NAME

ExtUtils::Constant - generate XS code to import C header constants

SYNOPSIS

```
    use ExtUtils::Constant qw (WriteConstants);
    WriteConstants(
        NAME => 'Foo',
        NAMES => [qw(FOO BAR BAZ)],
    );
    # Generates wrapper code to make the values of the constants FOO BAR
BAZ
    #  available to perl
```

DESCRIPTION

ExtUtils::Constant facilitates generating C and XS wrapper code to allow perl modules to AUTOLOAD constants defined in C library header files. It is principally used by the `h2xs` utility, on which this code is based. It doesn't contain the routines to scan header files to extract these constants.

USAGE

Generally one only needs to call the `WriteConstants` function, and then

```
    #include "const-c.inc"
```

in the C section of `Foo.xs`

```
    INCLUDE: const-xs.inc
```

in the XS section of `Foo.xs`.

For greater flexibility use `constant_types()`, `C_constant` and `XS_constant`, with which `WriteConstants` is implemented.

Currently this module understands the following types. h2xs may only know a subset. The sizes of the numeric types are chosen by the `Configure` script at compile time.

IV

signed integer, at least 32 bits.

UV

unsigned integer, the same size as *IV*

NV

floating point type, probably `double`, possibly `long double`

PV

NUL terminated string, length will be determined with `strlen`

PVN

A fixed length thing, given as a [pointer, length] pair. If you know the length of a string at compile time you may use this instead of *PV*

SV

A **mortal** SV.

YES

Truth. (PL_sv_yes) The value is not needed (and ignored).

NO

Defined Falsehood. (PL_sv_no) The value is not needed (and ignored).

UNDEF

undef. The value of the macro is not needed.

FUNCTIONS

C_stringify NAME

A function which returns a 7 bit ASCII correctly \ escaped version of the string passed suitable for C's "" or ''. It will die if passed Unicode characters.

perl_stringify NAME

A function which returns a 7 bit ASCII correctly \ escaped version of the string passed suitable for a perl "" string.

constant_types

A function returning a single scalar with #define definitions for the constants used internally between the generated C and XS functions.

memEQ_clause NAME, CHECKED_AT, INDENT

A function to return a suitable C if statement to check whether *NAME* is equal to the C variable name. If *CHECKED_AT* is defined, then it is used to avoid memEQ for short names, or to generate a comment to highlight the position of the character in the switch statement.

If *CHECKED_AT* is a reference to a scalar, then instead it gives the characters pre-checked at the beginning, (and the number of chars by which the C variable name has been advanced. These need to be chopped from the front of *NAME*).

assign INDENT, TYPE, PRE, POST, VALUE...

A function to return a suitable assignment clause. If *TYPE* is aggregate (eg *PVN* expects both pointer and length) then there should be multiple *VALUE*s for the components. *PRE* and *POST* if defined give snippets of C code to proceed and follow the assignment. *PRE* will be at the start of a block, so variables may be defined in it.

return_clause

return_clause ITEM, INDENT

A function to return a suitable #ifdef clause. *ITEM* is a hashref (as passed to C_constant and match_clause. *INDENT* is the number of spaces to indent, defaulting to 6.

XXX document me

switch_clause INDENT, NAMELEN, ITEMHASH, ITEM...

An internal function to generate a suitable switch clause, called by C_constant *ITEM*s are in the hash ref format as given in the description of C_constant, and must all have the names of the same length, given by *NAMELEN* (This is not checked). *ITEMHASH* is a reference to a hash, keyed by name, values being the hashrefs in the *ITEM* list. (No parameters are modified, and there can be keys in the *ITEMHASH* that are not in the list of *ITEM*s without causing problems).

params WHAT

An internal function. *WHAT* should be a hashref of types the constant function will return. *params* returns a hashref keyed IV NV PV SV to show which combination of pointers will be needed in the C argument list.

dump_names

dump_names DEFAULT_TYPE, TYPES, INDENT, OPTIONS, ITEM...

An internal function to generate the embedded perl code that will regenerate the constant subroutines. *DEFAULT_TYPE*, *TYPES* and *ITEM*s are the same as for C_constant. *INDENT* is treated as number of spaces to indent by. *OPTIONS* is a hashref of options. Currently only `declare_types` is recognised. If the value is true a `$types` is always declared in the perl code generated, if defined and false never declared, and if undefined `$types` is only declared if the values in *TYPES* as passed in cannot be inferred from *DEFAULT_TYPES* and the *ITEM*s.

dogfood

dogfood PACKAGE, SUBNAME, DEFAULT_TYPE, TYPES, INDENT, BREAKOUT, ITEM...

An internal function to generate the embedded perl code that will regenerate the constant subroutines. Parameters are the same as for C_constant.

C_constant

C_constant PACKAGE, SUBNAME, DEFAULT_TYPE, TYPES, INDENT, BREAKOUT, ITEM...

A function that returns a **list** of C subroutine definitions that return the value and type of constants when passed the name by the XS wrapper. *ITEM...* gives a list of constant names. Each can either be a string, which is taken as a C macro name, or a reference to a hash with the following keys

name

The name of the constant, as seen by the perl code.

type

The type of the constant (*IV*, *NV* etc)

value

A C expression for the value of the constant, or a list of C expressions if the type is aggregate. This defaults to the *name* if not given.

macro

The C pre-processor macro to use in the `#ifdef`. This defaults to the *name*, and is mainly used if *value* is an `enum`. If a reference an array is passed then the first element is used in place of the `#ifdef` line, and the second element in place of the `#endif`. This allows pre-processor constructions such as

```
#if defined (foo)
#if !defined (bar)
...
#endif
#endif
```

to be used to determine if a constant is to be defined.

A "macro" 1 signals that the constant is always defined, so the `#if`/`#endif` test is omitted.

default

Default value to use (instead of `croak`ing with "your vendor has not defined...") to return if the macro isn't defined. Specify a reference to an array with type followed by value(s).

pre

C code to use before the assignment of the value of the constant. This allows you to use temporary variables to extract a value from part of a `struct` and

return this as *value*. This C code is places at the start of a block, so you can declare variables in it.

post

C code to place between the assignment of value (to a temporary) and the return from the function. This allows you to clear up anything in *pre*. Rarely needed.

def_pre =item def_post

Equivalents of *pre* and *post* for the default value.

utf8

Generated internally. Is zero or undefined if name is 7 bit ASCII, "no" if the name is 8 bit (and so should only match if SvUTF8() is false), "yes" if the name is utf8 encoded.

The internals automatically clone any name with characters 128-255 but none 256+ (ie one that could be either in bytes or utf8) into a second entry which is utf8 encoded.

PACKAGE is the name of the package, and is only used in comments inside the generated C code.

The next 5 arguments can safely be given as `undef`, and are mainly used for recursion. *SUBNAME* defaults to `constant` if undefined.

DEFAULT_TYPE is the type returned by `ITEM`s that don't specify their type. In turn it defaults to *IV*. *TYPES* should be given either as a comma separated list of types that the C subroutine `constant` will generate or as a reference to a hash. *DEFAULT_TYPE* will be added to the list if not present, as will any types given in the list of *ITEM*s. The resultant list should be the same list of types that `XS_constant` is given. [Otherwise `XS_constant` and `C_constant` may differ in the number of parameters to the constant function. *INDENT* is currently unused and ignored. In future it may be used to pass in information used to change the C indentation style used.] The best way to maintain consistency is to pass in a hash reference and let this function update it.

BREAKOUT governs when child functions of *SUBNAME* are generated. If there are *BREAKOUT* or more *ITEM*s with the same length of name, then the code to switch between them is placed into a function named *SUBNAME_LEN*, for example `constant_5` for names 5 characters long. The default *BREAKOUT* is 3. A single `ITEM` is always inlined.

XS_constant PACKAGE, TYPES, SUBNAME, C_SUBNAME

A function to generate the XS code to implement the perl subroutine *PACKAGE*::constant used by *PACKAGE*::AUTOLOAD to load constants. This XS code is a wrapper around a C subroutine usually generated by `C_constant`, and usually named `constant`.

TYPES should be given either as a comma separated list of types that the C subroutine `constant` will generate or as a reference to a hash. It should be the same list of types as `C_constant` was given. [Otherwise `XS_constant` and `C_constant` may have different ideas about the number of parameters passed to the C function `constant`]

You can call the perl visible subroutine something other than `constant` if you give the parameter *SUBNAME*. The C subroutine it calls defaults to the name of the perl visible subroutine, unless you give the parameter *C_SUBNAME*.

autoload PACKAGE, VERSION, AUTOLOADER

A function to generate the AUTOLOAD subroutine for the module *PACKAGE VERSION* is the perl version the code should be backwards compatible with. It defaults to the version of perl running the subroutine. If *AUTOLOADER* is true, the AUTOLOAD subroutine falls back on AutoLoader::AUTOLOAD for all names that the constant() routine doesn't recognise.

WriteMakefileSnippet

WriteMakefileSnippet ATTRIBUTE => VALUE [, ...]

A function to generate perl code for Makefile.PL that will regenerate the constant subroutines. Parameters are named as passed to `WriteConstants`, with the addition of `INDENT` to specify the number of leading spaces (default 2).

Currently only `INDENT`, `NAME`, `DEFAULT_TYPE`, `NAMES`, `C_FILE` and `XS_FILE` are recognised.

WriteConstants ATTRIBUTE => VALUE [, ...]

Writes a file of C code and a file of XS code which you should `#include` and `INCLUDE` in the C and XS sections respectively of your module's XS code. You probably want to do this in your `Makefile.PL`, so that you can easily edit the list of constants without touching the rest of your module. The attributes supported are

NAME

Name of the module. This must be specified

DEFAULT_TYPE

The default type for the constants. If not specified `IV` is assumed.

BREAKOUT_AT

The names of the constants are grouped by length. Generate child subroutines for each group with this number or more names in.

NAMES

An array of constants' names, either scalars containing names, or hashrefs as detailed in *C_constant*.

C_FILE

The name of the file to write containing the C code. The default is `const-c.inc`. The `-` in the name ensures that the file can't be mistaken for anything related to a legitimate perl package name, and not naming the file `.c` avoids having to override Makefile.PL's `.xs` to `.c` rules.

XS_FILE

The name of the file to write containing the XS code. The default is `const-xs.inc`.

SUBNAME

The perl visible name of the XS subroutine generated which will return the constants. The default is `constant`.

C_SUBNAME

The name of the C subroutine generated which will return the constants. The default is *SUBNAME*. Child subroutines have _ and the name length appended, so constants with 10 character names would be in `constant_10` with the default *XS_SUBNAME*.

AUTHOR

Nicholas Clark <nick@ccl4.org> based on the code in `h2xs` by Larry Wall and others