Building a CPAN-Ready Perl Extension

James E Keenan
(jkeenan@cpan.org)
New Orleans Perlmongers
Friday, December 10, 2004

Preparation (1)

- Perl
- *h2xs, perldoc, cpan*
- *Test::Simple, Test::More*
  - verify presence: `perldoc -l Test::More`
- *ExtUtils::ModuleMaker*

Preparation (2)

- Automatic installation:
  - `sudo cpan install ExtUtils::ModuleMaker`
- Manual installation:
  - `perl Makefile.PL`
  - `make`
  - `make test`
  - `sudo make install`

Preparation (3)

- Installation alternatives:
  - *ppm*
  - *rpm*
  - *apget*
- On Windows: *nmake*

Preparation (4)

- Tarballs with code fragments:
  - *MM-0.02-code-fragments.tar.gz*
  - *MM-0.03-code-fragments.tar.gz*
- Place in your working directory
- For now, unzip only *MM-0.02*

h2xs

- Originally developed to pull C header files into Perl via 'XS' glue language
- But often used to prepare modules that don't include C code
  - `h2xs -AXn My::Module`

h2xs => 'ugly'

- Have to read *man* pages to decipher simplest mode of operation
- Writes an ugly Makefile.PL

```
• WriteMakefile(
    NAME              => 'My::Module',
    VERSION_FROM      => 'lib/My/Module.pm',
    PREREQ_PM         => {}, # e.g., Module::Name => 1.1
    ($] >= 5.005 ?    # Add these new keywords
                      # supported since 5.005
      (ABSTRACT_FROM  => 'lib/My/Module.pm',
                      # retrieve abstract from module
       AUTHOR         => 'James E Keenan <jimk@local>') :
()),
);
```

modulemaker

- Command-line utility included with Geoff Avery's *ExtUtils::ModuleMaker*
- `$ modulemaker`
- ... then just follow the prompts

modulemaker Summary (1)

- N: Module name
  - primary module
- A: Author information
  - author's name
  - CPAN ID
  - organization
  - website
  - author's e-mail address

modulemaker Summary (2)

- L: License
- D: Directives
  - C: Compact
    - 0 (default): `../My/Module/Makefile.PL`
    - 1: `../My-Module/Makefile.PL`
- N: New
  - 0: functional
  - 1 (default): object-oriented

modulemaker Summary (3)

- B: Build system
  - *ExtUtils::MakeMaker*
  - *Module::Build*
  - mixed
- G: Generate the module
- Q: Quit *modulemaker*

Structure of *My::Module*

- Changes
- lib/
  - My/
    - Module.pm
- Makefile.PL
- MANIFEST
- README
- scripts/
- t/
  - 001_load.t
- Todo

modulemaker's Makefile.PL

```
WriteMakefile(
    NAME           => 'My::Module',
    VERSION_FROM => 'lib/My/Module.pm', # finds $VERSION
    AUTHOR         => 'James E Keenan (jkeenan@cpan.org)',
    ABSTRACT       => '',
    PREREQ_PM      => {
                       'Test::Simple' => 0.44,
                      },
);
```

modulemaker's lib/My/Module.pm

[view in text editor]

modulemaker's t/001_load.t

```
use Test::More tests => 1;
BEGIN { use_ok( 'My::Module' ); }
```

If we had used *Text::Simple*

```
END {print "not ok 1\n" unless $loaded;}
use Test::Simple tests => 1;
$loaded = 1;
ok($loaded);
```

Is structure valid?

```
$ perl Makefile.PL
$ make
$ make test
$ sudo make install
$ make dist
$ make clean
```

Preparing to add real content

- To preserve what we've already done, we're going to create a new version: 0.02
- To save time and typing, I've written most of the new content you'll use tonight
- Unzip *MM-0.02-code-fragments.tar.gz*
- Copy-and-paste in as directed

Changes

For v0.02, edit *changes* to provide date and description of changes being made

```
$ vi Changes

  :r changes.0.02
```

lib/My/Module.pm

Update version number inside each `.pm` file

```
$ vi lib/My/Module.pm
:%s/0.01/0.02/gc
```

What will our module do?

- For simplicity, let's do what I did in *List::Compare*
- Creatively borrow code from the *Perl Cookbook*
- Put a nice, modular wrapper around it
- To start, we'll get the union of two lists
- So, we're ready to start coding, right?

Wrong!

Documentation first!

- Specification for functionality
- Will make code and tests easier to write

Documentation (I)

```
$ vi lib/My/Module.pm
  dd  # delete 1st blank line
  # move down to NAME section
  :r f/abstract.0.02  # add abstract
  # move down to SYNOPSIS
  :r f/synopsis.0.02
```

Documentation (2)

```
# move down to DESCRIPTION

  :r f/description.0.02
```

-- save file and make sure we haven't introduced any syntax errors

```
  :wq or :ZZ

$ perl -c lib/My/Module.pm
```

README

```
$ vi README
```

-- to save time, delete all and replace with code fragment

```
:1,$d
:0r f/README.0.02

:wq or :ZZ
```

So, *now* we're ready to write code, right?

Wrong!

Tests next!

- Test-driven development
- Will make code easier to write
- First, write the code that will test `get_union()`

t/001_load.t (1)

```
$ vi t/001_load.t

  dd  # delete 1st blank line
```

comment out original Test::More line because we don't yet know how many tests we'll eventually have

```
:r f/no_plan.0.02  # get new
Test::More line and BEGIN block
```

t/001_load.t (2)

```
:r f/setup_test_lists.0.02

:r f/get_union_tests.0.02
```

Note use of 'seen-hash' to test presence of elements in a list

```
  :wq or :ZZ

$ perl -c t/001_load.t

$ perl t/001_load.t
```

So, *now* we're ready to write code, right?

Yes!

At long last ... code!

```
$ vi lib/My/Module.pm
  :r BEGIN.0.02
```

-- no need for @EXPORT : no subs exported by default

-- separate line for each sub in @EXPORT_OK

-- establish one export tag (`all`) for future use

get_union()

```
:r f/get_union.0.02
:r f/return_true.0.02
```

-- reposition final '1' so module returns true

```
 :r f/BUGS.0.02
```

-- revise final sections of POD

-- miscellaneous cleanup

```
$ perl -c lib/My/Module.pm
```

make -- test -- install

```
$ perl Makefile.PL
$ make
$ make test
$ sudo make install
$ make dist
$ make clean
```

On to v0.03!

- To preserve what we've already done, we're going to create a new version: 0.03
- Unzip *MM-0.03-code-fragments.tar.gz*
- Copy-and-paste as needed

Building a CPAN-Ready Perl Extension:

Bonus Slides

James E Keenan
(jkeenan@cpan.org)
New Orleans Perlmongers
Friday, December 10, 2004

Test::Simple (1)

- Basic utilities for writing tests.

```
use Test::Simple tests => 1;
ok( $foo eq $bar, 'foo is bar' );
```

- Needs plan showing number of tests to be run.

Test::Simple (2)

- Exports only 1 function: `ok()`
  - `ok()` requires 1 argument -- the expression to be tested
  - `ok()` may take optional 2nd argument -- string describing purpose of test
  - `ok()` prints out either "ok" or "not ok" along with a test number

Test::Simple (3)

- `ok( get_temperature($hell) > 0, 'Hell not yet frozen over' );`
  - `ok 1 - Hell not yet frozen over`
- All tests are run in scalar context.
  - `ok( @stuff, 'Have some stuff' );`

    ... will fail if `@stuff` is empty

Test::More (1)

- Yet another framework for writing test scripts
- Use it
- `use Test::More tests => $Num_Tests;`
- **or**
  `use Test::More qw(no_plan);`

Test::More (2)

- or

```
use Test::More;
if( $^O eq 'MacOS' ) {
    plan skip all => 'Test
irrelevant on MacOS';
}
else {
    plan tests => 42;
}
```

Test::More (3)

- `BEGIN { use_ok( 'Some::Module' ); }`
- `ok($this eq $that, $test_name);`
- `is  ($this, $that,    $test_name);`
  `isnt($this, $that,    $test_name);`
- `like  ($this, qr/that/, $test_name);`
  `unlike($this, qr/that/, $test_name);`
- `cmp_ok($this, '==', $that, $test_name);`
- `can_ok($module, @methods);`
  `isa_ok($object, $class);`

Test::Harness (I)

- Run perl standard test scripts with statistics

```
use Test::Harness;
runtests(@test_files);
```

- Used by `make test` to run either one test file (*test.t*) or a subdirectory of test files (*t/*.t*).
- Used by `prove` to run a single test file

Test::Harness (2)

- Each individual file relies on *Test::Simple*, *Test::More* or another *Test::** module.
- Outputs statistics once all tests are run
- As long as you are using `make test` or `prove` you don't need to know the workings of *Test::Harness* -- or even call it

Test::Builder

- Backend for building test libraries
- This is what's inside both *Test::Simple* and *Test::More*
- Provides building blocks used to write other, more specialized test modules
- Unless you want to do that, you don't need to concern yourself with *Test::Builder*

prove

- Command-line utility now included with *Test::Harness*
- Hence, included in Perl 5.8

Test::Tutorial (1)

- This is probably the *first* thing you should read after attending this talk.
- `perldoc Test::Tutorial`

Building a CPAN-Ready Perl Extension

James E Keenan
(jkeenan@cpan.org)
New Orleans Perlmongers
Friday, December 10, 2004