Creating CPAN Modules with SWIG

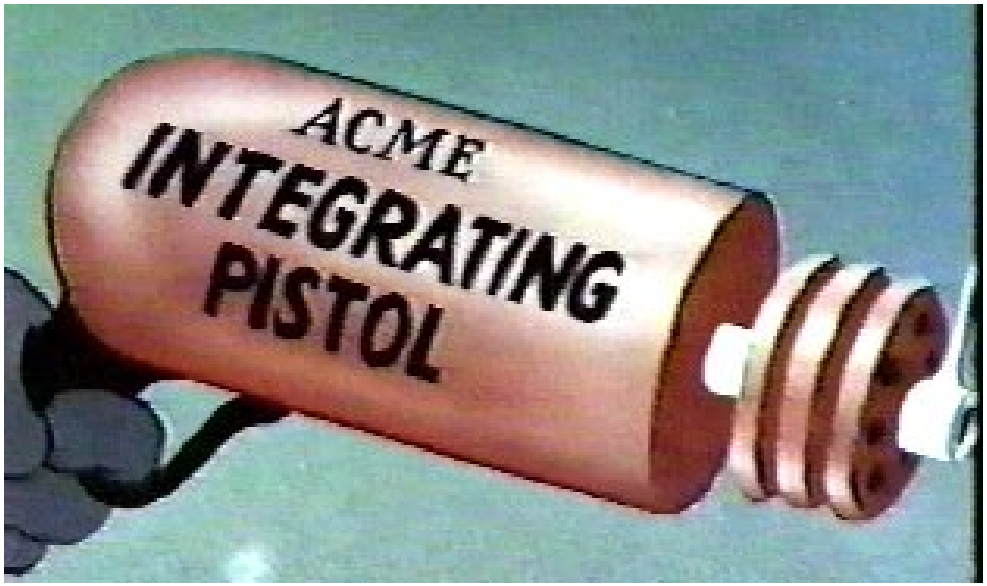
Jonathan Leto

```
%typemap(in) double const [] {
    AV *tempav;
    I32 len;
    int i;
    SV **tv;
    if (!SvROK($input))
        croak("Math::GSL : $input is
    if (SvTYPE(SvRV($input)) != SVt_PVAv
        croak("Math::GSL : $input is not  re ");

    tempav = (AV*)SvRV($input);
    len = av_len(tempav);
    $1 = (double *) malloc((len+1)*size
    for (i = 0; i <= len; i++) {
        tv = av_fetch(tempav, i, 0);
        $1[i] = (double) SvNV(*tv);
    }
}
```

Don't Write Glue

Use an Integrating Pistol



Need to integrate totally separate entities??
Integrate it instantly with the ACME Integrating Pistol!!

What is XS?

- eXternal Subroutine
- Large collection of macros which allow C/C++ method calls
- Extremely verbose
  - Math::GSL 0.10 has $\approx 274,000$ lines of XS
- perldoc perlxs

What is SWIG?

- Simplified Wrapper Interface Generator
- Creates a scripting language API to a C/C++ library
- 18 target languages supported
- Reads header files and transforms datatypes between languages
- Automatic argument type checking
- Concise
  - Math::GSL 0.10 has ≈ 500 lines of SWIG which generates $\approx 274,000$ lines of XS

What is Math::GSL?

- Perl Interface to the GNU Scientific Library (GSL), which is written in C
- Provides low-level access that emulates C calling style
- Also provides higher-level OO interface to some subsystems
- Broken into 48 subsystems like Vector, Matrix, RNG, Complex, Histogram, ...
- Extensive tests (3279 as of 0.10)

Module::Build and SWIG

- Module::Build does **not** support SWIG directly
- Subclassing to the rescue!

```
my $builder = $class->new(
    module_name         => 'Math::GSL',
    add_to_cleanup      => [ $cleanup ],
    create_makefile_pl  => 'passthrough',
    dist_abstract       => 'Interface to the GNU Scientific Library using SWIG',
    dist_author         => 'Jonathan Leto <jonathan@leto.net>',
    dist_version_from   => 'lib/Math/GSL.pm',
    include_dirs        => q{},
    extra_linker_flags  => '-shared -I./lib -I../lib ' . $ldflags,
    extra_compiler_flags=> q{-shared -fpic } . $ccflags,
    swig_flags          => $swig_flags,
    license             => 'gpl',
    requires => {
        'ExtUtils::PkgConfig'   =>  '1.03',
        'Scalar::Util'          =>       0,
        'Test::More'            =>       0,
        'Test::Exception'       =>    0.21,
        'Test::Class'           =>    0.12,
        version                 =>       0,
        perl                    =>  '5.8.8',
    },
    sign                    => 0,
    swig_source             => [
                                map { [ "$_.i" ] } @Subsystems
                            ],
);
$builder->add_build_element('swig');
$builder->create_build_script();
print "Have a great day!\n";
Build.PL
```

SWIG Example Code

::begin vim hack session::

```
%typemap(in) double const [] {
    AV *tempav;
    I32 len;
    int i;
    SV **tv;
    if (!SvROK($input))
        croak("Math::GSL : $input is not a reference!");
    if (SvTYPE(SvRV($input)) != SVt_PVAV)
        croak("Math::GSL : $input is not an array ref!");

    tempav = (AV*)SvRV($input);
    len = av_len(tempav);
    $1 = (double *) malloc((len+1)*sizeof(double));
    for (i = 0; i <= len; i++) {
        tv = av_fetch(tempav, i, 0);
        $1[i] = (double) SvNV(*tv);
    }
}

%apply double const [] { double *data, double *dest, double *f_in, double *f_out, double data[] };
%apply double const [] { double x[], double a[], double b[] };
%apply double const [] { const double * x, const double * y, const double * w };
%apply double const [] { const double x_array[], const double xrange[], const double yrange[]};
%apply double const [] { double * base, const double * base};
%apply double const [] { const double xrange[], const double yrange[] };
%apply double const [] { const double * array };
%apply double const [] { const double data2[], const double w[] };
```

::end vim hack session::

Active Development Continues

- Scientific Computing applications built on top of Math::GSL
- Full gsl_function callback support
- Static libraries and error handlers
- Porting to Darwin and Solaris
- Threads

Thanks

- Device::Cdio
- Thierry Moisan
- Eric Wilhelm
- #pdx.pm
- Leslie Hawthorn
- The entire Google Summer of Code crew

More Info

- http://www.swig.org
- http://leto.net/gitweb/
- http://leto.net/code/Math-GSL/
- http://groups.google.com/group/math-gsl-dev
- http://groups.google.com/group/perl-scientific-computing