What is perlxs

XS is an interface description file format used to create an extension interface between Perl and C code (or a C library) which one wishes to use with Perl. The XS interface is combined with the library to create a new library which can then be either dynamically loaded or statically linked into perl. The XS interface description is written in the XS language and is the core component of the Perl extension interface. After compilation by the **xsubpp** compiler, each XSUB amounts to a C function definition which will provide the glue between Perl calling conventions and C calling conventions.

The glue code pulls the arguments from the Perl stack, converts these Perl values to the formats expected by a C function, call this C function, transfers the return values of the C function back to Perl. These return values may be passed back to Perl either by putting them on the Perl stack, or by modifying the arguments supplied from the Perl side.

The above is a somewhat simplified view of what really happens. Since Perl allows more flexible calling conventions than C, XSUBs may do much more in practice, such as checking input parameters for validity, throwing exceptions (or returning undef/empty list) if the return value from the C function indicates failure, calling different C functions based on numbers and types of the arguments, providing an object-oriented interface, etc

Of course, one could write such glue code directly in C. However, this would be a tedious task, especially if one needs to write glue for multiple C functions, and/or one is not familiar enough with the Perl stack discipline and other such arcana. XS comes to the rescue here: instead of writing this glue C code in long-hand, one can write a more concise short-hand *description* of what should be done by the glue, and let the XS compiler **xsubpp** handle the rest.

How to create a perl xs module

Create a directory called XSFun and inside it create a file called lib/XSFun.pm content as follows:

```perl
package XSFun;

use strict;
use warnings;
use XSLoader;

use Exporter 5.57 'import';

our $VERSION     = '0.001';
our %EXPORT_TAGS = ( 'all' => [] );
our @EXPORT_OK   = ( @{ $EXPORT_TAGS{'all'} } );

XSLoader::load('XSFun', $VERSION);

1;
```

XSLoader loads C libraries dynamically so they can be used in Perl. As soon as you use XSFun, it will load our XS code using XSLoader and make it available and known to Perl.

In the top-level directory create the Makefile.PL which will generate a proper Makefile, content as follows:

```perl
use 5.008005;
use ExtUtils::MakeMaker;
WriteMakefile(
    NAME           => 'XSFun',
    VERSION_FROM   => 'lib/XSFun.pm',
    PREREQ_PM      => { 'Test::More' => 0, 'Exporter' => '5.57' },
    ABSTRACT_FROM  => 'lib/XSFun.pm',
    AUTHOR         => 'You',
    LIBS           => [''],
    DEFINE         => '',
    INC            => '-I.',
    OBJECT         => '$(O_FILES)',
);
```

Create the file XSFun.xs in the top-level directory with the following content:

```
#define PERL_NO_GET_CONTEXT
#include "EXTERN.h"
#include "perl.h"
#include "XSUB.h"
#include "ppport.h"

/* C functions */

MODULE = XSFun      PACKAGE = XSFun

double
add_numbers(double a, double b)
    CODE:
       RETVAL = a + b;
    OUTPUT:
       RETVAL

SV *
add_numbers_perl(SV *a, SV *b)
    CODE:
    {
        const double sum = SvNV(a) + SvNV(b);
        RETVAL = newSVnv(sum);
    }
    OUTPUT: RETVAL
```

We define a function called add_numbers. It takes two numbers of type double and as you can see by the definition, it also returns a double type.

We have a CODE section which sets the return value RETVAL to the sum of a and b. We also have an OUTPUT section that indicates the output is the return value RETVAL.

Instead of working with pure C types, we can use Perl types. We can write a function that receives pointers to two SVs (Scalar Values) and returns an SV pointer to the result. We will create a new SV to represent the value and return a pointer to that.

Create the file hello.pl in the top-level directory with the following content:

```
use strict;
use warnings;
use feature 'say';
use lib qw{blib/lib blib/arch};
use XSFun qw(:all);

say add_numbers(1, 2);
say add_numbers(1.4, 3.2);
say add_numbers_perl(1, 2);
say add_numbers_perl(1.4, 3.2);
```

```
perl Makefile.PL
make
perl hello.pl
```

```
$ perl hello.pl
3
4.6
3
4.6
```

While it's good to be able to speed-up portions of our code using XS, another major use of XS is to create bindings to C and C++ libraries.

Let's start with a simple function that fetches the library version. Add the following XS code to C<XSFun.xs>:

```
#include <chromaprint.h>

const char *
get_version()
    CODE:
        RETVAL = chromaprint_get_version();
    OUTPUT: RETVAL
```

Create the file hello.pl in the top-level directory with the following content:

```
use strict;
use warnings;
use feature 'say';
use lib qw{blib/lib blib/arch};
use XSFun qw(:all);

say get_version();
```

```
perl Makefile.PL
make
perl hello.pl
$ perl hello.pl
1.0.0
```

Caveats

```
void
  alpha()
      PPCODE:
          ST(0) = newSVpv("Hello World",0);
          sv_2mortal(ST(0));
          XSRETURN(1);

 SV *
 beta()
      CODE:
          RETVAL = newSVpv("Hello World",0);
      OUTPUT:
          RETVAL

 AV *
 array()
      CODE:
          RETVAL = newAV();
          /* do something with RETVAL */
      OUTPUT:
          RETVAL

AV *
array()
      CODE:
          RETVAL = newAV();
          sv_2mortal((SV*)RETVAL);
          /* do something with RETVAL */
      OUTPUT:
          RETVAL
```

perlxs	http://perldoc.perl.org/perlxs.html
xs-fun	https://github.com/xsawyerx/xs-fun
perlapi	http://perldoc.perl.org/perlapi.html
General index	http://perldoc.perl.org/index-internals.html
perlhack	http://perldoc.perl.org/perlhack.html
perlhacktips	http://perldoc.perl.org/perlhacktips.html
perlhacktut	http://perldoc.perl.org/perlhacktut.html
perlguts	http://perldoc.perl.org/perlguts.html
perlintern	http://perldoc.perl.org/perlintern.html
perlinterp	http://perldoc.perl.org/perlinterp.html
perlcall	http://perldoc.perl.org/perlcall.html
perlsource	http://perldoc/perl.org/perlsource.html