XS Recipes

Vincent Pit

2009-08-05

Whatsit?

XS is a set of macros aimed at simplifying the writing of Perl-like functions in C.

Whatsit?

XS is a set of macros aimed at simplifying the writing of Perl-like functions in C.

Pros :

Whatsit?

XS is a set of macros aimed at simplifying the writing of Perl-like functions in C.

Pros :

- Gives full access to the power of the perl API ;

Whatsit?

XS is a set of macros aimed at simplifying the writing of Perl-like functions in C.

Pros :

- Gives full access to the power of the perl API ;
- Simplifies the handling of the Perl stack from C ;

Whatsit?

XS is a set of macros aimed at simplifying the writing of Perl-like functions in C.

Pros :

- Gives full access to the power of the perl API ;
- Simplifies the handling of the Perl stack from C ;
- Easy wrapping of C <-> Perl objects with typemaps ;

Whatsit?

XS is a set of macros aimed at simplifying the writing of Perl-like functions in C.

Pros :

- Gives full access to the power of the perl API ;
- Simplifies the handling of the Perl stack from C ;
- Easy wrapping of C <-> Perl objects with typemaps ;
- Great toolchain.

Whatsit?

XS is a set of macros aimed at simplifying the writing of Perl-like functions in C.

Pros :

- Gives full access to the power of the perl API ;
- Simplifies the handling of the Perl stack from C ;
- Easy wrapping of C <-> Perl objects with typemaps ;
- Great toolchain.

Cons :

Whatsit?

XS is a set of macros aimed at simplifying the writing of Perl-like functions in C.

Pros :

- Gives full access to the power of the perl API ;
- Simplifies the handling of the Perl stack from C ;
- Easy wrapping of C <-> Perl objects with typemaps ;
- Great toolchain.

Cons :

- The parsing of XS is brittle and the error messages are confusing ;

Whatsit?

XS is a set of macros aimed at simplifying the writing of Perl-like functions in C.

Pros :

- Gives full access to the power of the perl API ;
- Simplifies the handling of the Perl stack from C ;
- Easy wrapping of C <-> Perl objects with typemaps ;
- Great toolchain.

Cons :

- The parsing of XS is brittle and the error messages are confusing ;
- You need to learn at least the basics of the perl API (but it's not as hard as it sounds).

Alternatives

- Foreign function interfaces :

Alternatives

- Foreign function interfaces :
  - `Win32::API` ;

Alternatives

- Foreign function interfaces :
  - `Win32::API` ;
  - `P5NCI` ;

Alternatives

- Foreign function interfaces :
  - `Win32::API` ;
  - `P5NCI` ;
  - `FFI`, built on `libffi`.

Alternatives

- Foreign function interfaces :
  - `Win32::API` ;
  - `P5NCI` ;
  - `FFI`, built on `libffi`.
- XS generators :

Alternatives

- Foreign function interfaces :
  - `Win32::API` ;
  - `P5NCI` ;
  - `FFI`, built on `libffi`.
- XS generators :
  - `Inline::C` ;

Alternatives

- Foreign function interfaces :
  - `Win32::API` ;
  - `P5NCI` ;
  - `FFI`, built on `libffi`.
- XS generators :
  - `Inline::C` ;
  - `SWIG`.

Finding resources

- `perldocs` :

Finding resources

- `perldocs` :
  - Tutorials : `perlxstut`, `perlcall` ;

Finding resources

- `perldocs` :
  - Tutorials : `perlxstut`, `perlcall` ;
  - Reference texts : `perlxs`, `perlguts` ;

Finding resources

- `perldocs` :
  - Tutorials : `perlxstut`, `perlcall` ;
  - Reference texts : `perlxs`, `perlguts` ;
  - API reference : `perlapi`, `perlintern` ;

Finding resources

- `perldocs` :
  - Tutorials : `perlxstut`, `perlcall` ;
  - Reference texts : `perlxs`, `perlguts` ;
  - API reference : `perlapi`, `perlintern` ;
  - Regular expressions engine : `perlreguts`, `perlreapi`.

Finding resources

- `perldocs` :
  - Tutorials : `perlxstut`, `perlcall` ;
  - Reference texts : `perlxs`, `perlguts` ;
  - API reference : `perlapi`, `perlintern` ;
  - Regular expressions engine : `perlreguts`, `perlreapi`.
- Others :

Finding resources

- `perldocs` :
  - Tutorials : `perlxstut`, `perlcall` ;
  - Reference texts : `perlxs`, `perlguts` ;
  - API reference : `perlapi`, `perlintern` ;
  - Regular expressions engine : `perlreguts`, `perlreapi`.
- Others :
  - Source of CPAN modules ;

Finding resources

- `perldocs` :
  - Tutorials : `perlxstut`, `perlcall` ;
  - Reference texts : `perlxs`, `perlguts` ;
  - API reference : `perlapi`, `perlintern` ;
  - Regular expressions engine : `perlreguts`, `perlreapi`.
- Others :
  - Source of CPAN modules ;
  - Perl 5 source : `http://perl5.git.perl.org/` ;

Finding resources

- `perldocs` :
  - Tutorials : `perlxstut`, `perlcall` ;
  - Reference texts : `perlxs`, `perlguts` ;
  - API reference : `perlapi`, `perlintern` ;
  - Regular expressions engine : `perlreguts`, `perlreapi`.
- Others :
  - Source of CPAN modules ;
  - Perl 5 source : `http://perl5.git.perl.org/` ;
  - `perl5-porters` archives : `http://www.xray.mpe.mpg.de/mailing-lists/perl5-porters` ;

Finding resources

- `perldocs` :
  - Tutorials : `perlxstut`, `perlcall` ;
  - Reference texts : `perlxs`, `perlguts` ;
  - API reference : `perlapi`, `perlintern` ;
  - Regular expressions engine : `perlreguts`, `perlreapi`.
- Others :
  - Source of CPAN modules ;
  - Perl 5 source : `http://perl5.git.perl.org/` ;
  - `perl5-porters` archives : `http://www.xray.mpe.mpg.de/mailing-lists/perl5-porters` ;
  - `illguts` : `http://search.cpan.org/dist/illguts/` ;

Finding resources

- `perldocs` :
  - Tutorials : `perlxstut`, `perlcall` ;
  - Reference texts : `perlxs`, `perlguts` ;
  - API reference : `perlapi`, `perlintern` ;
  - Regular expressions engine : `perlreguts`, `perlreapi`.
- Others :
  - Source of CPAN modules ;
  - Perl 5 source : `http://perl5.git.perl.org/` ;
  - `perl5-porters` archives : `http://www.xray.mpe.mpg.de/mailing-lists/perl5-porters` ;
  - `illguts` : `http://search.cpan.org/dist/illguts/` ;
  - `corehackers` wiki : `http://corehackers.perl.org/` ;

Basic XS recipes

Boilerplates

XS file

```
$ cat My-XSModule/XSModule.xs
#define PERL_NO_GET_CONTEXT
#include "EXTERN.h"
#include "perl.h"
#include "XSUB.h"
```

Boilerplates

XS file

```
$ cat My-XSModule/XSModule.xs
#define PERL_NO_GET_CONTEXT
#include "EXTERN.h"
#include "perl.h"
#include "XSUB.h"

/* Put your C code that uses the perl API here */
```

Boilerplates

XS file

```
$ cat My-XSModule/XSModule.xs
#define PERL_NO_GET_CONTEXT
#include "EXTERN.h"
#include "perl.h"
#include "XSUB.h"

/* Put your C code that uses the perl API here */

MODULE = My::XSModule      PACKAGE = My::XSModule

PROTOTYPES: DISABLE

# Put your XS code here
```

.pm file

```
$ cat My-XSModule/lib/My/XSModule.pm
package My::XSModule;
use strict;
use warnings;
our $VERSION;
BEGIN {
 $VERSION = '0.01';
 require XSLoader;
 XSLoader::load(__PACKAGE__, $VERSION);
}
# You can call your XS functions from there
1;
```

Setting up the distribution

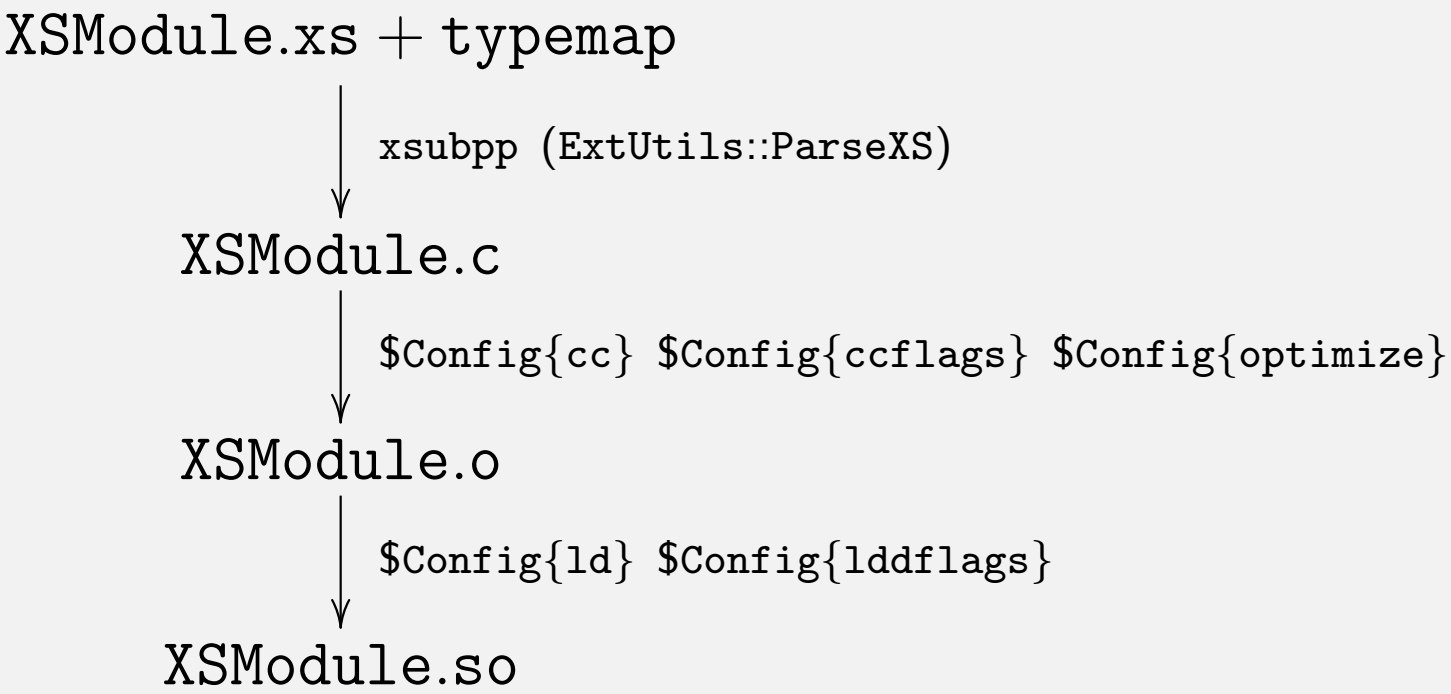
By default, `ExtUtils::MakeMaker` and `Module::Install` automatically build any XS file that's located in the root directory of the distribution.

By default, `Module::Build` automatically builds any XS file that's located under `lib`.

So you usually don't need to add anything specific to your `Makefile.PL`/`Build.PL`, except for the dependency on `XSLoader` (or `DynaLoader` if you're old school).

Setting up the distribution

How does it work?



Defining an XS function

```
SV *add(SV *x, SV *y)
CODE:
 RETVAL = newSVnv(SvNV(x) + SvNV(y));
OUTPUT:
 RETVAL
```

will become `&My::XSModule::add`.

Defining an XS function

```
SV *add(SV *x, SV *y)
CODE:
 RETVAL = newSVnv(SvNV(x) + SvNV(y));
OUTPUT:
 RETVAL
```

will become `&My::XSModule::add`.

XS subs are separated by blank lines.

Perl values

Datatype hierarchy

`SV` is the C typedef for Perl scalars.

Perl values

Datatype hierarchy

`SV` is the C typedef for Perl scalars.

A `SV` is made of :

Perl values

Datatype hierarchy

`SV` is the C typedef for Perl scalars.

A `SV` is made of :

- a common header (flags, refcount) ;

Perl values

Datatype hierarchy

`SV` is the C typedef for Perl scalars.

A `SV` is made of :

- a common header (flags, refcount) ;
- an union that can hold the main type-specific data ;

Perl values

Datatype hierarchy

`SV` is the C typedef for Perl scalars.

A `SV` is made of :

- a common header (flags, refcount) ;
- an union that can hold the main type-specific data ;
- a pointer to the extra type-specific data.

Perl values

Datatype hierarchy

`SV` is the C typedef for Perl scalars.

A `SV` is made of :

- a common header (flags, refcount) ;
- an union that can hold the main type-specific data ;
- a pointer to the extra type-specific data.

`AV`, `HV`, `CV`, `GV` (array, hash, code, glob) are just `SV`s with the extra type-specific data correctly typed.

Perl values

Datatype hierarchy

`SV` is the C typedef for Perl scalars.

A `SV` is made of :

- a common header (flags, refcount) ;
- an union that can hold the main type-specific data ;
- a pointer to the extra type-specific data.

`AV`, `HV`, `CV`, `GV` (array, hash, code, glob) are just `SV`s with the extra type-specific data correctly typed.

Since they all have the same size, you can freely cast them, and they can be allocated from an arena.

Perl values

Memory management

Perl variables are reference counted.

Perl values

Memory management

Perl variables are reference counted.

You have to increment their refcount when you store them in a separate location.

Perl values

Memory management

Perl variables are reference counted.

You have to increment their refcount when you store them in a separate location.

Mortalized values are values whose refcount will be decremented soon.

Perl values

Memory management

Perl variables are reference counted.

You have to increment their refcount when you store them in a separate location.

Mortalized values are values whose refcount will be decremented soon.

The perl stack itself is not refcounted, so you should always return mortalized values from your XS subs.

Perl values

Memory management

Perl variables are reference counted.

You have to increment their refcount when you store them in a separate location.

Mortalized values are values whose refcount will be decremented soon.

The perl stack itself is not refcounted, so you should always return mortalized values from your XS subs.

`RETVAL` is automatically mortalized.

The perl API

Mnemonics

- `[SAHCG]v[A-Z]+.*` : simple accessors/mutators macros, e.g. `SvFLAGS()`, `SvREFCNT_inc()`, `AvFILL()` ;

The perl API

Mnemonics

- `[SAHCG]v[A-Z]+.*` : simple accessors/mutators macros, e.g. `SvFLAGS()`, `SvREFCNT_inc()`, `AvFILL()` ;
- `[SAHCG]V[tfp]?_[A-Z_]+` : constants, e.g. `SVt_PVIV`, `SVf_IOK` ;

The perl API

Mnemonics

- `[SAHCG]v[A-Z]+.*` : simple accessors/mutators macros, e.g. `SvFLAGS()`, `SvREFCNT_inc()`, `AvFILL()` ;
- `[SAHCG]V[tfp]?_[A-Z_]+` : constants, e.g. `SVt_PVIV`, `SVf_IOK` ;
- `[sahcg]v_[a-z_]+.*` : macros wrappers around functions, e.g. `sv_setiv_mg()`, `av_len()`, `hv_fetch_ent()` ;

The perl API

Mnemonics

- `[SAHCG]v[A-Z]+.*` : simple accessors/mutators macros,
  e.g. `SvFLAGS()`, `SvREFCNT_inc()`, `AvFILL()` ;
- `[SAHCG]V[tfp]?_[A-Z_]+` : constants,
  e.g. `SVt_PVIV`, `SVf_IOK` ;
- `[sahcg]v_[a-z_]+.*` : macros wrappers around functions,
  e.g. `sv_setiv_mg()`, `av_len()`, `hv_fetch_ent()` ;
- `new([A-Z]+)[a-z_]*` : constructor for `$1`,
  e.g. `newSVpvn_flags()`, `newAV()`, `newHV()` ;

The perl API

Mnemonics

- `[SAHCG]v[A-Z]+.*` : simple accessors/mutators macros,
  e.g. `SvFLAGS()`, `SvREFCNT_inc()`, `AvFILL()` ;
- `[SAHCG]V[tfp]?_[A-Z_]+` : constants,
  e.g. `SVt_PVIV`, `SVf_IOK` ;
- `[sahcg]v_[a-z_]+.*` : macros wrappers around functions,
  e.g. `sv_setiv_mg()`, `av_len()`, `hv_fetch_ent()` ;
- `new([A-Z]+)[a-z_]*` : constructor for `$1`,
  e.g. `newSVpvn_flags()`, `newAV()`, `newHV()` ;
- `PL_[a-z_]+` : "global" variables,
  e.g. `PL_sv_undef`, `PL_stack_sp`.

The perl API

Mnemonics for functions

- `([iunpr])v` : `$1` gives the source type,
  e.g. `sv_setiv(sv, iv)`, `newSVrv(rv)` ;

The perl API

Mnemonics for functions

- `([iunpr])v` : `$1` gives the source type,
  e.g. `sv_setiv(sv, iv)`, `newSVrv(rv)` ;
- `pv([nsf]?)` : `n` for length, `s` for static, `f` for format,
  e.g. `newSVpvn(buf, len)`, `sv_catpvf(sv, fmt, ...)`.

Suffixes :

The perl API

Mnemonics for functions

- `([iunpr])v` : `$1` gives the source type,
  e.g. `sv_setiv(sv, iv)`, `newSVrv(rv)` ;
- `pv([nsf]?)` : `n` for length, `s` for static, `f` for format,
  e.g. `newSVpvn(buf, len)`, `sv_catpvf(sv, fmt, ...)`.

Suffixes :

- `_flags` : a more refined version (usually introduced later),
  e.g. `newSVpvn()` v.s. `newSVpvn_flags()` ;

The perl API

Mnemonics for functions

- `([iunpr])v` : `$1` gives the source type,
  e.g. `sv_setiv(sv, iv)`, `newSVrv(rv)` ;
- `pv([nsf]?)` : `n` for length, `s` for static, `f` for format,
  e.g. `newSVpvn(buf, len)`, `sv_catpvf(sv, fmt, ...)`.

Suffixes :

- `_flags` : a more refined version (usually introduced later),
  e.g. `newSVpvn()` v.s. `newSVpvn_flags()` ;
- `_shared` : COW hash keys, e.g. `newSVpvn_share()` ;

The perl API

Mnemonics for functions

- `([iunpr])v` : `$1` gives the source type,
  e.g. `sv_setiv(sv, iv)`, `newSVrv(rv)` ;
- `pv([nsf]?)` : `n` for length, `s` for static, `f` for format,
  e.g. `newSVpvn(buf, len)`, `sv_catpvf(sv, fmt, ...)`.

Suffixes :

- `_flags` : a more refined version (usually introduced later),
  e.g. `newSVpvn()` v.s. `newSVpvn_flags()` ;
- `_shared` : COW hash keys, e.g. `newSVpvn_share()` ;
- `_mg` or `_nomg` : apply/skip get/set magic on the main arg.,
  e.g. `sv_catpv()` v.s. `sv_catpv_mg()` & `SvPV_force()` v.s. `SvPV_force_nomg()`.

The perl API

How do I...

- Check the type of an SV : `SvTYPE(sv) >= SVt_...`

The perl API

How do I...

- Check the type of an SV : `SvTYPE(sv) >= SVt_...`
- Check if a variable is defined : `SvOK(sv)`

The perl API

How do I...

- Check the type of an SV : `SvTYPE(sv) >= SVt_...`
- Check if a variable is defined : `SvOK(sv)`
- Dereference a variable :

```
if (!SvROK(sv)) croak("oops"); sv = SvRV(sv)  ;
```

The perl API

How do I...

- Check the type of an SV : `SvTYPE(sv) >= SVt_...`
- Check if a variable is defined : `SvOK(sv)`
- Dereference a variable :
  `if (!SvROK(sv)) croak("oops"); sv = SvRV(sv) ;`
- Take a reference : `newRV_inc(sv);`

The perl API

How do I...

- Check the type of an SV : `SvTYPE(sv) >= SVt_...`
- Check if a variable is defined : `SvOK(sv)`
- Dereference a variable :
  `if (!SvROK(sv)) croak("oops"); sv = SvRV(sv) ;`
- Take a reference : `newRV_inc(sv);`
- Convert to an IV, UV, NV :
  `SvIV(sv), SvUV(sv), SvNV(sv)`

The perl API

How do I...

- Check the type of an SV : `SvTYPE(sv) >= SVt_...`
- Check if a variable is defined : `SvOK(sv)`
- Dereference a variable :
  `if (!SvROK(sv)) croak("oops"); sv = SvRV(sv) ;`
- Take a reference : `newRV_inc(sv);`
- Convert to an IV, UV, NV :
  `SvIV(sv), SvUV(sv), SvNV(sv)`
- Convert to a C string : SvPV_nolen(sv) or
  `char *s; STRLEN len; s = SvPV(sv, len);`

The perl API

How do I...

- Check the type of an SV : `SvTYPE(sv) >= SVt_...`
- Check if a variable is defined : `SvOK(sv)`
- Dereference a variable :
  `if (!SvROK(sv)) croak("oops"); sv = SvRV(sv) ;`
- Take a reference : `newRV_inc(sv);`
- Convert to an IV, UV, NV :
  `SvIV(sv), SvUV(sv), SvNV(sv)`
- Convert to a C string : SvPV_nolen(sv) or
  `char *s; STRLEN len; s = SvPV(sv, len);`
- Many optimized variants when you can assume things about the objects.

The perl API

Public and private functions

Don't use functions that are not part of the public API :

The perl API

Public and private functions

Don't use functions that are not part of the public API :

- They may break in a future version of `perl` ;

The perl API

Public and private functions

Don't use functions that are not part of the public API :

- They may break in a future version of `perl` ;
- You can't link against them on Windows ;

The perl API

Public and private functions

Don't use functions that are not part of the public API :

- They may break in a future version of `perl` ;
- You can't link against them on Windows ;
- It makes Nicholas grumpy.

The perl API

Public and private functions

Don't use functions that are not part of the public API :

- They may break in a future version of `perl` ;
- You can't link against them on Windows ;
- It makes Nicholas grumpy.

To know if a function is public, use `embed.fnc` :

```
Apd     |NV     |sv_2nv         |NULLOK SV *const sv
```

The perl API

Public and private functions

Don't use functions that are not part of the public API :

- They may break in a future version of `perl` ;
- You can't link against them on Windows ;
- It makes Nicholas grumpy.

To know if a function is public, use `embed.fnc` :

```
Apd     |NV     |sv_2nv          |NULLOK SV *const sv
pMd     |SV*    |sv_2num         |NN SV *const sv
```

The perl API

Contextes

```
static AV *myarray(int a, int b) {
 AV *av = newAV();
 for (; a < b; ++a) av_push(av, newSViv(a));
 return av;
}
```

The perl API

Contextes

```
static AV *myarray(int a, int b) {
 AV *av = newAV();
 for (; a < b; ++a) av_push(av, newSViv(a));
 return av;
}
...
SV *new(int a, int b)
CODE:
 RETVAL = newRV_noinc(myarray(a, b));
OUTPUT:
 RETVAL
```

The perl API

Contextes

```
static AV *myarray(int a, int b) {
 AV *av = newAV();
 for (; a < b; ++a) av_push(av, newSViv(a));
 return av;
}
...
SV *new(int a, int b)
CODE:
 RETVAL = newRV_noinc(myarray(a, b));
OUTPUT:
 RETVAL
$ make
```

The perl API

Contextes

```
static AV *myarray(int a, int b) {
 AV *av = newAV();
 for (; a < b; ++a) av_push(av, newSViv(a));
 return av;
}
...
SV *new(int a, int b)
CODE:
 RETVAL = newRV_noinc(myarray(a, b));
OUTPUT:
 RETVAL
$ make
...
XSModule.xs: In function 'myarray':
XSModule.xs:7: error: 'my_perl' undeclared
```

The perl API

Contextes

```
static AV *myarray(pTHX_ int a, int b) {
 AV *av = newAV();
 for (; a < b; ++a) av_push(av, newSViv(a));
 return av;
}
...
SV *new(int a, int b)
CODE:
 RETVAL = newRV_noinc(myarray(aTHX_ a, b));
OUTPUT:
 RETVAL
```

The perl API

Contextes

```
static AV *myarray(pTHX_ int a, int b) {
 AV *av = newAV();
 for (; a < b; ++a) av_push(av, newSViv(a));
 return av;
}
...
SV *new(int a, int b)
CODE:
 RETVAL = newRV_noinc(myarray(aTHX_ a, b));
OUTPUT:
 RETVAL
$ make
$
```

The perl API

Contextes

```
static AV *myarray(pTHX_ int a, int b) {
#define myarray(a, b) myarray(aTHX_ (a), (b))
 AV *av = newAV();
 for (; a < b; ++a) av_push(av, newSViv(a));
 return av;
}
...
SV *new(int a, int b)
CODE:
 RETVAL = newRV_noinc(myarray(a, b));
OUTPUT:
 RETVAL
$ make
$
```

The perl API

Allocating memory

The perl API

Allocating memory

- Thread local : `Newx, Safefree`

The perl API

Allocating memory

- Thread local : `Newx, Safefree`
- Shared : `PerlMemShared_malloc, PerlMemShared_free`

Passing a list

```
$ cat My-XSModule/XSModule.xs
NV sum(...)
CODE:
 RETVAL = 0;
 while (--items >= 0) RETVAL += SvNV(ST(items));
OUTPUT:
 RETVAL
```

Passing a list

```
$ cat My-XSModule/XSModule.xs
NV sum(...)
CODE:
 RETVAL = 0;
 while (--items >= 0) RETVAL += SvNV(ST(items));
OUTPUT:
 RETVAL
$ perl -Mblib -MMy::XSModule -E '
    say My::XSModule::sum(1 .. 10)'
```

Passing a list

```
$ cat My-XSModule/XSModule.xs
NV sum(...)
CODE:
 RETVAL = 0;
 while (--items >= 0) RETVAL += SvNV(ST(items));
OUTPUT:
 RETVAL
$ perl -Mblib -MMy::XSModule -E '
    say My::XSModule::sum(1 .. 10)'
55
```

Returning a list

```
$ cat My-XSModule/XSModule.xs
...
AV *listy(void)
CODE:
 RETVAL = newAV();
 av_push(RETVAL, newSVuv(1));
 av_push(RETVAL, newSVuv(2));
 av_push(RETVAL, newSVuv(3));
OUTPUT:
 RETVAL
```

Returning a list

```
$ cat My-XSModule/XSModule.xs
...
AV *listy(void)
CODE:
 RETVAL = newAV();
 av_push(RETVAL, newSVuv(1));
 av_push(RETVAL, newSVuv(2));
 av_push(RETVAL, newSVuv(3));
OUTPUT:
 RETVAL
$ perl -Mblib -MMy::XSModule -E '
    say My::XSModule::listy()'
```

Returning a list

```
$ cat My-XSModule/XSModule.xs
...
AV *listy(void)
CODE:
 RETVAL = newAV();
 av_push(RETVAL, newSVuv(1));
 av_push(RETVAL, newSVuv(2));
 av_push(RETVAL, newSVuv(3));
OUTPUT:
 RETVAL
$ perl -Mblib -MMy::XSModule -E '
    say My::XSModule::listy()'
ARRAY(0x88e40c8)
```

Returning a list

```
$ cat My-XSModule/XSModule.xs
...
void listy()
PPCODE:
 EXTEND(SP, 3);
 PUSHs(sv_2mortal(newSVuv(1)));
 PUSHs(sv_2mortal(newSVuv(2)));
 PUSHs(sv_2mortal(newSVuv(3)));
 XSRETURN(3);
```

Returning a list

```
$ cat My-XSModule/XSModule.xs
...
void listy()
PPCODE:
 EXTEND(SP, 3);
 PUSHs(sv_2mortal(newSVuv(1)));
 PUSHs(sv_2mortal(newSVuv(2)));
 PUSHs(sv_2mortal(newSVuv(3)));
 XSRETURN(3);
$ perl -Mblib -MMy::XSModule -E '
    say My::XSModule::list()'
```

Returning a list

```
$ cat My-XSModule/XSModule.xs
...
void listy()
PPCODE:
 EXTEND(SP, 3);
 PUSHs(sv_2mortal(newSVuv(1)));
 PUSHs(sv_2mortal(newSVuv(2)));
 PUSHs(sv_2mortal(newSVuv(3)));
 XSRETURN(3);
$ perl -Mblib -MMy::XSModule -E '
    say My::XSModule::list()'
123
```

Returning a list

```
$ cat My-XSModule/XSModule.xs
...
void listy()
PPCODE:
 EXTEND(SP, 3);
 PUSHs(sv_2mortal(newSVuv(1)));
 PUSHs(sv_2mortal(newSVuv(2)));
 PUSHs(sv_2mortal(newSVuv(3)));
 XSRETURN(3);
$ perl -Mblib -MMy::XSModule -E '
    say My::XSModule::list()'
123
```

No explicit `void` in the argument list

Typemaps

Too cumbersome? Use typemaps instead :

```
double add(double x, double y)
CODE:
 RETVAL = x + y;
OUTPUT:
 RETVAL
```

Typemaps

Too cumbersome? Use typemaps instead :

```
double add(double x, double y)
CODE:
 RETVAL = x + y;
OUTPUT:
 RETVAL
```

- Typemaps provide boilerplate Perl -> C and C -> Perl variable conversions ;

Typemaps

Too cumbersome? Use typemaps instead :

```
double add(double x, double y)
CODE:
 RETVAL = x + y;
OUTPUT:
 RETVAL
```

- Typemaps provide boilerplate Perl -> C and C -> Perl variable conversions ;
- Perl comes with a default typemap that covers many C types

Typemaps

Too cumbersome? Use typemaps instead :

```
double add(double x, double y)
CODE:
 RETVAL = x + y;
OUTPUT:
 RETVAL
```

- Typemaps provide boilerplate Perl -> C and C -> Perl variable conversions ;
- Perl comes with a default typemap that covers many C types
- Secret doc :

`http://search.cpan.org/dist/perl/ext/XS/Typemap/Typemap.xs`

Typemaps

Generated C code

```
$ cat My-XSModule/XSModule.c
XS(XS_My__XSModule_add);
XS(XS_My__XSModule_add)
{
 dXSARGS;
 if (items != 2)
  Perl_croak(aTHX_ "Usage: My::XSModule::add(x, y)");
 {
  double RETVAL; dXSTARG;
  double x = (double)SvNV(ST(0));
  double y = (double)SvNV(ST(1));
  RETVAL = x + y;
  XSprePUSH; PUSHn((double)RETVAL);
 }
 XSRETURN(1);
}
```

Typemaps

Generating XS subs from C prototypes

```
$ cat My-XSModule/XSModule.xs
double sin(double x);
```

Typemaps

Generating XS subs from C prototypes

```
$ cat My-XSModule/XSModule.xs
double sin(double x);
$ cat My-XSModule/XSModule.c
XS(XS_My__XSModule_sin);
XS(XS_My__XSModule_sin)
{
 dXSARGS;
 if (items != 1) Perl_croak(aTHX_ "Usage: My::XSModule::sin(x)");
 {
  double RETVAL; dXSTARG;
  double x = (double)SvNV(ST(0));
  RETVAL = sin(x);
  XSprePUSH; PUSHn((double)RETVAL);
 }
 XSRETURN(1);
}
```

Typemaps

Write your own typemap

```
struct my_xs_obj_t *    MYXSOBJ
```

Typemaps

Write your own typemap

```
struct my_xs_obj_t *    MYXSOBJ

INPUT
MYXSOBJ
 if (sv_derived_from($arg, \"My::XSModule::Object\"))
  $var = INT2PTR($type, SvIV((SV *) SvRV($arg)));
 else
  croak(\"Couldn't coerce $arg in a $type\");
```

Typemaps

Write your own typemap

```
struct my_xs_obj_t *    MYXSOBJ

INPUT
MYXSOBJ
 if (sv_derived_from($arg, \"My::XSModule::Object\"))
  $var = INT2PTR($type, SvIV((SV *) SvRV($arg)));
 else
  croak(\"Couldn't coerce $arg in a $type\");

OUTPUT
MYXSOBJ
 sv_setref_iv($arg, \"My::XSModule::Object\", PTR2IV($var));
```

Typemaps

Write your own typemap

```
struct my_xs_obj_t *    MYXSOBJ

INPUT
MYXSOBJ
 if (sv_derived_from($arg, \"My::XSModule::Object\"))
  $var = INT2PTR($type, SvIV((SV *) SvRV($arg)));
 else
  croak(\"Couldn't coerce $arg in a $type\");

OUTPUT
MYXSOBJ
 sv_setref_iv($arg, \"My::XSModule::Object\", PTR2IV($var));
```

- Tabs between the C type and `MYSOBJ` in the declaration ;

Typemaps

Write your own typemap

```
struct my_xs_obj_t *    MYXSOBJ

INPUT
MYXSOBJ
 if (sv_derived_from($arg, \"My::XSModule::Object\"))
  $var = INT2PTR($type, SvIV((SV *) SvRV($arg)));
 else
  croak(\"Couldn't coerce $arg in a $type\");

OUTPUT
MYXSOBJ
 sv_setref_iv($arg, \"My::XSModule::Object\", PTR2IV($var));
```

- Tabs between the C type and `MYSOBJ` in the declaration ;
- You can embed Perl code with `${...}` ;

Typemaps

Write your own typemap

```
struct my_xs_obj_t *    MYXSOBJ

INPUT
MYXSOBJ
 if (sv_derived_from($arg, \"My::XSModule::Object\"))
  $var = INT2PTR($type, SvIV((SV *) SvRV($arg)));
 else
  croak(\"Couldn't coerce $arg in a $type\");

OUTPUT
MYXSOBJ
 sv_setref_iv($arg, \"My::XSModule::Object\", PTR2IV($var));
```

- Tabs between the C type and `MYSOBJ` in the declaration ;
- You can embed Perl code with `${...}` ;
- Extends the default typemap.

Intermediate XS recipes

Detecting a C compiler

With `ExtUtils::CBuilder`

```
$ cat My-XSModule/Makefile.PL
...
use ExtUtils::MakeMaker;
use ExtUtils::CBuilder;
my $cb = ExtUtils::CBuilder->new(quiet => 1);
my @C  = $cb->have_compiler ? 'XSModule.c' : ();
WriteMakefile(C => \@C, ...);
```

Detecting a C compiler

With `ExtUtils::CBuilder`

```
$ cat My-XSModule/Makefile.PL
...
use ExtUtils::MakeMaker;
use ExtUtils::CBuilder;
my $cb = ExtUtils::CBuilder->new(quiet => 1);
my @C  = $cb->have_compiler ? 'XSModule.c' : ();
WriteMakefile(C => \@C, ...);
```

- Pro: Sturdier, prettier, comes for free with `Module::Build` and `perl 5.10` ;

Detecting a C compiler

With `ExtUtils::CBuilder`

```
$ cat My-XSModule/Makefile.PL
...
use ExtUtils::MakeMaker;
use ExtUtils::CBuilder;
my $cb = ExtUtils::CBuilder->new(quiet => 1);
my @C  = $cb->have_compiler ? 'XSModule.c' : ();
WriteMakefile(C => \@C, ...);
```

- Pro: Sturdier, prettier, comes for free with `Module::Build` and `perl 5.10` ;
- Con: Otherwise you need a `configure_requires` aware toolchain.

Detecting a C compiler

With `Devel::CheckLib`

```
$ cat My-XSModule/Makefile.PL
...
use ExtUtils::MakeMaker;
use Devel::CheckLib;
my @C = eval { assert_lib; 1 } ? 'XSModule.c' : ();
WriteMakefile(C => \@C, ...);
```

Detecting a C compiler

With `Devel::CheckLib`

```
$ cat My-XSModule/Makefile.PL
...
use ExtUtils::MakeMaker;
use Devel::CheckLib;
my @C = eval { assert_lib; 1 } ? 'XSModule.c' : ();
WriteMakefile(C => \@C, ...);
```

- Pro: Also sturdier, prettier, more versatile, provides an helper script to easily bundle it with your distribution ;

Detecting a C compiler

With `Devel::CheckLib`

```
$ cat My-XSModule/Makefile.PL
...
use ExtUtils::MakeMaker;
use Devel::CheckLib;
my @C = eval { assert_lib; 1 } ? 'XSModule.c' : ();
WriteMakefile(C => \@C, ...);
```

- Pro: Also sturdier, prettier, more versatile, provides an helper script to easily bundle it with your distribution ;
- Con: Otherwise you need a `configure_requires` aware toolchain.

Pure Perl fallbacks

```
$ cat My-XSModule/lib/My/XSModule.pm
...
BEGIN {
 unless (eval {
  require XSLoader;
  XSLoader::load(__PACKAGE__, $VERSION);
  1
 }) {
  *zap    = *zap_pp;
  *flurbz = *flurbz_pp;
 }
}
sub zap_pp    { ... }
sub flurbz_pp { ... }
```

Detecting a C library

With `Devel::CheckLib`

```
use Devel::CheckLib;
check_lib_or_exit(
 lib    => 'zap',
 header => 'libzap.h'
);
```

Detecting an operating system

- The trivial way :

```
die "OS unsupported" if $^O ne 'linux';
```

Detecting an operating system

- The trivial way :

```
die "OS unsupported" if $^O ne 'linux';
```

- The cleaner way :

```
use Devel::CheckOS;
die_if_os_isnt('Unix')  # OS families
die_if_os_is('freebsd') # OS names
```

Detecting an operating system

- The trivial way :

```
die "OS unsupported" if $^O ne 'linux';
```

- The cleaner way :

```
use Devel::CheckOS;
die_if_os_isnt('Unix')  # OS families
die_if_os_is('freebsd') # OS names
```

Still easily redistributable.

Detecting ActiveState Perl

```
if (eval { require ActivePerl; 1 }) {
 my $build = $ActivePerl::VERSION;
 if (defined $build) {
   ...
 }
}
```

Using another XS extension in XS

On the producer's side : `Makefile.PL`

```
$ cat My-XSTool/Makefile.PL
...
use ExtUtils::Depends;
my $ed = ExtUtils::Depends->new('My::XSTool');
$ed->add_xs('XSTool.xs');
$ed->install('my-xstool.h');
WriteMakefile($ed->get_makefile_vars, ...);
```

Using another XS extension in XS

On the producer's side : `.pm` file

```
$ cat My-XSTool/lib/My/XSTool.pm
package My::XSTool;
use strict;
use warnings;
our ($VERSION, @ISA);
sub dl_load_flags { 0x01 }
BEGIN {
 $VERSION = '0.01';
 push @ISA, 'DynaLoader';
 __PACKAGE__->bootstrap($VERSION);
}
...
```

Using another XS extension in XS

On the consumer's side

```
$ cat My-XSModule/Makefile.PL
...
use ExtUtils::Depends;
my $ed = ExtUtils::Depends->new(
 'My::XSModule' => 'My::XSTool', 'My::XSTool2'
);
WriteMakefile($ed->get_makefile_vars, ...);
```

What you will need

Always keep at hand :

What you will need

Always keep at hand :

- A `git` checkout of the source ;

What you will need

Always keep at hand :

- A `git` checkout of the source ;
- An extensive set of `perl`s to test against ;

What you will need

Always keep at hand :

- A `git` checkout of the source ;
- An extensive set of `perl`s to test against ;
- Debugging tools :

What you will need

Always keep at hand :

- A `git` checkout of the source ;
- An extensive set of `perl`s to test against ;
- Debugging tools :
  - `gdb` ;

What you will need

Always keep at hand :

- A `git` checkout of the source ;
- An extensive set of `perls` to test against ;
- Debugging tools :
  - `gdb` ;
  - `valgrind` ;

What you will need

Always keep at hand :

- A `git` checkout of the source ;
- An extensive set of `perls` to test against ;
- Debugging tools :
  - `gdb` ;
  - `valgrind` ;
  - `strace`.

Configuring your `git` checkout

```
$ git clone git://perl5.git.perl.org/perl.git
$ cd perl
$ git branch
* blead
  ...
$ git branch --track maint-5.10 origin/maint-5.10
```

Building perl

```
$ sh Configure -des \
    -Dprefix=$HOME/perl/builds/$features/$version \
    $OPTS
```

Building perl

```
$ sh Configure -des \
    -Dprefix=$HOME/perl/builds/$features/$version \
    $OPTS
$ make -jX
```

Building perl

```
$ sh Configure -des \
    -Dprefix=$HOME/perl/builds/$features/$version \
    $OPTS
$ make -jX
$ TEST_JOBS=X make test_harness # or "make test"
```

Building perl

```
$ sh Configure -des \
    -Dprefix=$HOME/perl/builds/$features/$version \
    $OPTS
$ make -jX
$ TEST_JOBS=X make test_harness # or "make test"
$ make install
```

Building `perl`

`Configure` options

Feature	`Configure` option	Hit modifier

Building `perl`

`Configure` options

Feature	`Configure` option	Hit modifier
Debugging	`-DDEBUGGING`	
	`-Doptimize=-g3`	`x 3`

Building perl

Configure options

Feature	Configure option	Hit modifier
Debugging	-DDEBUGGING	
	-Doptimize=-g3	x 3
Multiplicity	-Dusemultiplicity	
Threads	-Dusethreads	x 3

Building perl

Configure options

Feature	Configure option	Hit modifier
Debugging	-DDEBUGGING	
	-Doptimize=-g3	x 3
Multiplicity	-Dusemultiplicity	
Threads	-Duseithreads	x 3
64 bits	-Dusemorebits	x 2

Building perl

Configure options

Feature	Configure option	Hit modifier
Debugging	-DDEBUGGING	
	-Doptimize=-g3	x 3
Multiplicity	-Dusemultiplicity	
Threads	-Duseithreads	x 3
64 bits	-Dusemorebits	x 2
Shared libperl	-Duseshrplib	x 2

Building perl

Configure options

Feature	Configure option	Hit modifier
Debugging	-DDEBUGGING	
	-Doptimize=-g3	x 3
Multiplicity	-Dusemultiplicity	
Threads	-Duseithreads	x 3
64 bits	-Dusemorebits	x 2
Shared libperl	-Duseshrplib	x 2
Memory management	-Dusemymalloc	x 2

Building perl

Configure options

Feature	Configure option	Hit modifier
Debugging	-DDEBUGGING	
	-Doptimize=-g3	x 3
Multiplicity	-Dusemultiplicity	
Threads	-Duseithreads	x 3
64 bits	-Dusemorebits	x 2
Shared libperl	-Duseshrplib	x 2
Memory management	-Dusemymalloc	x 2
Total		x 72

Building `perl`

Which `perl` version to support?

Official releases :

Building `perl`

Which `perl` version to support?

Official releases :

- `5.005` and older ;

Building `perl`

Which `perl` version to support?

Official releases :

- `5.005` and older ;
- `5.6` : 3 versions ;

Building `perl`

Which `perl` version to support?

Official releases :

- `5.005` and older ;
- `5.6` : 3 versions ;
- `5.8` : 9 versions ;

Building `perl`

Which `perl` version to support?

Official releases :

- `5.005` and older ;
- `5.6` : 3 versions ;
- `5.8` : 9 versions ;
  - `5.8.0` : Kinda broken ;

Building `perl`

Which `perl` version to support?

Official releases :

- `5.005` and older ;
- `5.6` : 3 versions ;
- `5.8` : 9 versions ;
  - `5.8.0` : Kinda broken ;
  - `5.8.1-3` : UTF-8 broken ;

Building `perl`

Which `perl` version to support?

Official releases :

- `5.005` and older ;
- `5.6` : 3 versions ;
- `5.8` : 9 versions ;
  - `5.8.0` : Kinda broken ;
  - `5.8.1-3` : UTF-8 broken ;
  - `5.8.4-9` : OK.

Building `perl`

Which `perl` version to support?

Official releases :

- `5.005` and older ;
- `5.6` : 3 versions ;
- `5.8` : 9 versions ;
  - `5.8.0` : Kinda broken ;
  - `5.8.1-3` : UTF-8 broken ;
  - `5.8.4-9` : OK.
- `5.10` : 2 versions ;

Building `perl`

Which `perl` version to support?

Official releases :

- `5.005` and older ;
- `5.6` : 3 versions ;
- `5.8` : 9 versions ;
  - `5.8.0` : Kinda broken ;
  - `5.8.1-3` : UTF-8 broken ;
  - `5.8.4-9` : OK.
- `5.10` : 2 versions ;
- Check `blead` regularly.

Building `perl`

Which `perl` version to support?

Official releases :

- `5.005` and older ;
- `5.6` : 3 versions ;
- `5.8` : 9 versions ;
  - `5.8.0` : Kinda broken ;
  - `5.8.1-3` : UTF-8 broken ;
  - `5.8.4-9` : OK.
- `5.10` : 2 versions ;
- Check `blead` regularly.

Vendors :

Building `perl`

Which `perl` version to support?

Official releases :

- `5.005` and older ;
- `5.6` : 3 versions ;
- `5.8` : 9 versions ;
  - `5.8.0` : Kinda broken ;
  - `5.8.1-3` : UTF-8 broken ;
  - `5.8.4-9` : OK.
- `5.10` : 2 versions ;
- Check `blead` regularly.

Vendors :

- ActiveState ;

Building `perl`

Which `perl` version to support?

Official releases :

- `5.005` and older ;
- `5.6` : 3 versions ;
- `5.8` : 9 versions ;
  - `5.8.0` : Kinda broken ;
  - `5.8.1-3` : UTF-8 broken ;
  - `5.8.4-9` : OK.
- `5.10` : 2 versions ;
- Check `blead` regularly.

Vendors :

- ActiveState ;
- RH :-)

Building `perl`

Sum it up!

If you want to support all perls from `5.8.1` to now, you'll need

$$72 * (9 + 2 + 1) + 2 = 866$$

builds to test your extension.
More realistically :

$$2 * (8 + 1) + 5 * 2 + 1 = 29$$

Improve portability

Devel:PPPort

`Devel:PPPort` generates a C/Perl polyglot `ppport.h` file that can be used to :

Improve portability

Devel:PPPort

`Devel:PPPort` generates a C/Perl polyglot `ppport.h` file that can be used to :

- audit your code for macros/symbols not present in older perls when you run it against your XS code as a Perl script ;

Improve portability

Devel:PPPort

`Devel:PPPort` generates a C/Perl polyglot `ppport.h` file that can be used to :

- audit your code for macros/symbols not present in older perls when you run it against your XS code as a Perl script ;
- provides back-compatiblity macros when you `#include` it in your XS code.

Improve portability

Devel:PPPort

`Devel:PPPort` generates a C/Perl polyglot `ppport.h` file that can be used to :

- audit your code for macros/symbols not present in older perls when you run it against your XS code as a Perl script ;
- provides back-compatiblity macros when you `#include` it in your XS code.

`perl ppport.h --strip` removes all unneeded Perl and doc from `ppport.h`.

Improve portability

Devel:PPPort

`Devel:PPPort` generates a C/Perl polyglot `ppport.h` file that can be used to :

- audit your code for macros/symbols not present in older perls when you run it against your XS code as a Perl script ;
- provides back-compatiblity macros when you `#include` it in your XS code.

`perl ppport.h --strip` removes all unneeded Perl and doc from `ppport.h`.

`perl ppport.h --help` and `perldoc ppport.h` show all options.

Improve portability

```
$ perl -MDevel::PPPort -e 'Devel::PPPort::WriteFile()'
```

Improve portability

```
$ perl -MDevel::PPPort -e 'Devel::PPPort::WriteFile()'
$ perl ppport.h XSModule.xs --compat-version=5.005
```

Improve portability

```
$ perl -MDevel::PPPort -e 'Devel::PPPort::WriteFile()'
$ perl ppport.h XSModule.xs --compat-version=5.005
Scanning XSModule.xs ...
=== Analyzing XSModule.xs ===
Uses newSVpvn_share, which depends on PERL_HASH, STMT_END, STMT_START,
SvIVX, SvPOK_on, newSVpvn, newSV, newSVpv, sv_upgrade
File needs newSVpvn_share, adding static request
Needs to include 'ppport.h'
Analysis completed
Suggested changes:
-- XSModule.xs
+++ XSModule.xs.patched
@@ -2,6 +2,8 @@
 #include "EXTERN.h"
 #include "perl.h"
 #include "XSUB.h"
+#define NEED_newSVpvn_share
+#include "ppport.h"
...
```

Testing tools

Testing tools

- Make good use of your extensive set of `perl`s ;

Testing tools

- Make good use of your extensive set of `perl`s ;
- If you have pure-Perl wrappers, you can use `Test::NoXS` to test that they are correctly used as fallbacks :

```
$ cat My-XSModule/t/pp.t
#!perl
use strict;
use warnings;
use Test::NoXS 'My::XSModule';
use My::XSModule qw/zap/;
...
```

Testing tools

Test::LeakTrace

- Test::LeakTrace checks for leaks of SVs :

```
$ cat My-XSModule/t/leak_sv.t
#!perl
use strict;
use warnings;
use Test::LeakTrace;
use My::XSModule qw/zap/;
no_leaks_ok {
 zap();
} 'No SVs were leaked';
```

Testing tools

Test::Valgrind

- Test::Valgrind can run valgrind on your extension to check for read/write errors and leaks of allocated memory.

```
$ cat My-XSModule/t/leak_vg.t
#!perl
use strict;
use warnings;
use Test::Valgrind;
use My::XSModule qw/zap/;
zap();
```

Code coverage

`Devel::Cover` automatically invokes `gcov` to get statement coverage of the XS files.

```
$ cover -test
$ $web_browser cover_db/coverage.html
```

Profile your extension

Sadly, `Devel::NYTProf` doesn't work with XS extensions. And `gprof` is useless for shared libraries.

But on linux you can use `oprofile` :

```
$ export SESS=/tmp/oprofile
```

Profile your extension

Sadly, `Devel::NYTProf` doesn't work with XS extensions. And `gprof` is useless for shared libraries.

But on linux you can use `oprofile` :

```
$ export SESS=/tmp/oprofile
$ sudo opcontrol --reset
```

Profile your extension

Sadly, `Devel::NYTProf` doesn't work with XS extensions. And `gprof` is useless for shared libraries.

But on linux you can use `oprofile` :

```
$ export SESS=/tmp/oprofile
$ sudo opcontrol --reset
$ sudo opcontrol --start --sesssion-dir=$SESS \
                 --no-vmlinux --callgraph=10  \
 --image=$PWD/blib/arch/auto/My/XSModule/XSModule.so
```

Profile your extension

Sadly, `Devel::NYTProf` doesn't work with XS extensions. And `gprof` is useless for shared libraries.

But on linux you can use `oprofile` :

```
$ export SESS=/tmp/oprofile
$ sudo opcontrol --reset
$ sudo opcontrol --start --sesssion-dir=$SESS \
                 --no-vmlinux --callgraph=10  \
 --image=$PWD/blib/arch/auto/My/XSModule/XSModule.so
$ make test
```

Profile your extension

Sadly, `Devel::NYTProf` doesn't work with XS extensions. And `gprof` is useless for shared libraries.

But on linux you can use `oprofile` :

```
$ export SESS=/tmp/oprofile
$ sudo opcontrol --reset
$ sudo opcontrol --start --sesssion-dir=$SESS \
                 --no-vmlinux --callgraph=10  \
 --image=$PWD/blib/arch/auto/My/XSModule/XSModule.so
$ make test
$ sudo opcontrol --dump --session-dir=$SESS
```

Profile your extension

Sadly, `Devel::NYTProf` doesn't work with XS extensions. And `gprof` is useless for shared libraries.

But on linux you can use `oprofile` :

```
$ export SESS=/tmp/oprofile
$ sudo opcontrol --reset
$ sudo opcontrol --start --sesssion-dir=$SESS \
                 --no-vmlinux --callgraph=10  \
 --image=$PWD/blib/arch/auto/My/XSModule/XSModule.so
$ make test
$ sudo opcontrol --dump --session-dir=$SESS
$ sudo opreport --session-dir=$SESS -c
```

Debugging segfaults

With gdb

- Reduce the problem to a minimal test case ;

Debugging segfaults

With `gdb`

- Reduce the problem to a minimal test case ;
- Make sure you have a `perl` compiled with `-DDEBUGGING` (= assertions + debugging symbols) available, as well as its source tree.

Debugging segfaults

With gdb

- Reduce the problem to a minimal test case ;
- Make sure you have a `perl` compiled with `-DDEBUGGING` (= assertions + debugging symbols) available, as well as its source tree.
- Build your extension with this `perl` :

```
$ perl-dbg Makefile.PL && make
```

Debugging segfaults

With gdb

- Reduce the problem to a minimal test case ;
- Make sure you have a `perl` compiled with `-DDEBUGGING` (= assertions + debugging symbols) available, as well as its source tree.
- Build your extension with this `perl` :

  ```
  $ perl-dbg Makefile.PL && make
  ```

- Run it with gdb.

Debugging segfaults

```
$ gdb --args perl-dbg -Mblib segv.t
```

Debugging segfaults

```
$ gdb --args perl-dbg -Mblib segv.t
GNU gdb 6.8
...
(gdb)
```

Debugging segfaults

```
$ gdb --args perl-dbg -Mblib segv.t
GNU gdb 6.8
...
(gdb) r
Starting program: /home/you/perl/builds/dbg/bin/perl-dbg -Mblib segv.t
Program received signal SIGSEGV, Segmentation fault.
0xb7f49988 in oops () at XSModule.xs:8
8           char *x = NULL; *x = 'z';
(gdb)
```

Debugging segfaults

```
$ gdb --args perl-dbg -Mblib segv.t
GNU gdb 6.8
...
(gdb) r
Starting program: /home/you/perl/builds/dbg/bin/perl-dbg -Mblib segv.t
Program received signal SIGSEGV, Segmentation fault.
0xb7f49988 in oops () at XSModule.xs:8
8         char *x = NULL; *x = 'z';
(gdb) bt
#0  0xb7f49988 in oops () at XSModule.xs:8
#1  0xb7f4a20d in XS_My__XSModule_add (cv=0x8fdb370) at XSModule.xs:17
#2  0x0811f703 in Perl_pp_entersub () at pp_hot.c:2847
#3  0x080c5e31 in Perl_runops_debug () at dump.c:1931
#4  0x080fe981 in S_run_body (oldscope=1) at perl.c:2384
#5  0x080fdf47 in perl_run (my_perl=0x8f4d008) at perl.c:2302
#6  0x08062641 in main (argc=3, argv=0xbfc973e4, env=0xbfc973f4)
    at perlmain.c:113
```

Debugging memory errors

With `Test::Valgrind`

```
$ perl-dbg -MTest::Valgrind -Mblib wrong.t
```

Debugging memory errors

With `Test::Valgrind`

```
$ perl-dbg -MTest::Valgrind -Mblib wrong.t
1..15
# Using valgrind 3.5.0 located at /usr/bin/valgrind
ok 1 - InvalidFree
ok 2 - MismatchedFree
ok 3 - InvalidRead
not ok 4 - InvalidWrite
#   Failed test 'InvalidWrite'
#          got: 1
#     expected: 0
#   Invalid write of size 1
#     oops (XSModule.so) [XSModule.xs:9]
#     XS_My__XSModule_add (XSModule.so) [XSModule.xs:18]
#     Perl_pp_entersub (perl-dbg) [pp_hot.c:2847]
#     Perl_runops_debug (perl-dbg) [dump.c:1931]
#     S_run_body (perl-dbg) [perl.c:2384]
#     perl_run (perl-dbg) [perl.c:2302]
#     main (perl-dbg) [perlmain.c:113]
...
```

Advanced XS recipes

Thread-safe global data

```
#define MY_CXT_KEY "My::XSModule::_guts" XS_VERSION
typedef struct { int a; char *b; } my_cxt_t;
START_MY_CXT
```

Thread-safe global data

```
#define MY_CXT_KEY "My::XSModule::_guts" XS_VERSION
typedef struct { int a; char *b; } my_cxt_t;
START_MY_CXT
...
# XS section
BOOT: {
 MY_CXT_INIT;
 MY_CXT.a = 0; MY_CXT.b = NULL;
}
```

Thread-safe global data

```
#define MY_CXT_KEY "My::XSModule::_guts" XS_VERSION
typedef struct { int a; char *b; } my_cxt_t;
START_MY_CXT
...
# XS section
BOOT: {
 MY_CXT_INIT;
 MY_CXT.a = 0; MY_CXT.b = NULL;
}

int get_a()
CODE:
 dMY_CXT;
 RETVAL = MY_CXT.a;
OUTPUT
 RETVAL
```

Thread-safe global data

Cloning thread-local storage

```
#ifdef USE_ITHREADS

void
CLONE(...)
CODE:
 MY_CXT_CLONE;

#endif
```

Thread-safe global data

Cloning thread-local storage

```
#ifdef USE_ITHREADS

void
CLONE(...)
CODE:
 MY_CXT_CLONE;

#endif
```

Mind the spaces!

Cloning

How it works

- A new interpreter is allocated ;

Cloning

How it works

- A new interpreter is allocated ;
- All the variables are copied from the old interpreter to the new ;

Cloning

How it works

- A new interpreter is allocated ;
- All the variables are copied from the old interpreter to the new ;
- Each time a variable is dup'd, an `old => new` entry is stored in a pointer table (so that the same variable isn't duplicated several times) ;

Cloning

How it works

- A new interpreter is allocated ;
- All the variables are copied from the old interpreter to the new ;
- Each time a variable is dup'd, an `old => new` entry is stored in a pointer table (so that the same variable isn't duplicated several times) ;
- The `CLONE` method is called in the context of the new interpreter ;

Cloning

How it works

- A new interpreter is allocated ;
- All the variables are copied from the old interpreter to the new ;
- Each time a variable is dup'd, an `old => new` entry is stored in a pointer table (so that the same variable isn't duplicated several times) ;
- The `CLONE` method is called in the context of the new interpreter ;
- The pointer table is destroyed.

Cloning

Caching a package variable

```
typedef struct { SV *ver; SV *pkg; } my_cxt_t;
STATIC void init_cxt(pMY_CXT) {
 MY_CXT.ver = GvSV(gv_fetchpvs("My::XSModule::VERSION",0,SVt_PV));
 MY_CXT.pkg = gv_stashpvs("My::XSModule", 1);
}

BOOT: {
 MY_CXT_INIT;
 init_cxt(MY_CXT);
}

void
CLONE(...)
CODE:
 MY_CXT_CLONE;
 init_cxt(MY_CXT);
```

Cloning

Keeping Perl variables in thread-local storage

```
typedef struct {
 SV *cb; tTHX owner;
} my_cxt_t;

BOOT: {
 MY_CXT_INIT;
 MY_CXT.cb    = NULL;
 MY_CXT.owner = aTHX;
}

void
set_cb(SV *cb)
CODE:
 dMY_CXT;
 if (SvROK(cb))
  MY_CXT.cb = SvRV(cb);
```

Cloning

Keeping Perl variables in thread-local storage

```
typedef struct {
 SV *cb; tTHX owner;
} my_cxt_t;

BOOT: {
 MY_CXT_INIT;
 MY_CXT.cb    = NULL;
 MY_CXT.owner = aTHX;
}

void
set_cb(SV *cb)
CODE:
 dMY_CXT;
 if (SvROK(cb))
  MY_CXT.cb = SvRV(cb);
```

```
void
CLONE(...)
CODE:
 SV *cb_dup;
 {
  dMY_CXT; CLONE_PARAMS p;
  p.stashes    = NULL;
  p.flags      = 0;
  p.proto_perl = MY_CXT.owner;
  cb_dup = sv_dup(MY_CXT.cb, &p);
  SvREFCNT_inc(cb_dup);
 }
 {
  MY_CXT_CLONE;
  MY_CXT.cb    = cb_dup;
  MY_CXT.owner = aTHX;
 }
```

Cloning

Keeping Perl variables in thread-local storage

```
typedef struct {
 SV *cb; tTHX owner;
} my_cxt_t;

BOOT: {
 MY_CXT_INIT;
 MY_CXT.cb    = NULL;
 MY_CXT.owner = aTHX;
}

void
set_cb(SV *cb)
CODE:
 dMY_CXT;
 if (SvROK(cb))
  MY_CXT.cb = SvRV(cb);
```

```
void
CLONE(...)
CODE:
 SV *cb_dup;
 {
  dMY_CXT; CLONE_PARAMS p;
  p.stashes    = NULL;
  p.flags      = 0;
  p.proto_perl = MY_CXT.owner;
  cb_dup = sv_dup(MY_CXT.cb, &p);
  SvREFCNT_inc(cb_dup);
 }
 {
  MY_CXT_CLONE;
  MY_CXT.cb    = cb_dup;
  MY_CXT.owner = aTHX;
 }
```

Cloning

Keeping Perl variables in thread-local storage

```
typedef struct {
 SV *cb; tTHX owner;
} my_cxt_t;

BOOT: {
 MY_CXT_INIT;
 MY_CXT.cb    = NULL;
 MY_CXT.owner = aTHX;
}

void
set_cb(SV *cb)
CODE:
 dMY_CXT;
 if (SvROK(cb))
  MY_CXT.cb = SvRV(cb);
```

```
void
CLONE(...)
CODE:
 SV *cb_dup;
 {
  dMY_CXT; CLONE_PARAMS p;
  p.stashes    = NULL;
  p.flags      = 0;
  p.proto_perl = MY_CXT.owner;
  cb_dup = sv_dup(MY_CXT.cb, &p);
  SvREFCNT_inc(cb_dup);
 }
 {
  MY_CXT_CLONE;
  MY_CXT.cb    = cb_dup;
  MY_CXT.owner = aTHX;
 }
```

Cloning

Remarks

- The call to `sv_dup()` relies on hitting the pointer table cache ("relinking SVs") ;

Cloning

Remarks

- The call to `sv_dup()` relies on hitting the pointer table cache ("relinking SVs") ;
- For fork emulation on Windows, this work only with 5.10.1 and higher.

The optree

A Perl program is represented by a tree of OPs.

The optree

A Perl program is represented by a tree of OPs.

```
$ perl -MO=Concise -e 'my $x = foo()'
```

The optree

A Perl program is represented by a tree of OPs.

```
$ perl -MO=Concise -e 'my $x = foo()'
8  <@> leave[1 ref] vKP/REFC ->(end)
1     <0> enter ->2
2     <;> nextstate(main 1 -e:1) v:{ ->3
7     <2> sassign vKS/2 ->8
5        <1> entersub[t3] sKS/TARG,1 ->6
-           <1> ex-list sK ->5
3              <0> pushmark s ->4
-              <1> ex-rv2cv sK ->-
4                 <#> gv[*foo] s/EARLYCV ->5
6        <0> padsv[$x:1,2] sRM*/LVINTRO ->7
-e syntax OK
```

The optree

A Perl program is represented by a tree of OPs.

```
$ perl -MO=Concise -e 'my $x = foo()'
8  <@> leave[1 ref] vKP/REFC ->(end)
1     <0> enter ->2
2     <;> nextstate(main 1 -e:1) v:{ ->3
7     <2> sassign vKS/2 ->8
5        <1> entersub[t3] sKS/TARG,1 ->6
-           <1> ex-list sK ->5
3              <0> pushmark s ->4
-              <1> ex-rv2cv sK ->-
4                 <#> gv[*foo] s/EARLYCV ->5
6        <0> padsv[$x:1,2] sRM*/LVINTRO ->7
-e syntax OK
```

Pluggable `perl`

The compilation process

Pluggable perl

The compilation process

- The source is parsed by the lexer/tokenizer, that generate the OPs on the fly (bottom-up) ;

Pluggable perl

The compilation process

- The source is parsed by the lexer/tokenizer, that generate the OPs on the fly (bottom-up) ;
- When the context is known, it's propagated to the children (top-down) ;

Pluggable perl

The compilation process

- The source is parsed by the lexer/tokenizer, that generate the OPs on the fly (bottom-up) ;
- When the context is known, it's propagated to the children (top-down) ;
- The peephole optimizer establishes the execution order ;

Pluggable `perl`

The compilation process

- The source is parsed by the lexer/tokenizer, that generate the `OP`s on the fly (bottom-up) ;
- When the context is known, it's propagated to the children (top-down) ;
- The peephole optimizer establishes the execution order ;
- Each `OP` contains a pointer to the C function called at run-time to handle it.

You can hook into any of those phases :

Pluggable `perl`

The compilation process

- The source is parsed by the lexer/tokenizer, that generate the `OP`s on the fly (bottom-up) ;
- When the context is known, it's propagated to the children (top-down) ;
- The peephole optimizer establishes the execution order ;
- Each `OP` contains a pointer to the C function called at run-time to handle it.

You can hook into any of those phases :

- Before parsing : source filters ;

Pluggable `perl`

The compilation process

- The source is parsed by the lexer/tokenizer, that generate the `OP`s on the fly (bottom-up) ;
- When the context is known, it's propagated to the children (top-down) ;
- The peephole optimizer establishes the execution order ;
- Each `OP` contains a pointer to the C function called at run-time to handle it.

You can hook into any of those phases :

- Before parsing : source filters ;
- At `OP` creation : check functions ;

Pluggable `perl`

The compilation process

- The source is parsed by the lexer/tokenizer, that generate the `OP`s on the fly (bottom-up) ;
- When the context is known, it's propagated to the children (top-down) ;
- The peephole optimizer establishes the execution order ;
- Each `OP` contains a pointer to the C function called at run-time to handle it.

You can hook into any of those phases :

- Before parsing : source filters ;
- At `OP` creation : check functions ;
- Replace the peephole optimizer ;

Pluggable perl

The compilation process

- The source is parsed by the lexer/tokenizer, that generate the OPs on the fly (bottom-up) ;
- When the context is known, it's propagated to the children (top-down) ;
- The peephole optimizer establishes the execution order ;
- Each OP contains a pointer to the C function called at run-time to handle it.

You can hook into any of those phases :

- Before parsing : source filters ;
- At OP creation : check functions ;
- Replace the peephole optimizer ;
- At OP execution : pp functions.

Pluggable `perl`

Check functions

Pluggable perl

Check functions

- `OP *(*check)(pTHX_ OP *)`

Pluggable perl

Check functions

- `OP *(*check)(pTHX_ OP *)`
- Stored in the `PL_check` array, indexed by the `OP` type ;

Pluggable perl

Check functions

- `OP *(*check)(pTHX_ OP *)`
- Stored in the `PL_check` array, indexed by the `OP` type ;
- Called just after an `OP` of the corresponding type is allocated.

Example : `inchworm.pm`

Goal : Making `~~` really equivalent to `scalar`.

Example : `inchworm.pm`

Goal : Making `~~` really equivalent to `scalar`.

But how it is compiled, anyway?

Example : inchworm.pm

Goal : Making ~~ really equivalent to scalar.

But how it is compiled, anyway?

```
$ perl5.10.0 -MO=Concise -e 'my $x = ~~@a'
```

Example : inchworm.pm

Goal : Making ~~ really equivalent to scalar.

But how it is compiled, anyway?

```
$ perl5.10.0 -MO=Concise -e 'my $x = ~~@a'
9  <@> leave[1 ref] vKP/REFC ->(end)
1     <0> enter ->2
2     <;> nextstate(main 1 -e:1) v:{ ->3
8     <2> sassign vKS/2 ->9
6        <1> complement[t4] sK ->7
5           <1> complement[t3] sK ->6
4              <1> rv2av[t2] sK/1 ->5
3                 <$> gv(*a) s ->4
7        <0> padsv[$x:1,2] sRM*/LVINTRO ->8
```

Example : inchworm.pm

Replacing OP_COMPLEMENT check function

```
static OP *(*my_old_ck_complement)(pTHX_ OP *) = 0;

static OP *my_ck_complement(pTHX_ OP *o) {
```

Example : inchworm.pm

Replacing OP_COMPLEMENT check function

```
static OP *(*my_old_ck_complement)(pTHX_ OP *) = 0;

static OP *my_ck_complement(pTHX_ OP *o) {
 OP *kid;

 o = CALL_FPTR(my_old_ck_complement)(aTHX_ o);
```

Example : inchworm.pm

Replacing OP_COMPLEMENT check function

```
static OP *(*my_old_ck_complement)(pTHX_ OP *) = 0;

static OP *my_ck_complement(pTHX_ OP *o) {
 OP *kid;

 o = CALL_FPTR(my_old_ck_complement)(aTHX_ o);

 kid = cUNOPo->op_first;
 if (kid->op_type == OP_COMPLEMENT && kid->op_flags & OPf_KIDS) {
```

Example : inchworm.pm

Replacing OP_COMPLEMENT check function

```
static OP *(*my_old_ck_complement)(pTHX_ OP *) = 0;

static OP *my_ck_complement(pTHX_ OP *o) {
 OP *kid;

 o = CALL_FPTR(my_old_ck_complement)(aTHX_ o);

 kid = cUNOPo->op_first;
 if (kid->op_type == OP_COMPLEMENT && kid->op_flags & OPf_KIDS) {
  op_null(o);
  o = cUNOPx(kid)->op_first;
  op_null(kid);
 }
```

Example : inchworm.pm

Replacing OP_COMPLEMENT check function

```
static OP *(*my_old_ck_complement)(pTHX_ OP *) = 0;

static OP *my_ck_complement(pTHX_ OP *o) {
 OP *kid;

 o = CALL_FPTR(my_old_ck_complement)(aTHX_ o);

 kid = cUNOPo->op_first;
 if (kid->op_type == OP_COMPLEMENT && kid->op_flags & OPf_KIDS) {
  op_null(o);
  o = cUNOPx(kid)->op_first;
  op_null(kid);
 }

 return o;
}
```

Example : inchworm.pm

Replacing OP_COMPLEMENT check function

```
# XS section
BOOT:
{
 my_old_ck_complement     = PL_check[OP_COMPLEMENT];
 PL_check[OP_COMPLEMENT] = MEMBER_TO_FPTR(my_ck_complement);
}
```

Example : inchworm.pm

Replacing OP_COMPLEMENT check function

```
# XS section
BOOT:
{
 my_old_ck_complement     = PL_check[OP_COMPLEMENT];
 PL_check[OP_COMPLEMENT] = MEMBER_TO_FPTR(my_ck_complement);
}
```

This gives :

```
$ perl5.10.0 -Mblib -MMy::XSModule -MO=Concise -e 'my $x = ~~@a'
```

Example : inchworm.pm

Replacing OP_COMPLEMENT check function

```
# XS section
BOOT:
{
 my_old_ck_complement     = PL_check[OP_COMPLEMENT];
 PL_check[OP_COMPLEMENT] = MEMBER_TO_FPTR(my_ck_complement);
}
```

This gives :

```
$ perl5.10.0 -Mblib -MMy::XSModule -MO=Concise -e 'my $x = ~~@a'
7  <@> leave[1 ref] vKP/REFC ->(end)
1     <0> enter ->2
2     <;> nextstate(main 1 -e:1) v:{ ->3
6     <2> sassign vKS/2 ->7
4        <1> rv2av[t2] sK/1 ->5
3           <$> gv(*a) s ->4
5        <0> padsv[$x:1,2] sRM*/LVINTRO ->6
-e syntax OK
```

Example : `inchworm.pm`

Action at a distance

Problem : this solution will act on all scopes.

Example : `inchworm.pm`

Action at a distance

Problem : this solution will act on all scopes.
Solution : rewriting it as a lexical pragma.

Example : inchworm.pm

Action at a distance

Problem : this solution will act on all scopes.
Solution : rewriting it as a lexical pragma.

```
$ cat lib/inchworm.ppm
sub import {
 $^H{+(__PACKAGE__)} = 1;
 $^H |= 0x020000;
 return;
}

sub unimport {
 $^H{+(__PACKAGE__)} = 0;
 return;
}
```

Example : inchworm.pm

Action at a distance

And now for inchworm.xs :

```
static UV get_hint(pTHX) {
 SV **val = hv_fetch(GvHV(PL_hintgv), "inchworm", 8, 0);
 return val ? SvUV(*val) : 0;
}
```

Example : inchworm.pm

Action at a distance

And now for inchworm.xs :

```
static UV get_hint(pTHX) {
 SV **val = hv_fetch(GvHV(PL_hintgv), "inchworm", 8, 0);
 return val ? SvUV(*val) : 0;
}

static OP *my_ck_complement(pTHX_ OP *o) {
 OP *kid;

 o = CALL_FPTR(my_old_ck_complement)(aTHX_ o);
 ...
```

Example : inchworm.pm

Action at a distance

And now for inchworm.xs :

```
static UV get_hint(pTHX) {
 SV **val = hv_fetch(GvHV(PL_hintgv), "inchworm", 8, 0);
 return val ? SvUV(*val) : 0;
}

static OP *my_ck_complement(pTHX_ OP *o) {
 OP *kid;

 o = CALL_FPTR(my_old_ck_complement)(aTHX_ o);
 if (!get_hint(aTHX))
  return o;
 ...
```

Pluggable perl

pp functions

Pluggable perl

pp functions

- `OP *(*pp)(pTHX)`

Pluggable perl

pp functions

- `OP *(*pp)(pTHX)`
- Stored into a member of the `OP` structure ;

Pluggable perl

pp functions

- `OP *(*pp)(pTHX)`
- Stored into a member of the `OP` structure ;
- The default values are stored into the `PL_ppaddr` array ;

Pluggable perl

pp functions

- `OP *(*pp)(pTHX)`
- Stored into a member of the `OP` structure ;
- The default values are stored into the `PL_ppaddr` array ;
- Called when the `OP` is executed.

Pluggable perl

pp functions

- OP *(*pp)(pTHX)
- Stored into a member of the OP structure ;
- The default values are stored into the PL_ppaddr array ;
- Called when the OP is executed.

```
$ cat run.c
int Perl_runops_standard(pTHX) {
 dVAR;
 while ((PL_op = CALL_FPTR(PL_op->op_ppaddr)(aTHX))) {
  PERL_ASYNC_CHECK();
 }
 TAINT_NOT;
 return 0;
}
```

Example : `length::undef`

Goal : Making `length` return `undef` when passed `undef` (this was added to `blead`).

Example : `length::undef`

Goal : Making `length` return `undef` when passed `undef` (this was added to `blead`).

```
static OP *(*my_old_ck_length)(pTHX_ OP *) = 0;

static OP *my_ck_length(pTHX_ OP *o) {
 o = CALL_FPTR(my_old_ck_length)(aTHX_ o);
 if (!get_hint(aTHX))
  return o;
```

Example : `length::undef`

Goal : Making `length` return `undef` when passed `undef` (this was added to `blead`).

```
static OP *(*my_old_ck_length)(pTHX_ OP *) = 0;

static OP *my_ck_length(pTHX_ OP *o) {
 o = CALL_FPTR(my_old_ck_length)(aTHX_ o);
 if (!get_hint(aTHX))
  return o;

 o->op_ppaddr = my_pp_length;

 return o;
}
```

Example : `length::undef`

Goal : Making `length` return `undef` when passed `undef` (this was added to `blead`).

```
static OP *(*my_old_ck_length)(pTHX_ OP *) = 0;

static OP *my_ck_length(pTHX_ OP *o) {
 o = CALL_FPTR(my_old_ck_length)(aTHX_ o);
 if (!get_hint(aTHX))
  return o;

 save_old_ppaddr(o);
 o->op_ppaddr = my_pp_length;

 return o;
}
```

Example : `length::undef`

The new `pp_length`

```
static OP *my_pp_length(pTHX) {
```

Example : length::undef

The new pp_length

```
static OP *my_pp_length(pTHX) {
 if (!SvOK(TOPs))
```

Example : length::undef

The new pp_length

```
static OP *my_pp_length(pTHX) {
 if (!SvOK(TOPs))
  RETURN;
```

Example : `length::undef`

The new `pp_length`

```
static OP *my_pp_length(pTHX) {
 if (!SvOK(TOPs))
  RETURN;

 return CALL_FPTR(fetch_old_ppaddr(PL_op))(aTHX);
}
```

Example : `length::undef`

`{save,fetch}_old_ppaddr`

Example : `length::undef`

`{save,fetch}_old_ppaddr`

- Usually implemented by borrowing the pointer table implementation of `sv.c` ;

Example : `length::undef`

`{save,fetch}_old_ppaddr`

- Usually implemented by borrowing the pointer table implementation of `sv.c` ;
- Makes more sense to use shared memory ;

Example : `length::undef`

`{save,fetch}_old_ppaddr`

- Usually implemented by borrowing the pointer table implementation of `sv.c` ;
- Makes more sense to use shared memory ;
- See also `B::Hooks::OP::Annotation` ;

Example : `length::undef`

`{save,fetch}_old_ppaddr`

- Usually implemented by borrowing the pointer table implementation of `sv.c` ;
- Makes more sense to use shared memory ;
- See also `B::Hooks::OP::Annotation` ;
- Or do it with `B::Hooks::OP::PPAddr` ;

Thanks!

Thank you for your attention!

Questions?