Inline
or
Pathologically Polluting Perl

Andy Adler

Nonsense in the intellect promotes corruption in the will
-C.S.Lewis
Outline

• Ways to link Perl to Other Stuff
• History of Inline
• Using Inline::C, Inline::Java
• Writing your own Inline::
Why Inline?

• Isn’t Perl Perfect?

• No. “Perl” != “Perfect”

• However, “Perl” =~ /Per([fect]*)/ which is more than we can say for C, Java, Python, etc..
Linking Perl to Stuff: XS

Good:
• Powerful

Bad:
• Requires learning a new language
• Requires knowing about Perlguts, even for simple stuff
• need to create many accessory files
Linking Perl to Stuff: **SWIG**

**Good:**
- Automates much of the build process

**Bad:**
- Requires learning a new language
- Not part of Perl distribution
- Versioning issues (May be solved now)
- Creates extra files
Philosophical Aside

**Assertion:** Creating lots of files is bad

- Many languages (notably Java) force you to create files
- However, the *raison d’être* of files is to organize information for the user. Any programming language with interferes with this is evil, evil, evil

**Now,** back to your regularly scheduled talk.
Linking Perl to Stuff: **Inline**

**Good:**
- Very DWIM (Takes care of details for you)
- Almost no unnecessary syntax
- Easy to learn
- Can write *One liners* with Inline

**Bad:**
- Not as powerful as XS
- Can’t distribute modules without XS (to be removed in ver 0.50)
Using Inline::C

CODE:

use Inline C => <<'END_C';
void greet(char *greetee) {
    printf("Hello, %s\n", greetee);
}
END_C

greet("world");

OUTPUT:

Hello, world
Using Inline::C

CODE:
use Inline C;
print JAxH('Perl');
__END__
__C__
SV* JAxH(char* x) {
    return newSVpvf(
        "Just Another %s Hacker\n", x);
}
OUTPUT:
Just Another Perl Hacker
Inline Use: (Win2K ActivePerl)

```bash
$ TIMEFORMAT="Time= %R"
$ time C:/perl/bin/perl ex2.pl
Just Another Perl Hacker
Time= 6.743
$ time C:/perl/bin/perl ex2.pl
Just Another Perl Hacker
Time= 0.239
```
Inline Directories

$ ls -lR .
drwxr-xr-x 0 Nov 4 20:42 _Inline
-rw-r--r-- 135 Nov 4 20:39 ex2.pl
./_Inline:
drwxr-xr-x 0 Nov 4 20:42 build
-rw-r--r-- 221 Nov 4 20:42 config
drwxr-xr-x 0 Nov 4 20:42 lib
./_Inline/build:
./_Inline/lib:
drwxr-xr-x 0 Nov 4 20:42 auto
./_Inline/lib/auto:
drwxr-xr-x 0 Nov 4 20:42 ex2_pl_1031
./_Inline/lib/auto/ex2_pl_1031:
-r--r--r-- 0 Nov 4 20:42 ex2_pl_1031.bs
-r-xr-xr-x 20480 Nov 4 20:42 ex2_pl_1031.dll
-r--r--r-- 832 Nov 4 20:42 ex2_pl_1031.exp
-rw-r--r-- 594 Nov 4 20:42 ex2_pl_1031.inl
-r--r--r-- 2234 Nov 4 20:42 ex2_pl_1031.lib
Warning

- The next slide contains windows specific code.
- Viewer discretion is advised
use Inline C => DATA => LIBS => '-luser32', PREFIX => 'my_';
MessageBoxA('Inline Message Box', 'Just Another Perl Hacker');
__END__
__C__
#include <windows.h>
int my_MessageBoxA(char* C, char* T){
    return MessageBoxA(0, T, C, 0); }
External Libraries
See Perl Run. Run Perl, Run!

Inline::CPR -> Create C interpreter

#!/usr/bin/cpr
int main(void) {
    printf("Hello, world\n");
}
Inline ILSMs

**ILSM** = Inline Language Support Module

- Inline::CPP
- Acme::Inline::PERL
- Inline::Java
- Inline::Guile
- Inline::C
- Inline::Befunge
- Inline::BC
- Inline::TT
- Inline::WebChat
- Inline::Ruby
- Inline::Tcl
- Inline::Python
- Inline::Pdlpp
- Inline::Octave
- Inline::Basic
- Inline::Filters
- Inline::Awk
- Inline::ASM
- Inline::Struct
Creating an Inline Module

Techniques to link to Perl

– Compile to a dynamic library (*.so, *.dll) and link to Perl at run time (::C, ::CPP, ::Java::API)
– Open a socket connection between Perl and the other interpreter (::Python, ::Java)
– Pipe stdio, stderr between Perl and other interpreter (::Octave). (using IPC::Open3)
How to create Inline Module

Look at Inline::PERL

Inline::PERL gives you the power of the PERL programming language from within your Perl programs. …

PERL is a programming language for writing CGI applications. It's main strength is that it doesn't have any unnecessary warnings or strictures.
Create Inline Module

• Create the following methods
  – Register
  – Build
  – Load
  – Validate

• Object variables contains all the code and administrative information
Example of a “build” method

sub load {
    my $o = shift;
    my $obj = $o->{API}{location};
    open PERL_OBJ, "< $obj" or croak "Can't open $obj for output\n!";
    my $code = join ' ', <PERL_OBJ>;
    close /*PERL_OBJ;
    eval "package $o->{API}{pkg};\n    $code";
    croak "$obj:\n$@" if $@;
}

Current Status of Inline

• Stayed at Version 0.43 for a long time.
• Version 0.44 has just been released
  – New, cleaner build
  – Bug fixes
• Version 0.50 promises:
  – Distribute modules without Inline
  – Cleaner features