From *i_stddef.U*:

This variable conditionally defines the `I_STDDEF` symbol, which indicates to the C program that `<stddef.h>` exists and should be included.

`i_stdint`

From *i_stdint.U*:

This variable conditionally defines the `I_STDINT` symbol, which indicates to the C program that `<stdint.h>` exists and should be included.

`i_stdlib`

From *i_stdlib.U*:

This variable conditionally defines the `I_STDLIB` symbol, which indicates to the C program that `<stdlib.h>` exists and should be included.

`i_string`

From *i_string.U*:

This variable conditionally defines the `I_STRING` symbol, which indicates that `<string.h>` should be included rather than `<strings.h>`.

`i_sunmath`

From *i_sunmath.U*:

This variable conditionally defines the `I_SUNMATH` symbol, and indicates whether a C program should include `<sunmath.h>`.

`i_sysaccess`

From *i_sysaccess.U*:

This variable conditionally defines the `I_SYS_ACCESS` symbol, and indicates whether a C program should include `<sys/access.h>`.

`i_sysdir`

From *i_sysdir.U*:

This variable conditionally defines the `I_SYS_DIR` symbol, and indicates whether a C program should include `<sys/dir.h>`.

`i_sysfile`

From *i_sysfile.U*:

This variable conditionally defines the `I_SYS_FILE` symbol, and indicates whether a C program should include `<sys/file.h>` to get `R_OK` and friends.

`i_sysfilio`

From *i_sysioctl.U*:

This variable conditionally defines the `I_SYS_FILIO` symbol, which indicates to the C program that `<sys/filio.h>` exists and should be included in preference to `<sys/ioctl.h>`.

`i_sysin`

From *i_niin.U*:

This variable conditionally defines `I_SYS_IN`, which indicates to the C program that it should include `<sys/in.h>` instead of `<netinet/in.h>`.

`i_sysioctl`

From *i_sysioctl.U*:

This variable conditionally defines the `I_SYS_IOCTL` symbol, which indicates to the C program that `<sys/ioctl.h>` exists and should be included.

`i_syslog`

From *i_syslog.U*:

This variable conditionally defines the `I_SYSLOG` symbol, and indicates whether a C program should include `<syslog.h>`.

`i_sysmman`

From *i_sysmman.U*:

This variable conditionally defines the `I_SYS_MMAN` symbol, and indicates whether a C program should include `<sys/mman.h>`.

`i_sysmode`

From *i_sysmode.U*:

This variable conditionally defines the `I_SYSMODE` symbol, and indicates whether a C program should include `<sys/mode.h>`.

`i_sysmount`

From *i_sysmount.U*:

This variable conditionally defines the `I_SYSMOUNT` symbol, and indicates whether a C program should include `<sys/mount.h>`.

`i_sysndir`

From *i_sysndir.U*:

This variable conditionally defines the `I_SYS_NDIR` symbol, and indicates whether a C program should include `<sys/ndir.h>`.

`i_sysparam`

From *i_sysparam.U*:

This variable conditionally defines the `I_SYS_PARAM` symbol, and indicates whether a C program should include `<sys/param.h>`.

`i_syspoll`

From *i_syspoll.U*:

This variable conditionally defines the `I_SYS_POLL` symbol, which indicates to the C program that it should include `<sys/poll.h>`.

`i_sysresrc`

From *i_sysresrc.U*:

This variable conditionally defines the `I_SYS_RESOURCE` symbol, and indicates whether a C program should include `<sys/resource.h>`.

`i_sysseclt`

From *i_sysseclt.U*:

This variable conditionally defines the `I_SYS_SECURITY` symbol, and indicates whether a C program should include `<sys/security.h>`.

`i_sysselect`

From *i_sysselect.U*:

This variable conditionally defines `I_SYS_SELECT`, which indicates to the C program that it should include `<sys/select.h>` in order to get the definition of struct timeval.

`i_syssockio`

From *i_sysioctl.U*:

This variable conditionally defines `I_SYS_SOCKETIO` to indicate to the C program that socket ioctl codes may be found in `<sys/sockio.h>` instead of `<sys/ioctl.h>`.

`i_sysstat`

From *i_sysstat.U*:

This variable conditionally defines the `I_SYS_STAT` symbol, and indicates whether a C program should include `<sys/stat.h>`.

`i_sysstatfs`

From *i_sysstatfs.U*:

This variable conditionally defines the `I_SYSSTATFS` symbol, and indicates whether a C program should include `<sys/statfs.h>`.

`i_sysstatvfs`

From *i_sysstatvfs.U*:

This variable conditionally defines the `I_SYSSTATVFS` symbol, and indicates whether a C program should include `<sys/statvfs.h>`.

`i_systime`

From *i_time.U*:

This variable conditionally defines `I_SYS_TIME`, which indicates to the C program that it should include `<sys/time.h>`.

`i_systimek`

From *i_time.U*:

This variable conditionally defines `I_SYS_TIME_KERNEL`, which indicates to the C program that it should include `<sys/time.h>` with `KERNEL` defined.

`i_systimes`

From *i_systimes.U*:

This variable conditionally defines the `I_SYS_TIMES` symbol, and indicates whether a C program should include `<sys/times.h>`.

`i_systypes`

From *i_systypes.U*:

This variable conditionally defines the `I_SYS_TYPES` symbol, and indicates whether a C program should include `<sys/types.h>`.

`i_sysuio`

From *i_sysuio.U*:

This variable conditionally defines the `I_SYSUIO` symbol, and indicates whether a C program should include `<sys/uio.h>`.

`i_sysun`

From *i_sysun.U*:

This variable conditionally defines `I_SYS_UN`, which indicates to the C program that it should include `<sys/un.h>` to get UNIX domain socket definitions.

`i_sysutpname`

From *i_sysutpname.U*:

This variable conditionally defines the `I_SYSUTSNAME` symbol, and indicates whether a C program should include `<sys/utsname.h>`.

`i_sysvfs`

From *i_sysvfs.U*:

This variable conditionally defines the `I_SYSVFS` symbol, and indicates whether a C program should include `<sys/vfs.h>`.

`i_syswait`

From *i_syswait.U*:

This variable conditionally defines `I_SYS_WAIT`, which indicates to the C program that it should include `<sys/wait.h>`.

`i_termio`

From *i_termio.U*:

This variable conditionally defines the `I_TERMIO` symbol, which indicates to the C program that it should include `<termio.h>` rather than `<sgtty.h>`.

`i_termios`

From *i_termio.U*:

This variable conditionally defines the `I_TERMIOS` symbol, which indicates to the C program that the POSIX `<termios.h>` file is to be included.

`i_time`

From *i_time.U*:

This variable conditionally defines `I_TIME`, which indicates to the C program that it should include `<time.h>`.

`i_unistd`

From *i_unistd.U*:

This variable conditionally defines the `I_UNISTD` symbol, and indicates whether a C program should include `<unistd.h>`.

`i_ustat`

From *i_ustat.U*:

This variable conditionally defines the `I_USTAT` symbol, and indicates whether a C program should include `<ustat.h>`.

`i_utime`

From *i_utime.U*:

This variable conditionally defines the `I_UTIME` symbol, and indicates whether a C program should include `<utime.h>`.

`i_values`

From *i_values.U*:

This variable conditionally defines the `I_VALUES` symbol, and indicates whether a C program may include `<values.h>` to get symbols like `MAXLONG` and friends.

`i_varargs`

From *i_varhdr.U*:

This variable conditionally defines `I_VARARGS`, which indicates to the C program that it should include `<varargs.h>`.

`i_varhdr`

From *i_varhdr.U*:

Contains the name of the header to be included to get `va_dcl` definition. Typically one of *varargs.h* or *stdarg.h*.

`i_vfork`

From *i_vfork.U*:

This variable conditionally defines the `I_VFORK` symbol, and indicates whether a C program should include *vfork.h*.

`i_xlocale`

From *d_newlocale.U*:

This symbol, if defined, indicates to the C program that it should include `<xlocale.h>` to get `uselocale()` and its friends

`ignore_versioned_solibs`

From *libs.U*:

This variable should be non-empty if non-versioned shared libraries (*libfoo.so.x.y*) are to be ignored (because they cannot be linked against).

`inc_version_list`

From *inc_version_list.U*:

This variable specifies the list of subdirectories in over which *perl.c:incpush()* and *lib/lib.pm* will automatically search when adding directories to `@INC`. The elements in the list are separated by spaces. This is only useful if you have a perl library directory tree structured like the default one. See `INSTALL` for how this works. The versioned `site_perl` directory was introduced in 5.005, so that is the lowest possible value.

This list includes architecture-dependent directories back to version `$api_versionstring` (e.g. 5.5.640) and architecture-independent directories all the way back to 5.005.

`inc_version_list_init`

From *inc_version_list.U*:

This variable holds the same list as `inc_version_list`, but each item is enclosed in double quotes and separated by commas, suitable for use in the `PERL_INC_VERSION_LIST` initialization.

`incpath`

From *usrinc.U*:

This variable must precede the normal include path to get the right one, as in *\$incpath/usr/include* or *\$incpath/usr/lib*. Value can be "" or */bsd43* on mips.

`incpth`

From *libpth.U*:

This variable must precede the normal include path to get the right one, as in *\$incpath/usr/include* or *\$incpath/usr/lib*. Value can be "" or */bsd43* on mips.

`inews`

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

`initialinstalllocation`

From *bin.U*:

When `userelocatableinc` is true, this variable holds the location that make install should copy the perl binary to, with all the run-time relocatable paths calculated from this at install time. When used, it is initialized to the original value of `binexp`, and then `binexp` is set to `.../`, as the other binaries are found relative to the perl binary.

`installarchlib`

From *archlib.U*:

This variable is really the same as `archlibexp` but may differ on those systems using `AFS`. For extra portability, only this variable should be used in makefiles.

`installbin`

From *bin.U*:

This variable is the same as `binexp` unless `AFS` is running in which case the user is explicitly prompted for it. This variable should always be used in your makefiles for maximum portability.

`installhtml1dir`

From *html1dir.U*:

This variable is really the same as `html1direxp`, unless you are using a different `installprefix`. For extra portability, you should only use this variable within your makefiles.

`installhtml3dir`

From *html3dir.U*:

This variable is really the same as `html3direxp`, unless you are using a different `installprefix`. For extra portability, you should only use this variable within your makefiles.

`installman1dir`

From *man1dir.U*:

This variable is really the same as `man1direxp`, unless you are using `AFS` in which case it points to the read/write location whereas `man1direxp` only points to the read-only access location. For extra portability, you should only use this variable within your makefiles.

`installman3dir`

From *man3dir.U*:

This variable is really the same as `man3direxp`, unless you are using `AFS` in which case it points to the read/write location whereas `man3direxp` only points to the read-only access location. For extra portability, you should only use this variable within your makefiles.

`installprefix`

From *installprefix.U*:

This variable holds the name of the directory below which "make install" will install the package. For most users, this is the same as `prefix`. However, it is useful for installing the software into a different (usually temporary) location after which it can be bundled up and moved somehow to the final location specified by `prefix`.

`installprefixexp`

From *installprefix.U*:

This variable holds the full absolute path of `installprefix` with all `--expansion` done.

`installprivlib`

From *privlib.U*:

This variable is really the same as `privlibexp` but may differ on those systems using `AFS`. For extra portability, only this variable should be used in makefiles.

`installscript`

From *scriptdir.U*:

This variable is usually the same as `scriptdirexp`, unless you are on a system running `AFS`, in which case they may differ slightly. You should always use this variable within your makefiles for portability.

`installsitearch`

From *sitearch.U*:

This variable is really the same as `sitearchexp` but may differ on those systems using `AFS`. For extra portability, only this variable should be used in makefiles.

`installsitebin`

From *sitebin.U*:

This variable is usually the same as `sitebinexp`, unless you are on a system running `AFS`, in which case they may differ slightly. You should always use this variable within your makefiles for portability.

`installsitehtml1dir`

From *sitehtml1dir.U*:

This variable is really the same as `sitehtml1direxp`, unless you are using `AFS` in which case it points to the read/write location whereas `html1direxp` only points to the read-only access location. For extra portability, you should only use this variable within your makefiles.

`installsitehtml3dir`

From *sitehtml3dir.U*:

This variable is really the same as `sitehtml3direxp`, unless you are using AFS in which case it points to the read/write location whereas `html3direxp` only points to the read-only access location. For extra portability, you should only use this variable within your makefiles.

`installsitelib`

From *sitelib.U*:

This variable is really the same as `sitelibexp` but may differ on those systems using AFS. For extra portability, only this variable should be used in makefiles.

`installsiteman1dir`

From *siteman1dir.U*:

This variable is really the same as `siteman1direxp`, unless you are using AFS in which case it points to the read/write location whereas `man1direxp` only points to the read-only access location. For extra portability, you should only use this variable within your makefiles.

`installsiteman3dir`

From *siteman3dir.U*:

This variable is really the same as `siteman3direxp`, unless you are using AFS in which case it points to the read/write location whereas `man3direxp` only points to the read-only access location. For extra portability, you should only use this variable within your makefiles.

`installsitescript`

From *sitescript.U*:

This variable is usually the same as `sitescriptexp`, unless you are on a system running AFS, in which case they may differ slightly. You should always use this variable within your makefiles for portability.

`installstyle`

From *installstyle.U*:

This variable describes the `style` of the perl installation. This is intended to be useful for tools that need to manipulate entire perl distributions. Perl itself doesn't use this to find its libraries -- the library directories are stored directly in *Config.pm*. Currently, there are only two styles: `lib` and `lib/perl5`. The default library locations (e.g. `privlib`, `sitelib`) are either `$prefix/lib` or `$prefix/lib/perl5`. The former is useful if `$prefix` is a directory dedicated to perl (e.g. `/opt/perl`), while the latter is useful if `$prefix` is shared by many packages, e.g. if `$prefix=/usr/local`.

Unfortunately, while this `style` variable is used to set defaults for all three directory hierarchies (core, vendor, and site), there is no guarantee that the same style is actually appropriate for all those directories. For example, `$prefix` might be `/opt/perl`, but `$siteprefix` might be `/usr/local`. (Perhaps, in retrospect, the `lib` style should never have been supported, but it did seem like a nice idea at the time.)

The situation is even less clear for tools such as MakeMaker that can be used to install additional modules into non-standard places. For example, if a user intends to install a module into a private directory (perhaps by setting `PREFIX` on the *Makefile.PL* command line), then there is no reason to assume that the Configure-time `$installstyle` setting will be relevant for that `PREFIX`.

This may later be extended to include other information, so be careful with pattern-matching on the results.

For compatibility with *perl5.005* and earlier, the default setting is based on whether or not `$prefix` contains the string `perl`.

`installusrbinperl`

From *instubperl.U*:

This variable tells whether Perl should be installed also as `/usr/bin/perl` in addition to `$installbin/perl`

`installvendorarch`

From *vendorarch.U*:

This variable is really the same as `vendorarchexp` but may differ on those systems using AFS. For extra portability, only this variable should be used in makefiles.

`installvendorbin`

From *vendorbin.U*:

This variable is really the same as `vendorbinexp` but may differ on those systems using AFS. For extra portability, only this variable should be used in makefiles.

`installvendorhtml1dir`

From *vendorhtml1dir.U*:

This variable is really the same as `vendorhtml1direxp` but may differ on those systems using AFS. For extra portability, only this variable should be used in makefiles.

`installvendorhtml3dir`

From *vendorhtml3dir.U*:

This variable is really the same as `vendorhtml3direxp` but may differ on those systems using AFS. For extra portability, only this variable should be used in makefiles.

`installvendorlib`

From *vendorlib.U*:

This variable is really the same as `vendorlibexp` but may differ on those systems using AFS. For extra portability, only this variable should be used in makefiles.

`installvendorman1dir`

From *vendorman1dir.U*:

This variable is really the same as `vendorman1direxp` but may differ on those systems using AFS. For extra portability, only this variable should be used in makefiles.

`installvendorman3dir`

From *vendorman3dir.U*:

This variable is really the same as `vendorman3direxp` but may differ on those systems using AFS. For extra portability, only this variable should be used in makefiles.

`installvendorscript`

From *vendorscript.U*:

This variable is really the same as `vendorscriptexp` but may differ on those systems using AFS. For extra portability, only this variable should be used in makefiles.

`intsize`

From *intsize.U*:

This variable contains the value of the `INTSIZE` symbol, which indicates to the C program how many bytes there are in an int.

`issymlink`

From *issymlink.U*:

This variable holds the test command to test for a symbolic link (if they are supported). Typical

values include `test -h` and `test -L`.

`ivdformat`

From *perlxf.U*:

This variable contains the format string used for printing a Perl `IV` as a signed decimal integer.

`ivsize`

From *perlxv.U*:

This variable is the size of an `IV` in bytes.

`ivtype`

From *perlxv.U*:

This variable contains the C type used for Perl's `IV`.

k

`known_extensions`

From *Extensions.U*:

This variable holds a list of all extensions (both `XS` and non-`XS`) included in the package source distribution. This information is only really of use during the Perl build, as the list makes no distinction between extensions which were build and installed, and those which where not. See `extensions` for the list of extensions actually built and available.

`ksh`

From *Loc.U*:

This variable is defined but not used by `Configure`. The value is the empty string and is not useful.

l

`ld`

From *dlsrc.U*:

This variable indicates the program to be used to link libraries for dynamic loading. On some systems, it is `ld`. On `ELF` systems, it should be `$cc`. Mostly, we'll try to respect the hint file setting.

`ld_can_script`

From *dlsrc.U*:

This variable shows if the loader accepts scripts in the form of `-Wl,--version-script=ld.script`. This is currently only supported for `GNU ld` on `ELF` in dynamic loading builds.

`lddlflags`

From *dlsrc.U*:

This variable contains any special flags that might need to be passed to `$ld` to create a shared library suitable for dynamic loading. It is up to the makefile to use it. For `hpux`, it should be `-b`. For `sunos 4.1`, it is empty.

`ldflags`

From *ccflags.U*:

This variable contains any additional C loader flags desired by the user. It is up to the Makefile to use this.

`ldflags_uselargefiles`

From *uselfs.U*:

This variable contains the loader flags needed by large file builds and added to `ldflags` by hints files.

`ldlibpthname`

From *libperl.U*:

This variable holds the name of the shared library search path, often `LD_LIBRARY_PATH`. To get an empty string, the hints file must set this to `none`.

`less`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the `less` program. After Configure runs, the value is reset to a plain `less` and is not useful.

`lib_ext`

From *Unix.U*:

This is an old synonym for `_a`.

`libc`

From *libc.U*:

This variable contains the location of the C library.

`libperl`

From *libperl.U*:

The perl executable is obtained by linking *perlmain.c* with `libperl`, any static extensions (usually just `DynaLoader`), and any other libraries needed on this system. `libperl` is usually *libperl.a*, but can also be *libperl.so.xxx* if the user wishes to build a perl executable with a shared library.

`libpth`

From *libpth.U*:

This variable holds the general path (space-separated) used to find libraries. It is intended to be used by other units.

`libs`

From *libs.U*:

This variable holds the additional libraries we want to use. It is up to the Makefile to deal with it. The list can be empty.

`libsdirs`

From *libs.U*:

This variable holds the directory names aka `dirnames` of the libraries we found and accepted, duplicates are removed.

`libsfiles`

From *libs.U*:

This variable holds the filenames aka `basenames` of the libraries we found and accepted.

`libsfound`

From *libs.U*:

This variable holds the full pathnames of the libraries we found and accepted.

`libspath`

From *libs.U*:

This variable holds the directory names probed for libraries.

`libswanted`

From *Myinit.U*:

This variable holds a list of all the libraries we want to search. The order is chosen to pick up the `c` library ahead of `ucb` or `bsd` libraries for SVR4.

`libswanted_uselargefiles`

From *uselfs.U*:

This variable contains the libraries needed by large file builds and added to `ldflags` by hints files. It is a space separated list of the library names without the `lib` prefix or any suffix, just like *libswanted*..

`line`

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

`lint`

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

`lkflags`

From *ccflags.U*:

This variable contains any additional C partial linker flags desired by the user. It is up to the Makefile to use this.

`ln`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the `ln` program. After Configure runs, the value is reset to a plain `ln` and is not useful.

`lns`

From *Ins.U*:

This variable holds the name of the command to make symbolic links (if they are supported). It can be used in the Makefile. It is either `ln -s` or `ln`

`localtime_r_proto`

From *d_localtime_r.U*:

This variable encodes the prototype of `localtime_r`. It is zero if `d_localtime_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_localtime_r` is defined.

`locincpth`

From *ccflags.U*:

This variable contains a list of additional directories to be searched by the compiler. The appropriate `-I` directives will be added to `ccflags`. This is intended to simplify setting local directories from the Configure command line. It's not much, but it parallels the `loclibpth` stuff in *libpth.U*.

`loclibpth`

From *libpth.U*:

This variable holds the paths (space-separated) used to find local libraries. It is prepended to

libpth, and is intended to be easily set from the command line.

`longdblinfobytes`

From *infnan.U*:

This variable contains comma-separated list of hexadecimal bytes for the long double precision infinity.

`longdblkind`

From *d_longdbl.U*:

This variable, if defined, encodes the type of a long double: 0 = double, 1 = IEEE 754 128-bit little endian, 2 = IEEE 754 128-bit big endian, 3 = x86 80-bit little endian, 4 = x86 80-bit big endian, 5 = double-double 128-bit little endian, 6 = double-double 128-bit big endian, 7 = 128-bit mixed double-double (64-bit LEs in BE), 8 = 128-bit mixed double-double (64-bit BEs in LE), -1 = unknown format.

`longdblmanbits`

From *mantbits.U*:

This symbol, if defined, tells how many mantissa bits there are in long double precision floating point format. Note that this can be `LDBL_MANT_DIG` minus one, since `LDBL_MANT_DIG` can include the IEEE 754 implicit bit. The common x86-style 80-bit long double does not have an implicit bit.

`longdblnanbytes`

From *infnan.U*:

This variable contains comma-separated list of hexadecimal bytes for the long double precision not-a-number.

`longdblsize`

From *d_longdbl.U*:

This variable contains the value of the `LONG_DOUBLE_SIZE` symbol, which indicates to the C program how many bytes there are in a long double, if this system supports long doubles. Note that this is `sizeof(long double)`, which may include unused bytes.

`longlongsize`

From *d_longlong.U*:

This variable contains the value of the `LONG_LONG_SIZE` symbol, which indicates to the C program how many bytes there are in a long long, if this system supports long long.

`longsize`

From *intsize.U*:

This variable contains the value of the `LONG_SIZE` symbol, which indicates to the C program how many bytes there are in a long.

`lp`

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

`lpr`

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

`ls`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the `ls` program. After Configure runs, the value is reset to a plain `ls` and is not useful.

`lseeksize`

From *lseektype.U*:

This variable defines `lseektype` to be something like `off_t`, `long`, or whatever type is used to declare `lseek` offset's type in the kernel (which also appears to be `lseek`'s return type).

`lseektype`

From *lseektype.U*:

This variable defines `lseektype` to be something like `off_t`, `long`, or whatever type is used to declare `lseek` offset's type in the kernel (which also appears to be `lseek`'s return type).

m

`mail`

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

`mailx`

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

`make`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the `make` program. After Configure runs, the value is reset to a plain `make` and is not useful.

`make_set_make`

From *make.U*:

Some versions of `make` set the variable `MAKE`. Others do not. This variable contains the string to be included in *Makefile.SH* so that `MAKE` is set if needed, and not if not needed. Possible values are:

`make_set_make=#` # If your `make` program handles this for you,

`make_set_make=MAKE=$make` # if it doesn't.

This uses a comment character so that we can distinguish a `set` value (from a previous *config.sh* or Configure `-D` option) from an uncomputed value.

`mallocobj`

From *mallosrc.U*:

This variable contains the name of the *malloc.o* that this package generates, if that *malloc.o* is preferred over the system `malloc`. Otherwise the value is null. This variable is intended for generating Makefiles. See *mallosrc*.

`mallosrc`

From *mallosrc.U*:

This variable contains the name of the *malloc.c* that comes with the package, if that *malloc.c* is preferred over the system `malloc`. Otherwise the value is null. This variable is intended for

generating Makefiles.

`malloctype`

From *malloctype.U*:

This variable contains the kind of ptr returned by malloc and realloc.

`man1dir`

From *man1dir.U*:

This variable contains the name of the directory in which manual source pages are to be put. It is the responsibility of the *Makefile.SH* to get the value of this into the proper command. You must be prepared to do the *~name* expansion yourself.

`man1direxp`

From *man1dir.U*:

This variable is the same as the `man1dir` variable, but is filename expanded at configuration time, for convenient use in makefiles.

`man1ext`

From *man1dir.U*:

This variable contains the extension that the manual page should have: one of `n`, `1`, or `1`. The Makefile must supply the `..`. See `man1dir`.

`man3dir`

From *man3dir.U*:

This variable contains the name of the directory in which manual source pages are to be put. It is the responsibility of the *Makefile.SH* to get the value of this into the proper command. You must be prepared to do the *~name* expansion yourself.

`man3direxp`

From *man3dir.U*:

This variable is the same as the `man3dir` variable, but is filename expanded at configuration time, for convenient use in makefiles.

`man3ext`

From *man3dir.U*:

This variable contains the extension that the manual page should have: one of `n`, `1`, or `3`. The Makefile must supply the `..`. See `man3dir`.

`mips_type`

From *usrinc.U*:

This variable holds the environment type for the mips system. Possible values are "BSD 4.3" and "System V".

`mistrustnm`

From *Csym.U*:

This variable can be used to establish a fallback for the cases where `nm` fails to find a symbol. If `usenm` is false or `usenm` is true and `mistrustnm` is false, this variable has no effect. If `usenm` is true and `mistrustnm` is `compile`, a test program will be compiled to try to find any symbol that can't be located via `nm` lookup. If `mistrustnm` is `run`, the test program will be run as well as being compiled.

`mkdir`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the `mkdir` program. After Configure runs, the value is reset to a plain `mkdir` and is not useful.

`mmaptype`

From *d_mmap.U*:

This symbol contains the type of pointer returned by `mmap()` (and simultaneously the type of the first argument). It can be `void *` or `caddr_t`.

`modetype`

From *modetype.U*:

This variable defines `modetype` to be something like `mode_t`, `int`, unsigned short, or whatever type is used to declare file modes for system calls.

`more`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the `more` program. After Configure runs, the value is reset to a plain `more` and is not useful.

`multiarch`

From *multiarch.U*:

This variable conditionally defines the `MULTIARCH` symbol which signifies the presence of multiplatform files. This is normally set by hints files.

`mv`

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

`myarchname`

From *archname.U*:

This variable holds the architecture name computed by Configure in a previous run. It is not intended to be perused by any user and should never be set in a hint file.

`mydomain`

From *myhostname.U*:

This variable contains the eventual value of the `MYDOMAIN` symbol, which is the domain of the host the program is going to run on. The domain must be appended to `myhostname` to form a complete host name. The dot comes with `mydomain`, and need not be supplied by the program.

`myhostname`

From *myhostname.U*:

This variable contains the eventual value of the `MYHOSTNAME` symbol, which is the name of the host the program is going to run on. The domain is not kept with `hostname`, but must be gotten from `mydomain`. The dot comes with `mydomain`, and need not be supplied by the program.

`myuname`

From *Oldconfig.U*:

The output of `uname -a` if available, otherwise the `hostname`. The whole thing is then lower-cased and slashes and single quotes are removed.

n

n

From *n.U*:

This variable contains the `-n` flag if that is what causes the `echo` command to suppress newline. Otherwise it is null. Correct usage is `$echo $n "prompt for a question: $c"`.

`need_va_copy`

From *need_va_copy.U*:

This symbol, if defined, indicates that the system stores the variable argument list datatype, `va_list`, in a format that cannot be copied by simple assignment, so that some other means must be used when copying is required. As such systems vary in their provision (or non-provision) of copying mechanisms, *handy.h* defines a platform-independent macro, `Perl_va_copy(src, dst)`, to do the job.

`netdb_hlen_type`

From *netdbtype.U*:

This variable holds the type used for the 2nd argument to `gethostbyaddr()`. Usually, this is `int` or `size_t` or `unsigned`. This is only useful if you have `gethostbyaddr()`, naturally.

`netdb_host_type`

From *netdbtype.U*:

This variable holds the type used for the 1st argument to `gethostbyaddr()`. Usually, this is `char *` or `void *`, possibly with or without a `const` prefix. This is only useful if you have `gethostbyaddr()`, naturally.

`netdb_name_type`

From *netdbtype.U*:

This variable holds the type used for the argument to `gethostbyname()`. Usually, this is `char *` or `const char *`. This is only useful if you have `gethostbyname()`, naturally.

`netdb_net_type`

From *netdbtype.U*:

This variable holds the type used for the 1st argument to `getnetbyaddr()`. Usually, this is `int` or `long`. This is only useful if you have `getnetbyaddr()`, naturally.

nm

From *Loc.U*:

This variable is used internally by `Configure` to determine the full pathname (if any) of the `nm` program. After `Configure` runs, the value is reset to a plain `nm` and is not useful.

`nm_opt`

From *usenm.U*:

This variable holds the options that may be necessary for `nm`.

`nm_so_opt`

From *usenm.U*:

This variable holds the options that may be necessary for `nm` to work on a shared library but that can not be used on an archive library. Currently, this is only used by Linux, where `nm --dynamic` is *required* to get symbols from an `ELF` library which has been stripped, but `nm --dynamic` is *fatal* on an archive library. Maybe Linux should just always set `usenm=false`.

`nonxs_ext`

From *Extensions.U*:

This variable holds a list of all non-xs extensions built and installed by the package. By default, all non-xs extensions distributed will be built, with the exception of platform-specific extensions (currently only one VMS specific extension).

`nroff`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the `nroff` program. After Configure runs, the value is reset to a plain `nroff` and is not useful.

`nv_overflows_integers_at`

From *perlsv.U*:

This variable gives the largest integer value that NVs can hold as a constant floating point expression. If it could not be determined, it holds the value 0.

`nv_preserves_uv_bits`

From *perlsv.U*:

This variable indicates how many of bits type `uvtype` a variable `nvtype` can preserve.

`nveformat`

From *perlsvf.U*:

This variable contains the format string used for printing a Perl NV using %e-ish floating point format.

`nvEUformat`

From *perlsvf.U*:

This variable contains the format string used for printing a Perl NV using %E-ish floating point format.

`nvffformat`

From *perlsvf.U*:

This variable contains the format string used for printing a Perl NV using %f-ish floating point format.

`nvFUformat`

From *perlsvf.U*:

This variable contains the format string used for printing a Perl NV using %F-ish floating point format.

`nvformat`

From *perlsvf.U*:

This variable contains the format string used for printing a Perl NV using %g-ish floating point format.

`nvGUformat`

From *perlsvf.U*:

This variable contains the format string used for printing a Perl NV using %G-ish floating point format.

`nvmanbits`

From *manbits.U*:

This variable tells how many bits the mantissa of a Perl NV has, not including the possible

implicit bit.

`nvsize`

From *perlsv.U*:

This variable is the size of a Perl `NV` in bytes. Note that some floating point formats have unused bytes.

`nvtype`

From *perlsv.U*:

This variable contains the C type used for Perl's `NV`.

o

`o_nonblock`

From *nblock_io.U*:

This variable bears the symbol value to be used during `open()` or `fcntl()` to turn on non-blocking I/O for a file descriptor. If you wish to switch between blocking and non-blocking, you may try `ioctl(FIONBIO)` instead, but that is only supported by some devices.

`obj_ext`

From *Unix.U*:

This is an old synonym for `_o`.

`old_pthread_create_joinable`

From *d_pthratrj.U*:

This variable defines the constant to use for creating joinable (aka undetached) pthreads. Unused if *pthread.h* defines `PTHREAD_CREATE_JOINABLE`. If used, possible values are `PTHREAD_CREATE_UNDETACHED` and `__UNDETACHED`.

`optimize`

From *ccflags.U*:

This variable contains any *optimizer/debugger* flag that should be used. It is up to the Makefile to use it.

`orderlib`

From *orderlib.U*:

This variable is `true` if the components of libraries must be ordered (with ``lorder $* | tsort``) before placing them in an archive. Set to `false` if `ranlib` or `ar` can generate random libraries.

`osname`

From *Oldconfig.U*:

This variable contains the operating system name (e.g. `sunos`, `solaris`, `hpux`, etc.). It can be useful later on for setting defaults. Any spaces are replaced with underscores. It is set to a null string if we can't figure it out.

`osvers`

From *Oldconfig.U*:

This variable contains the operating system version (e.g. `4.1.3`, `5.2`, etc.). It is primarily used for helping select an appropriate hints file, but might be useful elsewhere for setting defaults. It is set to `"` if we can't figure it out. We try to be flexible about how much of the version number to keep, e.g. if `4.1.1`, `4.1.2`, and `4.1.3` are essentially the same for this package, hints files might just be `os_4.0` or `os_4.1`, etc., not keeping separate files for each little release.

`otherlibdirs`

From *otherlibdirs.U*:

This variable contains a colon-separated set of paths for the perl binary to search for additional library files or modules. These directories will be tacked to the end of `@INC`. Perl will automatically search below each path for version- and architecture-specific directories. See `inc_version_list` for more details. A value of `none` means `none` and is used to preserve this value for the next run through Configure.

p

`package`

From *package.U*:

This variable contains the name of the package being constructed. It is primarily intended for the use of later Configure units.

`pager`

From *pager.U*:

This variable contains the name of the preferred pager on the system. Usual values are (the full pathnames of) `more`, `less`, `pg`, or `cat`.

`passcat`

From *nis.U*:

This variable contains a command that produces the text of the `/etc/passwd` file. This is normally `"cat /etc/passwd"`, but can be `"ypcat passwd"` when `NIS` is used. On some systems, such as `os390`, there may be no equivalent command, in which case this variable is unset.

`patchlevel`

From *patchlevel.U*:

The patchlevel level of this package. The value of `patchlevel` comes from the *patchlevel.h* file. In a version number such as 5.6.1, this is the 6. In *patchlevel.h*, this is referred to as `PERL_VERSION`.

`path_sep`

From *Unix.U*:

This is an old synonym for `p_` in *Head.U*, the character used to separate elements in the command shell search `PATH`.

`perl`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the perl program. After Configure runs, the value is reset to a plain `perl` and is not useful.

`perl5`

From *perl5.U*:

This variable contains the full path (if any) to a previously installed *perl5.005* or later suitable for running the script to determine `inc_version_list`.

P

`PERL_API_REVISION`

From *patchlevel.h*:

This number describes the earliest compatible `PERL_REVISION` of Perl (compatibility here being defined as sufficient *binary/API* compatibility to run `XS` code built with the older version). Normally this does not change across maintenance releases. Please read the comment in *patchlevel.h*.

PERL_API_SUBVERSION

From *patchlevel.h*:

This number describes the earliest compatible PERL_SUBVERSION of Perl (compatibility here being defined as sufficient *binary/API* compatibility to run XS code built with the older version). Normally this does not change across maintenance releases. Please read the comment in *patchlevel.h*.

PERL_API_VERSION

From *patchlevel.h*:

This number describes the earliest compatible PERL_VERSION of Perl (compatibility here being defined as sufficient *binary/API* compatibility to run XS code built with the older version). Normally this does not change across maintenance releases. Please read the comment in *patchlevel.h*.

PERL_CONFIG_SH

From *Oldsyms.U*:

This is set to `true` in *config.sh* so that a shell script sourcing *config.sh* can tell if it has been sourced already.

PERL_PATCHLEVEL

From *Oldsyms.U*:

This symbol reflects the patchlevel, if available. Will usually come from the *.patch* file, which is available when the perl source tree was fetched with *rsync*.

perl_patchlevel

From *patchlevel.U*:

This is the Perl patch level, a numeric change identifier, as defined by whichever source code maintenance system is used to maintain the patches; currently Perforce. It does not correlate with the Perl version numbers or the maintenance versus development dichotomy except by also being increasing.

PERL_REVISION

From *Oldsyms.U*:

In a Perl version number such as 5.6.2, this is the 5. This value is manually set in *patchlevel.h*

perl_static_inline

From *d_static_inline.U*:

This variable defines the PERL_STATIC_INLINE symbol to the best-guess incantation to use for static inline functions. Possibilities include `static inline` (c99) `static __inline__` (gcc -ansi) `static __inline` (MSVC) `static _inline` (older MSVC) `static` (c89 compilers)

PERL_SUBVERSION

From *Oldsyms.U*:

In a Perl version number such as 5.6.2, this is the 2. Values greater than 50 represent potentially unstable development subversions. This value is manually set in *patchlevel.h*

PERL_VERSION

From *Oldsyms.U*:

In a Perl version number such as 5.6.2, this is the 6. This value is manually set in *patchlevel.h*

perladmin

From *perladmin.U*:

Electronic mail address of the perl5 administrator.

perllibs

From *End.U*:

The list of libraries needed by Perl only (any libraries needed by extensions only will be dropped, if using dynamic loading).

perlpith

From *perlpith.U*:

This variable contains the eventual value of the `PERLPATH` symbol, which contains the name of the perl interpreter to be used in shell scripts and in the "eval `exec`" idiom. This variable is not necessarily the pathname of the file containing the perl interpreter; you must append the executable extension (`_exe`) if it is not already present. Note that Perl code that runs during the Perl build process cannot reference this variable, as Perl may not have been installed, or even if installed, may be a different version of Perl.

pg

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the `pg` program. After Configure runs, the value is reset to a plain `pg` and is not useful.

phostname

From *myhostname.U*:

This variable contains the eventual value of the `PHOSTNAME` symbol, which is a command that can be fed to `popen()` to get the host name. The program should probably not presume that the domain is or isn't there already.

pidtype

From *pidtype.U*:

This variable defines `PIDTYPE` to be something like `pid_t`, `int`, `ushort`, or whatever type is used to declare process ids in the kernel.

plibpth

From *libpth.U*:

Holds the private path used by Configure to find out the libraries. Its value is prepended to `libpth`. This variable takes care of special machines, like the mips. Usually, it should be empty.

pmake

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

pr

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

prefix

From *prefix.U*:

This variable holds the name of the directory below which the user will install the package. Usually, this is `/usr/local`, and executables go in `/usr/local/bin`, library stuff in `/usr/local/lib`, man pages in `/usr/local/man`, etc. It is only used to set defaults for things in *bin.U*, *mansrc.U*, *privlib.U*, or *scriptdir.U*.

prefixexp

From *prefix.U*:

This variable holds the full absolute path of the directory below which the user will install the package. Derived from `prefix`.

privlib

From *privlib.U*:

This variable contains the eventual value of the `PRIVLIB` symbol, which is the name of the private library for this package. It may have a `~` on the front. It is up to the makefile to eventually create this directory while performing installation (with `~` substitution).

privlibexp

From *privlib.U*:

This variable is the `~name` expanded version of `privlib`, so that you may use it directly in Makefiles or shell scripts.

proclselfexe

From *d_proclselfexe.U*:

If `d_proclselfexe` is defined, `$proclselfexe` is the filename of the symbolic link pointing to the absolute pathname of the executing program.

prototype

From *prototype.U*:

This variable holds the eventual value of `CAN_PROTOTYPE`, which indicates the C compiler can handle function prototypes.

ptrsize

From *ptrsize.U*:

This variable contains the value of the `PTRSIZE` symbol, which indicates to the C program how many bytes there are in a pointer.

q

quadkind

From *quadtype.U*:

This variable, if defined, encodes the type of a quad: 1 = int, 2 = long, 3 = long long, 4 = `int64_t`.

quadtype

From *quadtype.U*:

This variable defines `Quad_t` to be something like long, int, long long, `int64_t`, or whatever type is used for 64-bit integers.

r

randbits

From *randfunc.U*:

Indicates how many bits are produced by the function used to generate normalized random numbers.

randfunc

From *randfunc.U*:

Indicates the name of the random number function to use. Values include `drand48`, `random`, and `rand`. In C programs, the `Drand01` macro is defined to generate uniformly distributed

random numbers over the range [0., 1.] (see `drand01` and `nrand`).

`random_r_proto`

From *d_random_r.U*:

This variable encodes the prototype of `random_r`. It is zero if `d_random_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_random_r` is defined.

`randseedtype`

From *randfunc.U*:

Indicates the type of the argument of the `seedfunc`.

`ranlib`

From *orderlib.U*:

This variable is set to the pathname of the `ranlib` program, if it is needed to generate random libraries. Set to `:` if `ar` can generate random libraries or if random libraries are not supported

`rd_nodata`

From *nblock_io.U*:

This variable holds the return code from `read()` when no data is present. It should be `-1`, but some systems return `0` when `O_NDELAY` is used, which is a shame because you cannot make the difference between no data and an *EOF*.. Sigh!

`readdir64_r_proto`

From *d_readdir64_r.U*:

This variable encodes the prototype of `readdir64_r`. It is zero if `d_readdir64_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_readdir64_r` is defined.

`readdir_r_proto`

From *d_readdir_r.U*:

This variable encodes the prototype of `readdir_r`. It is zero if `d_readdir_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_readdir_r` is defined.

`revision`

From *patchlevel.U*:

The value of `revision` comes from the *patchlevel.h* file. In a version number such as `5.6.1`, this is the `5`. In *patchlevel.h*, this is referred to as `PERL_REVISION`.

`rm`

From *Loc.U*:

This variable is used internally by `Configure` to determine the full pathname (if any) of the `rm` program. After `Configure` runs, the value is reset to a plain `rm` and is not useful.

`rm_try`

From *Unix.U*:

This is a cleanup variable for try test programs. Internal `Configure` use only.

`rmail`

From *Loc.U*:

This variable is defined but not used by `Configure`. The value is the empty string and is not useful.

`run`

From *Cross.U*:

This variable contains the command used by Configure to copy and execute a cross-compiled executable in the target host. Useful and available only during Perl build. Empty string "" if not cross-compiling.

runnm

From *usenm.U*:

This variable contains `true` or `false` depending whether the `nm` extraction should be performed or not, according to the value of `usenm` and the flags on the Configure command line.

S

sched_yield

From *d_pthread_y.U*:

This variable defines the way to yield the execution of the current thread.

scriptdir

From *scriptdir.U*:

This variable holds the name of the directory in which the user wants to put publicly scripts for the package in question. It is either the same directory as for binaries, or a special one that can be mounted across different architectures, like `/usr/share`. Programs must be prepared to deal with `~name` expansion.

scriptdirexp

From *scriptdir.U*:

This variable is the same as `scriptdir`, but is filename expanded at configuration time, for programs not wanting to bother with it.

sed

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the `sed` program. After Configure runs, the value is reset to a plain `sed` and is not useful.

seedfunc

From *randfunc.U*:

Indicates the random number generating seed function. Values include `srand48`, `srandom`, and `srand`.

selectminbits

From *selectminbits.U*:

This variable holds the minimum number of bits operated by `select`. That is, if you do `select(n, ...)`, how many bits at least will be cleared in the masks if some activity is detected. Usually this is either `n` or `32*ceil(n/32)`, especially many little-endians do the latter. This is only useful if you have `select()`, naturally.

selecttype

From *selecttype.U*:

This variable holds the type used for the 2nd, 3rd, and 4th arguments to `select`. Usually, this is `fd_set *`, if `HAS_FD_SET` is defined, and `int *` otherwise. This is only useful if you have `select()`, naturally.

sendmail

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not

useful

From *d_setgrent_r.U*:

This variable encodes the prototype of `setgrent_r`. It is zero if `d_setgrent_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_setgrent_r` is defined.

sethostent_r_proto

From *d_sethostent_r.U*:

This variable encodes the prototype of `sethostent_r`. It is zero if `d_sethostent_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_sethostent_r` is defined.

setlocale_r_proto

From *d_setlocale_r.U*:

This variable encodes the prototype of `setlocale_r`. It is zero if `d_setlocale_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_setlocale_r` is defined.

setnetent_r_proto

From *d_setnetent_r.U*:

This variable encodes the prototype of `setnetent_r`. It is zero if `d_setnetent_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_setnetent_r` is defined.

setprotoent_r_proto

From *d_setprotoent_r.U*:

This variable encodes the prototype of `setprotoent_r`. It is zero if `d_setprotoent_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_setprotoent_r` is defined.

setpwent_r_proto

From *d_setpwent_r.U*:

This variable encodes the prototype of `setpwent_r`. It is zero if `d_setpwent_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_setpwent_r` is defined.

setservent_r_proto

From *d_setservent_r.U*:

This variable encodes the prototype of `setservent_r`. It is zero if `d_setservent_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_setservent_r` is defined.

sGMTIME_max

From *time_size.U*:

This variable defines the maximum value of the `time_t` offset that the system function `gmtime()` accepts

sGMTIME_min

From *time_size.U*:

This variable defines the minimum value of the `time_t` offset that the system function `gmtime()` accepts

sh

From *sh.U*:

This variable contains the full pathname of the shell used on this system to execute Bourne shell scripts. Usually, this will be `/bin/sh`, though it's possible that some systems will have `/bin/ksh`, `/bin/pdksh`, `/bin/ash`, `/bin/bash`, or even something such as `D:/bin/sh.exe`. This unit comes before *Options.U*, so you can't set `sh` with a `-D` option, though you can override this (and `startsh`) with `-O -Dsh=/bin/whatever -Dstartsh=whatever`

shar

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

sharpbang

From *spitshell.U*:

This variable contains the string `#!` if this system supports that construct.

shmatttype

From *d_shmat.U*:

This symbol contains the type of pointer returned by `shmat()`. It can be `void *` or `char *`.

shortsize

From *intsize.U*:

This variable contains the value of the `SHORTSIZE` symbol which indicates to the C program how many bytes there are in a short.

shrpenv

From *libperl.U*:

If the user builds a shared *libperl.so*, then we need to tell the `perl` executable where it will be able to find the installed *libperl.so*. One way to do this on some systems is to set the environment variable `LD_RUN_PATH` to the directory that will be the final location of the shared *libperl.so*. The makefile can use this with something like `$shrpenv $(CC) -o perl perlmain.o $libperl $libs`. Typical values are `shrpenv="env LD_RUN_PATH=$archlibexp/CORE"` or `shrpenv=""`. See the main perl *Makefile.SH* for actual working usage. Alternatively, we might be able to use a command line option such as `-R $archlibexp/CORE` (Solaris) or `-Wl,-rpath $archlibexp/CORE` (Linux).

shsharp

From *spitshell.U*:

This variable tells further Configure units whether your `sh` can handle `#` comments.

sig_count

From *sig_name.U*:

This variable holds a number larger than the largest valid signal number. This is usually the same as the `NSIG` macro.

sig_name

From *sig_name.U*:

This variable holds the signal names, space separated. The leading `SIG` in signal name is removed. A `ZERO` is prepended to the list. This is currently not used, `sig_name_init` is used instead.

sig_name_init

From *sig_name.U*:

This variable holds the signal names, enclosed in double quotes and separated by commas, suitable for use in the `SIG_NAME` definition below. A `ZERO` is prepended to the list, and the list is terminated with a plain `0`. The leading `SIG` in signal names is removed. See `sig_num`.

sig_num

From *sig_name.U*:

This variable holds the signal numbers, space separated. A `ZERO` is prepended to the list (corresponding to the fake `SIGZERO`). Those numbers correspond to the value of the signal listed in the same place within the `sig_name` list. This is currently not used, `sig_num_init` is used instead.

`sig_num_init`

From *sig_name.U*:

This variable holds the signal numbers, enclosed in double quotes and separated by commas, suitable for use in the `SIG_NUM` definition below. A `ZERO` is prepended to the list, and the list is terminated with a plain `0`.

`sig_size`

From *sig_name.U*:

This variable contains the number of elements of the `sig_name` and `sig_num` arrays.

`signal_t`

From *d_voidsig.U*:

This variable holds the type of the signal handler (void or int).

`sitearch`

From *sitearch.U*:

This variable contains the eventual value of the `SITEARCH` symbol, which is the name of the private library for this package. It may have a `~` on the front. It is up to the makefile to eventually create this directory while performing installation (with `~` substitution). The standard distribution will put nothing in this directory. After perl has been installed, users may install their own local architecture-dependent modules in this directory with MakeMaker *Makefile.PL* or equivalent. See `INSTALL` for details.

`sitearchexp`

From *sitearch.U*:

This variable is the `~name` expanded version of `sitearch`, so that you may use it directly in Makefiles or shell scripts.

`sitebin`

From *sitebin.U*:

This variable holds the name of the directory in which the user wants to put add-on publicly executable files for the package in question. It is most often a local directory such as `/usr/local/bin`. Programs using this variable must be prepared to deal with `~name` substitution. The standard distribution will put nothing in this directory. After perl has been installed, users may install their own local executables in this directory with MakeMaker *Makefile.PL* or equivalent. See `INSTALL` for details.

`sitebinexp`

From *sitebin.U*:

This is the same as the `sitebin` variable, but is filename expanded at configuration time, for use in your makefiles.

`sitehtmlldir`

From *sitehtml1dir.U*:

This variable contains the name of the directory in which site-specific html source pages are to be put. It is the responsibility of the *Makefile.SH* to get the value of this into the proper command. You must be prepared to do the `~name` expansion yourself. The standard distribution will put nothing in this directory. After perl has been installed, users may install

their own local html pages in this directory with MakeMaker *Makefile.PL* or equivalent. See `INSTALL` for details.

`sitehtml1direxp`

From *sitehtml1dir.U*:

This variable is the same as the `sitehtml1dir` variable, but is filename expanded at configuration time, for convenient use in makefiles.

`sitehtml3dir`

From *sitehtml3dir.U*:

This variable contains the name of the directory in which site-specific library html source pages are to be put. It is the responsibility of the *Makefile.SH* to get the value of this into the proper command. You must be prepared to do the *~name* expansion yourself. The standard distribution will put nothing in this directory. After perl has been installed, users may install their own local library html pages in this directory with MakeMaker *Makefile.PL* or equivalent. See `INSTALL` for details.

`sitehtml3direxp`

From *sitehtml3dir.U*:

This variable is the same as the `sitehtml3dir` variable, but is filename expanded at configuration time, for convenient use in makefiles.

`sitelib`

From *sitelib.U*:

This variable contains the eventual value of the `SITELIB` symbol, which is the name of the private library for this package. It may have a `~` on the front. It is up to the makefile to eventually create this directory while performing installation (with `~` substitution). The standard distribution will put nothing in this directory. After perl has been installed, users may install their own local architecture-independent modules in this directory with MakeMaker *Makefile.PL* or equivalent. See `INSTALL` for details.

`sitelib_stem`

From *sitelib.U*:

This variable is `$sitelibexp` with any trailing version-specific component removed. The elements in `inc_version_list` (*inc_version_list.U*) can be tacked onto this variable to generate a list of directories to search.

`sitelibexp`

From *sitelib.U*:

This variable is the *~name* expanded version of `sitelib`, so that you may use it directly in Makefiles or shell scripts.

`siteman1dir`

From *siteman1dir.U*:

This variable contains the name of the directory in which site-specific manual source pages are to be put. It is the responsibility of the *Makefile.SH* to get the value of this into the proper command. You must be prepared to do the *~name* expansion yourself. The standard distribution will put nothing in this directory. After perl has been installed, users may install their own local man1 pages in this directory with MakeMaker *Makefile.PL* or equivalent. See `INSTALL` for details.

`siteman1direxp`

From *siteman1dir.U*:

This variable is the same as the `siteman1dir` variable, but is filename expanded at configuration time, for convenient use in makefiles.

`siteman3dir`

From *siteman3dir.U*:

This variable contains the name of the directory in which site-specific library man source pages are to be put. It is the responsibility of the *Makefile.SH* to get the value of this into the proper command. You must be prepared to do the *~name* expansion yourself. The standard distribution will put nothing in this directory. After perl has been installed, users may install their own local man3 pages in this directory with MakeMaker *Makefile.PL* or equivalent. See `INSTALL` for details.

`siteman3direxp`

From *siteman3dir.U*:

This variable is the same as the `siteman3dir` variable, but is filename expanded at configuration time, for convenient use in makefiles.

`siteprefix`

From *siteprefix.U*:

This variable holds the full absolute path of the directory below which the user will install add-on packages. See `INSTALL` for usage and examples.

`siteprefixexp`

From *siteprefix.U*:

This variable holds the full absolute path of the directory below which the user will install add-on packages. Derived from `siteprefix`.

`sitescript`

From *sitescript.U*:

This variable holds the name of the directory in which the user wants to put add-on publicly executable files for the package in question. It is most often a local directory such as `/usr/local/bin`. Programs using this variable must be prepared to deal with *~name* substitution. The standard distribution will put nothing in this directory. After perl has been installed, users may install their own local scripts in this directory with MakeMaker *Makefile.PL* or equivalent. See `INSTALL` for details.

`sitescriptexp`

From *sitescript.U*:

This is the same as the `sitescript` variable, but is filename expanded at configuration time, for use in your makefiles.

`size`

From *size.U*:

This variable contains the size of a `sizetype` in bytes.

`sizetype`

From *sizetype.U*:

This variable defines `sizetype` to be something like `size_t`, unsigned long, or whatever type is used to declare length parameters for string functions.

`sleep`

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not

`sLOCALTIME_max`

From *time_size.U*:

This variable defines the maximum value of the `time_t` offset that the system function `localtime()` accepts

`sLOCALTIME_min`

From *time_size.U*:

This variable defines the minimum value of the `time_t` offset that the system function `localtime()` accepts

`smaill`

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

`so`

From *so.U*:

This variable holds the extension used to identify shared libraries (also known as shared objects) on the system. Usually set to `so`.

`sockethdr`

From *d_socket.U*:

This variable has any `cpp -I` flags needed for socket support.

`socketlib`

From *d_socket.U*:

This variable has the names of any libraries needed for socket support.

`socksizetype`

From *socksizetype.U*:

This variable holds the type used for the size argument for various socket calls like `accept`. Usual values include `socklen_t`, `size_t`, and `int`.

`sort`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the sort program. After Configure runs, the value is reset to a plain `sort` and is not useful.

`spackage`

From *package.U*:

This variable contains the name of the package being constructed, with the first letter uppercased, *i.e.* suitable for starting sentences.

`spitshell`

From *spitshell.U*:

This variable contains the command necessary to spit out a runnable shell on this system. It is either `cat` or a `grep -v` for `#` comments.

`sPRId64`

From *quadfio.U*:

This variable, if defined, contains the string used by `stdio` to format 64-bit decimal numbers (format `d`) for output.

`sPRIeldbl`

From *longdblfiio.U*:

This variable, if defined, contains the string used by stdio to format long doubles (format `e`) for output.

`sPRIEUldbl`

From *longdblfiio.U*:

This variable, if defined, contains the string used by stdio to format long doubles (format `E`) for output. The `U` in the name is to separate this from `sPRIeldbl` so that even case-blind systems can see the difference.

`sPRIfldbl`

From *longdblfiio.U*:

This variable, if defined, contains the string used by stdio to format long doubles (format `f`) for output.

`sPRIFUldbl`

From *longdblfiio.U*:

This variable, if defined, contains the string used by stdio to format long doubles (format `F`) for output. The `U` in the name is to separate this from `sPRIfldbl` so that even case-blind systems can see the difference.

`sPRIgldbl`

From *longdblfiio.U*:

This variable, if defined, contains the string used by stdio to format long doubles (format `g`) for output.

`sPRIGUldbl`

From *longdblfiio.U*:

This variable, if defined, contains the string used by stdio to format long doubles (format `G`) for output. The `U` in the name is to separate this from `sPRIgldbl` so that even case-blind systems can see the difference.

`sPRIi64`

From *quadfiio.U*:

This variable, if defined, contains the string used by stdio to format 64-bit decimal numbers (format `i`) for output.

`sPRIo64`

From *quadfiio.U*:

This variable, if defined, contains the string used by stdio to format 64-bit octal numbers (format `o`) for output.

`sPRIu64`

From *quadfiio.U*:

This variable, if defined, contains the string used by stdio to format 64-bit unsigned decimal numbers (format `u`) for output.

`sPRIx64`

From *quadfiio.U*:

This variable, if defined, contains the string used by stdio to format 64-bit hexadecimal numbers (format `x`) for output.

sPRIXU64

From *quadfio.U*:

This variable, if defined, contains the string used by stdio to format 64-bit hExADECimAl numbers (format *x*) for output. The *U* in the name is to separate this from *sPRIx64* so that even case-blind systems can see the difference.

srand48_r_proto

From *d_srand48_r.U*:

This variable encodes the prototype of *srand48_r*. It is zero if *d_srand48_r* is undef, and one of the *REENTRANT_PROTO_T_ABC* macros of *reentr.h* if *d_srand48_r* is defined.

srandom_r_proto

From *d_srandom_r.U*:

This variable encodes the prototype of *srandom_r*. It is zero if *d_srandom_r* is undef, and one of the *REENTRANT_PROTO_T_ABC* macros of *reentr.h* if *d_srandom_r* is defined.

src

From *src.U*:

This variable holds the (possibly relative) path of the package source. It is up to the Makefile to use this variable and set *VPATH* accordingly to find the sources remotely. Use *\$pkgsrc* to have an absolute path.

sSCNfldbl

From *longdblfiio.U*:

This variable, if defined, contains the string used by stdio to format long doubles (format *f*) for input.

ssize_t

From *ssize_t.U*:

This variable defines *ssize_t* to be something like *ssize_t*, *long* or *int*. It is used by functions that return a count of bytes or an error condition. It must be a signed type. We will pick a type such that *sizeof(SSize_t) == sizeof(Size_t)*.

st_ino_sign

From *st_ino_def.U*:

This variable contains the signedness of struct *stat*'s *st_ino*. 1 for unsigned, -1 for signed.

st_ino_size

From *st_ino_def.U*:

This variable contains the size of struct *stat*'s *st_ino* in bytes.

startperl

From *startperl.U*:

This variable contains the string to put on the front of a perl script to make sure (hopefully) that it runs with perl and not some shell. Of course, that leading line must be followed by the classical perl idiom: *eval 'exec perl -S \$0 \${1+\${@}}' if \$running_under_some_shell*; to guarantee perl startup should the shell execute the script. Note that this magic incantation is not understood by *csh*.

startsh

From *startsh.U*:

This variable contains the string to put on the front of a shell script to make sure (hopefully)

that it runs with sh and not some other shell.

`static_ext`

From *Extensions.U*:

This variable holds a list of XS extension files we want to link statically into the package. It is used by Makefile.

`stdchar`

From *stdchar.U*:

This variable conditionally defines `STDCHAR` to be the type of char used in *stdio.h*. It has the values "unsigned char" or `char`.

`stdio_base`

From *d_stdstdio.U*:

This variable defines how, given a `FILE` pointer, `fp`, to access the `_base` field (or equivalent) of *stdio.h*'s `FILE` structure. This will be used to define the macro `FILE_base(fp)`.

`stdio_bufsiz`

From *d_stdstdio.U*:

This variable defines how, given a `FILE` pointer, `fp`, to determine the number of bytes store in the I/O buffer pointer to by the `_base` field (or equivalent) of *stdio.h*'s `FILE` structure. This will be used to define the macro `FILE_bufsiz(fp)`.

`stdio_cnt`

From *d_stdstdio.U*:

This variable defines how, given a `FILE` pointer, `fp`, to access the `_cnt` field (or equivalent) of *stdio.h*'s `FILE` structure. This will be used to define the macro `FILE_cnt(fp)`.

`stdio_filbuf`

From *d_stdstdio.U*:

This variable defines how, given a `FILE` pointer, `fp`, to tell stdio to refill its internal buffers (?). This will be used to define the macro `FILE_filbuf(fp)`.

`stdio_ptr`

From *d_stdstdio.U*:

This variable defines how, given a `FILE` pointer, `fp`, to access the `_ptr` field (or equivalent) of *stdio.h*'s `FILE` structure. This will be used to define the macro `FILE_ptr(fp)`.

`stdio_stream_array`

From *stdio_streams.U*:

This variable tells the name of the array holding the stdio streams. Usual values include `_job`, `__job`, and `__sF`.

`strerror_r_proto`

From *d_strerror_r.U*:

This variable encodes the prototype of `strerror_r`. It is zero if `d_strerror_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_strerror_r` is defined.

`strings`

From *i_string.U*:

This variable holds the full path of the string header that will be used. Typically `/usr/include/string.h` or `/usr/include/strings.h`.

submit

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

subversion

From *patchlevel.U*:

The subversion level of this package. The value of subversion comes from the *patchlevel.h* file. In a version number such as 5.6.1, this is the 1. In *patchlevel.h*, this is referred to as `PERL_SUBVERSION`. This is unique to perl.

sysman

From *sysman.U*:

This variable holds the place where the manual is located on this system. It is not the place where the user wants to put his manual pages. Rather it is the place where Configure may look to find manual for unix commands (section 1 of the manual usually). See `mansrc`.

sysroot

From *Sysroot.U*:

This variable is empty unless supplied by the Configure user. It can contain a path to an alternative root directory, under which headers and libraries for the compilation target can be found. This is generally used when cross-compiling using a gcc-like compiler.

t

tail

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

tar

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

targetarch

From *Cross.U*:

If cross-compiling, this variable contains the target architecture. If not, this will be empty.

targetdir

From *Cross.U*:

This variable contains a path that will be created on the target host using `targetmkdir`, and then used to copy the cross-compiled executables to. Defaults to `/tmp` if not set.

targetenv

From *Cross.U*:

If cross-compiling, this variable can be used to modify the environment on the target system. However, how and where it's used, and even if it's used at all, is entirely dependent on both the transport mechanism (`targetrun`) and what the target system is. Unless the relevant documentation says otherwise, it is generally not useful.

targethost

From *Cross.U*:

This variable contains the name of a separate host machine that can be used to run compiled test programs and perl tests on. Set to empty string if not in use.

`targetmkdir`

From *Cross.U*:

This variable contains the command used by Configure to create a new directory on the target host.

`targetport`

From *Cross.U*:

This variable contains the number of a network port to be used to connect to the host in `targethost`, if unset defaults to 22 for ssh.

`targetsh`

From *sh.U*:

If cross-compiling, this variable contains the location of sh on the target system. If not, this will be the same as `$sh`.

`tbl`

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

`tee`

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

`test`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the test program. After Configure runs, the value is reset to a plain `test` and is not useful.

`timeincl`

From *i_time.U*:

This variable holds the full path of the included time header(s).

`timetype`

From *d_time.U*:

This variable holds the type returned by `time()`. It can be `long`, or `time_t` on BSD sites (in which case `<sys/types.h>` should be included). Anyway, the type `Time_t` should be used.

`tmpnam_r_proto`

From *d_tmpnam_r.U*:

This variable encodes the prototype of `tmpnam_r`. It is zero if `d_tmpnam_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_tmpnam_r` is defined.

`to`

From *Cross.U*:

This variable contains the command used by Configure to copy to from the target host. Useful and available only during Perl build. The string `:` if not cross-compiling.

`touch`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the touch program. After Configure runs, the value is reset to a plain `touch` and is not useful.

`tr`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the `tr` program. After Configure runs, the value is reset to a plain `tr` and is not useful.

`trnl`

From *trnl.U*:

This variable contains the value to be passed to the `tr(1)` command to transliterate a newline. Typical values are `\012` and `\n`. This is needed for EBCDIC systems where newline is not necessarily `\012`.

`troff`

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

`ttyname_r_proto`

From *d_ttyname_r.U*:

This variable encodes the prototype of `ttyname_r`. It is zero if `d_ttyname_r` is undef, and one of the `REENTRANT_PROTO_T_ABC` macros of *reentr.h* if `d_ttyname_r` is defined.

u

`u16size`

From *perl5v.U*:

This variable is the size of an U16 in bytes.

`u16type`

From *perl5v.U*:

This variable contains the C type used for Perl's U16.

`u32size`

From *perl5v.U*:

This variable is the size of an U32 in bytes.

`u32type`

From *perl5v.U*:

This variable contains the C type used for Perl's U32.

`u64size`

From *perl5v.U*:

This variable is the size of an U64 in bytes.

`u64type`

From *perl5v.U*:

This variable contains the C type used for Perl's U64.

`u8size`

From *perl5v.U*:

This variable is the size of an U8 in bytes.

`u8type`

From *perlsv.U*:

This variable contains the C type used for Perl's U8.

`uidformat`

From *uidf.U*:

This variable contains the format string used for printing a Uid_t.

`uidsign`

From *uidsign.U*:

This variable contains the signedness of a uidtype. 1 for unsigned, -1 for signed.

`uidsize`

From *uidsize.U*:

This variable contains the size of a uidtype in bytes.

`uidtype`

From *uidtype.U*:

This variable defines Uid_t to be something like uid_t, int, ushort, or whatever type is used to declare user ids in the kernel.

`uname`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the uname program. After Configure runs, the value is reset to a plain `uname` and is not useful.

`uniq`

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the uniq program. After Configure runs, the value is reset to a plain `uniq` and is not useful.

`uquadtype`

From *quadtype.U*:

This variable defines Uquad_t to be something like unsigned long, unsigned int, unsigned long long, uint64_t, or whatever type is used for 64-bit integers.

`use5005threads`

From *sethreads.U*:

This variable conditionally defines the `USE_5005THREADS` symbol, and indicates that Perl should be built to use the 5.005-based threading implementation. Only valid up to 5.8.x.

`use64bitall`

From *use64bits.U*:

This variable conditionally defines the `USE_64_BIT_ALL` symbol, and indicates that 64-bit integer types should be used when available. The maximal possible 64-bitness is employed: LP64 or ILP64, meaning that you will be able to use more than 2 gigabytes of memory. This mode is even more binary incompatible than `USE_64_BIT_INT`. You may not be able to run the resulting executable in a 32-bit CPU at all or you may need at least to reboot your OS to 64-bit mode.

`use64bitint`

From *use64bits.U*:

This variable conditionally defines the `USE_64_BIT_INT` symbol, and indicates that 64-bit integer types should be used when available. The minimal possible 64-bitness is employed, just enough to get 64-bit integers into Perl. This may mean using for example "long longs", while your memory may still be limited to 2 gigabytes.

`usebacktrace`

From *usebacktrace.U*:

This variable indicates whether we are compiling with backtrace support.

`usecrosscompile`

From *Cross.U*:

This variable conditionally defines the `USE_CROSS_COMPILE` symbol, and indicates that Perl has been cross-compiled.

`usedevel`

From *Devel.U*:

This variable indicates that Perl was configured with development features enabled. This should not be done for production builds.

`usedl`

From *dlsrc.U*:

This variable indicates if the system supports dynamic loading of some sort. See also `dlsrc` and `dlobj`.

`usedtrace`

From *usedtrace.U*:

This variable indicates whether we are compiling with `dtrace` support. See also `dtrace`.

`usefaststdio`

From *usefaststdio.U*:

This variable conditionally defines the `USE_FAST_STDIO` symbol, and indicates that Perl should be built to use `fast stdio`. Defaults to define in Perls 5.8 and earlier, to undef later.

`useithreads`

From *usethreads.U*:

This variable conditionally defines the `USE_ITHREADS` symbol, and indicates that Perl should be built to use the interpreter-based threading implementation.

`usekernprocpathname`

From *usekernprocpathname.U*:

This variable, indicates that we can use `sysctl` with `KERN_PROC_PATHNAME` to get a full path for the executable, and hence convert `$^X` to an absolute path.

`uselargefiles`

From *uselfs.U*:

This variable conditionally defines the `USE_LARGE_FILES` symbol, and indicates that large file interfaces should be used when available.

`uselongdouble`

From *uselongdbl.U*:

This variable conditionally defines the `USE_LONG_DOUBLE` symbol, and indicates that long

doubles should be used when available.

`usemallocwrap`

From *mallocsrc.U*:

This variable contains `y` if we are wrapping `malloc` to prevent integer overflow during size calculations.

`usemorebits`

From *usemorebits.U*:

This variable conditionally defines the `USE_MORE_BITS` symbol, and indicates that explicit 64-bit interfaces and long doubles should be used when available.

`usemultiplicity`

From *usemultiplicity.U*:

This variable conditionally defines the `MULTIPLICITY` symbol, and indicates that Perl should be built to use multiplicity.

`usemymalloc`

From *mallocsrc.U*:

This variable contains `y` if the `malloc` that comes with this package is desired over the system's version of `malloc`. People often include special versions of `malloc` for efficiency, but such versions are often less portable. See also `mallocsrc` and `mallocojb`. If this is `y`, then `-lmalloc` is removed from `$libs`.

`usenm`

From *usenm.U*:

This variable contains `true` or `false` depending whether the `nm` extraction is wanted or not.

`usensgetexecutablepath`

From *usensgetexecutablepath.U*:

This symbol, if defined, indicates that we can use `_NSGetExecutablePath` and `realpath` to get a full path for the executable, and hence convert `$^X` to an absolute path.

`useopcode`

From *Extensions.U*:

This variable holds either `true` or `false` to indicate whether the Opcode extension should be used. The sole use for this currently is to allow an easy mechanism for users to skip the Opcode extension from the Configure command line.

`useperlio`

From *useperlio.U*:

This variable conditionally defines the `USE_PERLIO` symbol, and indicates that the PerlIO abstraction should be used throughout.

`useposix`

From *Extensions.U*:

This variable holds either `true` or `false` to indicate whether the `POSIX` extension should be used. The sole use for this currently is to allow an easy mechanism for hints files to indicate that `POSIX` will not compile on a particular system.

`usequadmath`

From *usequadmath.U*:

This variable conditionally defines the `USE_QUADMATH` symbol, and indicates that the

quadmath library `__float128` long doubles should be used when available.

`usereentrant`

From *usethreads.U*:

This variable conditionally defines the `USE_REENTRANT_API` symbol, which indicates that the thread code may try to use the various `_r` versions of library functions. This is only potentially meaningful if `usethreads` is set and is very experimental, it is not even prompted for.

`userelocatableinc`

From *bin.U*:

This variable is set to `true` to indicate that perl should relocate `@INC` entries at runtime based on the path to the perl binary. Any `@INC` paths starting `.../` are relocated relative to the directory containing the perl binary, and a logical cleanup of the path is then made around the join point (removing *dir/./* pairs)

`useshrplib`

From *libperl.U*:

This variable is set to `true` if the user wishes to build a shared `libperl`, and `false` otherwise.

`usesitecustomize`

From *d_sitecustomize.U*:

This variable is set to `true` when the user requires a mechanism that allows the sysadmin to add entries to `@INC` at runtime. This variable being set, makes perl run *\$sitelib/sitecustomize.pl* at startup.

`usesocks`

From *usesocks.U*:

This variable conditionally defines the `USE_SOCKS` symbol, and indicates that Perl should be built to use `SOCKS`.

`usethreads`

From *usethreads.U*:

This variable conditionally defines the `USE_THREADS` symbol, and indicates that Perl should be built to use threads.

`usevendorprefix`

From *vendorprefix.U*:

This variable tells whether the `vendorprefix` and consequently other `vendor*` paths are in use.

`useversionedarchname`

From *archname.U*:

This variable indicates whether to include the `$api_versionstring` as a component of the `$archname`.

`usevfork`

From *d_vfork.U*:

This variable is set to `true` when the user accepts to use `vfork`. It is set to `false` when no `vfork` is available or when the user explicitly requests not to use `vfork`.

`usrinc`

From *usrinc.U*:

This variable holds the path of the include files, which is usually */usr/include*. It is mainly used by other Configure units.

uuname

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

uvoformat

From *perlxf.U*:

This variable contains the format string used for printing a Perl UV as an unsigned octal integer.

uvsize

From *perlxv.U*:

This variable is the size of a UV in bytes.

uvtype

From *perlxv.U*:

This variable contains the C type used for Perl's UV.

uvuformat

From *perlxf.U*:

This variable contains the format string used for printing a Perl UV as an unsigned decimal integer.

uvxformat

From *perlxf.U*:

This variable contains the format string used for printing a Perl UV as an unsigned hexadecimal integer in lowercase abcdef.

uvXUformat

From *perlxf.U*:

This variable contains the format string used for printing a Perl UV as an unsigned hexadecimal integer in uppercase ABCDEF.

v**vaproto**

From *vaproto.U*:

This variable conditionally defines CAN_VAPROTO on systems supporting prototype declaration of functions with a variable number of arguments. See also prototype.

vendorarch

From *vendorarch.U*:

This variable contains the value of the PERL_VENDORARCH symbol. It may have a ~ on the front. The standard distribution will put nothing in this directory. Vendors who distribute perl may wish to place their own architecture-dependent modules and extensions in this directory with MakeMaker *Makefile.PL* `INSTALLDIRS=vendor` or equivalent. See `INSTALL` for details.

vendorarchexp

From *vendorarch.U*:

This variable is the ~*name* expanded version of vendorarch, so that you may use it directly in Makefiles or shell scripts.

vendorbin

From *vendorbin.U*:

This variable contains the eventual value of the `VENDORBIN` symbol. It may have a `~` on the front. The standard distribution will put nothing in this directory. Vendors who distribute perl may wish to place additional binaries in this directory with MakeMaker *Makefile.PL* `INSTALLDIRS=vendor` or equivalent. See `INSTALL` for details.

`vendorbinexp`

From *vendorbin.U*:

This variable is the *~name* expanded version of `vendorbin`, so that you may use it directly in Makefiles or shell scripts.

`vendorhtml1dir`

From *vendorhtml1dir.U*:

This variable contains the name of the directory for html pages. It may have a `~` on the front. The standard distribution will put nothing in this directory. Vendors who distribute perl may wish to place their own html pages in this directory with MakeMaker *Makefile.PL* `INSTALLDIRS=vendor` or equivalent. See `INSTALL` for details.

`vendorhtml1direxp`

From *vendorhtml1dir.U*:

This variable is the *~name* expanded version of `vendorhtml1dir`, so that you may use it directly in Makefiles or shell scripts.

`vendorhtml3dir`

From *vendorhtml3dir.U*:

This variable contains the name of the directory for html library pages. It may have a `~` on the front. The standard distribution will put nothing in this directory. Vendors who distribute perl may wish to place their own html pages for modules and extensions in this directory with MakeMaker *Makefile.PL* `INSTALLDIRS=vendor` or equivalent. See `INSTALL` for details.

`vendorhtml3direxp`

From *vendorhtml3dir.U*:

This variable is the *~name* expanded version of `vendorhtml3dir`, so that you may use it directly in Makefiles or shell scripts.

`vendorlib`

From *vendorlib.U*:

This variable contains the eventual value of the `VENDORLIB` symbol, which is the name of the private library for this package. The standard distribution will put nothing in this directory. Vendors who distribute perl may wish to place their own modules in this directory with MakeMaker *Makefile.PL* `INSTALLDIRS=vendor` or equivalent. See `INSTALL` for details.

`vendorlib_stem`

From *vendorlib.U*:

This variable is `$vendorlibexp` with any trailing version-specific component removed. The elements in `inc_version_list` (*inc_version_list.U*) can be tacked onto this variable to generate a list of directories to search.

`vendorlibexp`

From *vendorlib.U*:

This variable is the *~name* expanded version of `vendorlib`, so that you may use it directly in Makefiles or shell scripts.

`vendorman1dir`

From *vendorman1dir.U*:

This variable contains the name of the directory for man1 pages. It may have a ~ on the front. The standard distribution will put nothing in this directory. Vendors who distribute perl may wish to place their own man1 pages in this directory with MakeMaker *Makefile.PL* `INSTALLDIRS=vendor` or equivalent. See `INSTALL` for details.

`vendorman1direxp`

From *vendorman1dir.U*:

This variable is the *~name* expanded version of `vendorman1dir`, so that you may use it directly in Makefiles or shell scripts.

`vendorman3dir`

From *vendorman3dir.U*:

This variable contains the name of the directory for man3 pages. It may have a ~ on the front. The standard distribution will put nothing in this directory. Vendors who distribute perl may wish to place their own man3 pages in this directory with MakeMaker *Makefile.PL* `INSTALLDIRS=vendor` or equivalent. See `INSTALL` for details.

`vendorman3direxp`

From *vendorman3dir.U*:

This variable is the *~name* expanded version of `vendorman3dir`, so that you may use it directly in Makefiles or shell scripts.

`vendorprefix`

From *vendorprefix.U*:

This variable holds the full absolute path of the directory below which the vendor will install add-on packages. See `INSTALL` for usage and examples.

`vendorprefixexp`

From *vendorprefix.U*:

This variable holds the full absolute path of the directory below which the vendor will install add-on packages. Derived from `vendorprefix`.

`vendorscript`

From *vendorscript.U*:

This variable contains the eventual value of the `VENDORSRIPT` symbol. It may have a ~ on the front. The standard distribution will put nothing in this directory. Vendors who distribute perl may wish to place additional executable scripts in this directory with MakeMaker *Makefile.PL* `INSTALLDIRS=vendor` or equivalent. See `INSTALL` for details.

`vendorscriptexp`

From *vendorscript.U*:

This variable is the *~name* expanded version of `vendorscript`, so that you may use it directly in Makefiles or shell scripts.

`version`

From *patchlevel.U*:

The full version number of this package, such as 5.6.1 (or 5_6_1). This combines revision, patchlevel, and subversion to get the full version number, including any possible subversions. This is suitable for use as a directory name, and hence is filesystem dependent.

`version_patchlevel_string`

From *patchlevel.U*:

This is a string combining version, subversion and perl_patchlevel (if perl_patchlevel is non-zero). It is typically something like 'version 7 subversion 1' or 'version 7 subversion 1 patchlevel 11224'. It is computed here to avoid duplication of code in *myconfig.SH* and *lib/Config.pm*.

versiononly

From *versiononly.U*:

If set, this symbol indicates that only the version-specific components of a perl installation should be installed. This may be useful for making a test installation of a new version without disturbing the existing installation. Setting versiononly is equivalent to setting installperl's -v option. In particular, the non-versioned scripts and programs such as a2p, c2ph, h2xs, pod2*, and perldoc are not installed (see *INSTALL* for a more complete list). Nor are the man pages installed. Usually, this is undef.

vi

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

x

xlibpth

From *libpth.U*:

This variable holds extra path (space-separated) used to find libraries on this platform, for example CPU-specific libraries (on multi-CPU platforms) may be listed here.

y

yacc

From *yacc.U*:

This variable holds the name of the compiler compiler we want to use in the Makefile. It can be yacc, byacc, or bison -y.

yaccflags

From *yacc.U*:

This variable contains any additional yacc flags desired by the user. It is up to the Makefile to use this.

z

zcat

From *Loc.U*:

This variable is defined but not used by Configure. The value is the empty string and is not useful.

zip

From *Loc.U*:

This variable is used internally by Configure to determine the full pathname (if any) of the zip program. After Configure runs, the value is reset to a plain *zip* and is not useful.

GIT DATA

Information on the git commit from which the current perl binary was compiled can be found in the variable `$Config::Git_Data`. The variable is a structured string that looks something like this:

```
git_commit_id='ea0c2dbd5f5ac6845ecc7ec6696415bf8e27bd52'
```

```
git_describe='GitLive-blead-1076-gea0c2db'  
git_branch='smartmatch'  
git_uncommitted_changes=''  
git_commit_id_title='Commit id:'  
git_commit_date='2009-05-09 17:47:31 +0200'
```

Its format is not guaranteed not to change over time.

NOTE

This module contains a good example of how to use tie to implement a cache and an example of how to make a tied variable readonly to those outside of it.