NAME

h2xs - convert .h C header files to Perl extensions

SYNOPSIS

h2xs [**OPTIONS** ...] [headerfile ... [extra_libraries]]

h2xs -h|-?|--help

DESCRIPTION

h2xs builds a Perl extension from C header files. The extension will include functions which can be used to retrieve the value of any #define statement which was in the C header files.

The *module_name* will be used for the name of the extension. If module_name is not supplied then the name of the first header file will be used, with the first character capitalized.

If the extension might need extra libraries, they should be included here. The extension Makefile.PL will take care of checking whether the libraries actually exist and how they should be loaded. The extra libraries should be specified in the form -lm -lposix, etc, just as on the cc command line. By default, the Makefile.PL will search through the library path determined by Configure. That path can be augmented by including arguments of the form **-L/another/library/path** in the extra-libraries argument.

In spite of its name, *h2xs* may also be used to create a skeleton pure Perl module. See the **-X** option.

OPTIONS

-A, **--omit-autoload**

Omit all autoload facilities. This is the same as **-c** but also removes the `use AutoLoader` statement from the .pm file.

-B, **--beta-version**

Use an alpha/beta style version number. Causes version number to be "0.00_01" unless **-v** is specified.

-C, **--omit-changes**

Omits creation of the *Changes* file, and adds a HISTORY section to the POD template.

-F, **--cpp-flags**=*addflags*

Additional flags to specify to C preprocessor when scanning header for function declarations. Writes these options in the generated *Makefile.PL* too.

-M, **--func-mask**=*regular expression*

selects functions/macros to process.

-O, **--overwrite-ok**

Allows a pre-existing extension directory to be overwritten.

-P, **--omit-pod**

Omit the autogenerated stub POD section.

-X, **--omit-XS**

Omit the XS portion. Used to generate a skeleton pure Perl module. `-c` and `-f` are implicitly enabled.

-a, **--gen-accessors**

Generate an accessor method for each element of structs and unions. The generated methods are named after the element name; will return the current value of the element if called without additional arguments; and will set the element to the supplied value (and

return the new value) if called with an additional argument. Embedded structures and unions are returned as a pointer rather than the complete structure, to facilitate chained calls.

These methods all apply to the Ptr type for the structure; additionally two methods are constructed for the structure type itself, `_to_ptr` which returns a Ptr type pointing to the same structure, and a `new` method to construct and return a new structure, initialised to zeroes.

-b, **--compat-version**=*version*

Generates a .pm file which is backwards compatible with the specified perl version.

For versions < 5.6.0, the changes are. - no use of 'our' (uses 'use vars' instead) - no 'use warnings'

Specifying a compatibility version higher than the version of perl you are using to run h2xs will have no effect. If unspecified h2xs will default to compatibility with the version of perl you are using to run h2xs.

-c, **--omit-constant**

Omit `constant()` from the .xs file and corresponding specialised `AUTOLOAD` from the .pm file.

-d, **--debugging**

Turn on debugging messages.

-e, **--omit-enums**=[*regular expression*]

If *regular expression* is not given, skip all constants that are defined in a C enumeration. Otherwise skip only those constants that are defined in an enum whose name matches *regular expression*.

Since *regular expression* is optional, make sure that this switch is followed by at least one other switch if you omit *regular expression* and have some pending arguments such as header-file names. This is ok:

```
h2xs -e -n Module::Foo foo.h
```

This is not ok:

```
h2xs -n Module::Foo -e foo.h
```

In the latter, foo.h is taken as *regular expression*.

-f, **--force**

Allows an extension to be created for a header even if that header is not found in standard include directories.

-g, **--global**

Include code for safely storing static data in the .xs file. Extensions that do no make use of static data can ignore this option.

-h, **-?**, **--help**

Print the usage, help and version for this h2xs and exit.

-k, **--omit-const-func**

For function arguments declared as `const`, omit the const attribute in the generated XS code.

-m, **--gen-tied-var**

Experimental: for each variable declared in the header file(s), declare a perl variable of the same name magically tied to the C variable.

-n, **--name**=*module_name*

Specifies a name to be used for the extension, e.g., -n RPC::DCE

-o, **--opaque-re**=*regular expression*

Use "opaque" data type for the C types matched by the regular expression, even if these types are `typedef`-equivalent to types from typemaps. Should not be used without **-x**.

This may be useful since, say, types which are `typedef`-equivalent to integers may represent OS-related handles, and one may want to work with these handles in OO-way, as in `$handle->do_something()`. Use `-o .` if you want to handle all the `typedef`ed types as opaque types.

The type-to-match is whitewashed (except for commas, which have no whitespace before them, and multiple `*` which have no whitespace between them).

-p, **--remove-prefix**=*prefix*

Specify a prefix which should be removed from the Perl function names, e.g., -p sec_rgy_ This sets up the XS **PREFIX** keyword and removes the prefix from functions that are autoloaded via the `constant()` mechanism.

-s, **--const-subs**=*sub1,sub2*

Create a perl subroutine for the specified macros rather than autoload with the constant() subroutine. These macros are assumed to have a return type of **char ***, e.g., -s sec_rgy_wildcard_name,sec_rgy_wildcard_sid.

-t, **--default-type**=*type*

Specify the internal type that the constant() mechanism uses for macros. The default is IV (signed integer). Currently all macros found during the header scanning process will be assumed to have this type. Future versions of `h2xs` may gain the ability to make educated guesses.

--use-new-tests

When **--compat-version** (**-b**) is present the generated tests will use `Test::More` rather than `Test` which is the default for versions before 5.6.2. `Test::More` will be added to PREREQ_PM in the generated `Makefile.PL`.

--use-old-tests

Will force the generation of test code that uses the older `Test` module.

--skip-exporter

Do not use `Exporter` and/or export any symbol.

--skip-ppport

Do not use `Devel::PPPort`: no portability to older version.

--skip-autoloader

Do not use the module `AutoLoader`; but keep the constant() function and `sub AUTOLOAD` for constants.

--skip-strict

Do not use the pragma `strict`.

--skip-warnings

Do not use the pragma `warnings`.

-v, **--version**=*version*

Specify a version number for this extension. This version number is added to the templates.

The default is 0.01, or 0.00_01 if `-B` is specified. The version specified should be numeric.

-x, **--autogen-xsubs**

Automatically generate XSUBs basing on function declarations in the header file. The package `C::Scan` should be installed. If this option is specified, the name of the header file may look like `NAME1,NAME2`. In this case NAME1 is used instead of the specified string, but XSUBs are emitted only for the declarations included from file NAME2.

Note that some types of arguments/return-values for functions may result in XSUB-declarations/typemap-entries which need hand-editing. Such may be objects which cannot be converted from/to a pointer (like `long long`), pointers to functions, or arrays. See also the section on *LIMITATIONS of* ***-x***.

EXAMPLES

```
# Default behavior, extension is Rusers
h2xs rpcsvc/rusers

# Same, but extension is RUSERS
h2xs -n RUSERS rpcsvc/rusers

# Extension is rpcsvc::rusers. Still finds <rpcsvc/rusers.h>
h2xs rpcsvc::rusers

# Extension is ONC::RPC.  Still finds <rpcsvc/rusers.h>
h2xs -n ONC::RPC rpcsvc/rusers

# Without constant() or AUTOLOAD
h2xs -c rpcsvc/rusers

# Creates templates for an extension named RPC
h2xs -cfn RPC

# Extension is ONC::RPC.
h2xs -cfn ONC::RPC

# Extension is a pure Perl module with no XS code.
h2xs -X My::Module

# Extension is Lib::Foo which works at least with Perl5.005_03.
# Constants are created for all #defines and enums h2xs can find
# in foo.h.
h2xs -b 5.5.3 -n Lib::Foo foo.h

# Extension is Lib::Foo which works at least with Perl5.005_03.
# Constants are created for all #defines but only for enums
# whose names do not start with 'bar_'.
h2xs -b 5.5.3 -e '^bar_' -n Lib::Foo foo.h

# Makefile.PL will look for library -lrpc in
# additional directory /opt/net/lib
h2xs rpcsvc/rusers -L/opt/net/lib -lrpc

# Extension is DCE::rgynbase
```

```
# prefix "sec_rgy_" is dropped from perl function names
h2xs -n DCE::rgynbase -p sec_rgy_ dce/rgynbase

# Extension is DCE::rgynbase
# prefix "sec_rgy_" is dropped from perl function names
# subroutines are created for sec_rgy_wildcard_name and
# sec_rgy_wildcard_sid
h2xs -n DCE::rgynbase -p sec_rgy_ \
-s sec_rgy_wildcard_name,sec_rgy_wildcard_sid dce/rgynbase

# Make XS without defines in perl.h, but with function declarations
# visible from perl.h. Name of the extension is perl1.
# When scanning perl.h, define -DEXT=extern -DdEXT= -DINIT(x)=
# Extra backslashes below because the string is passed to shell.
# Note that a directory with perl header files would
#  be added automatically to include path.
h2xs -xAn perl1 -F "-DEXT=extern -DdEXT= -DINIT\(x\)=" perl.h

# Same with function declaration in proto.h as visible from perl.h.
h2xs -xAn perl2 perl.h,proto.h

# Same but select only functions which match /^av_/
h2xs -M '^av_' -xAn perl2 perl.h,proto.h

# Same but treat SV* etc as "opaque" types
h2xs -o '^[S]V \*$' -M '^av_' -xAn perl2 perl.h,proto.h
```

Extension based on .h and .c files

Suppose that you have some C files implementing some functionality, and the corresponding header files. How to create an extension which makes this functionality accessible in Perl? The example below assumes that the header files are *interface_simple.h* and *interface_hairy.h*, and you want the perl module be named as `Ext::Ension`. If you need some preprocessor directives and/or linking with external libraries, see the flags `-F`, `-L` and `-l` in *OPTIONS*.

Find the directory name

Start with a dummy run of h2xs:

```
h2xs -Afn Ext::Ension
```

The only purpose of this step is to create the needed directories, and let you know the names of these directories. From the output you can see that the directory for the extension is *Ext/Ension*.

Copy C files

Copy your header files and C files to this directory *Ext/Ension*.

Create the extension

Run h2xs, overwriting older autogenerated files:

```
h2xs -Oxan Ext::Ension interface_simple.h interface_hairy.h
```

h2xs looks for header files *after* changing to the extension directory, so it will find your header files OK.

Archive and test

As usual, run

```
cd Ext/Ension
perl Makefile.PL
make dist
make
make test
```

Hints

It is important to do `make dist` as early as possible. This way you can easily merge(1) your changes to autogenerated files if you decide to edit your `.h` files and rerun h2xs.

Do not forget to edit the documentation in the generated *.pm* file.

Consider the autogenerated files as skeletons only, you may invent better interfaces than what h2xs could guess.

Consider this section as a guideline only, some other options of h2xs may better suit your needs.

ENVIRONMENT

No environment variables are used.

AUTHOR

Larry Wall and others

SEE ALSO

perl, *perlxstut*, *ExtUtils::MakeMaker*, and *AutoLoader*.

DIAGNOSTICS

The usual warnings if it cannot read or write the files involved.

LIMITATIONS of -x

h2xs would not distinguish whether an argument to a C function which is of the form, say, `int *`, is an input, output, or input/output parameter. In particular, argument declarations of the form

```
    int
    foo(n)
 int *n
```

should be better rewritten as

```
    int
    foo(n)
 int &n
```

if n is an input parameter.

Additionally, *h2xs* has no facilities to intuit that a function

```
   int
   foo(addr,l)
 char *addr
 int   l
```

takes a pair of address and length of data at this address, so it is better to rewrite this function as

```
    int
    foo(sv)
```

```
    SV *addr
PREINIT:
    STRLEN len;
    char *s;
CODE:
    s = SvPV(sv,len);
    RETVAL = foo(s, len);
OUTPUT:
    RETVAL
```

or alternately

```
   static int
   my_foo(SV *sv)
   {
STRLEN len;
char *s = SvPV(sv,len);

return foo(s, len);
   }

   MODULE = foo PACKAGE = foo PREFIX = my_

   int
   foo(sv)
SV *sv
```

See *perlxs* and *perlxstut* for additional details.