C Perl, C Perl Inline, C Perl XS, C Perl Guts

Abram Hindle

Victoria Perl Mongers (victoria.pm.org)

abez@abez.ca

March 16, 2004

This Presentation

- What am I going to Cover?
  - Inline::C
  - Perl Data Structures / Guts
  - XS

This Presentation

- What am I not going to Cover?
  - How to program in C
  - C++
  - Anything in depth
  - Magic Perl Guts
  - OO and Perl Guts
  - References and Perl Guts
  - SWIG (maybe you should take a look if you’re wrapping a lib)

Inline::C

- What is it?
  - Write Perl Subroutines in C [1]
  - Compile and Link C into Perl
  - Embed subroutines into perl scripts and modules
  - Automatically Wrap functions based on prototypes.

Inline::C

- How do you use Inline::C?
  - `use Inline C => CODE;`
  - `bind Inline C => CODE;`
    - Dynamic generation of C code.
  -
```
use Inline C;
__END__
__C__
void cCode() {
}
```

Inline::C

- How do you use Inline::C?
  - Inline::C parses the function prototypes and wraps them into perl.
  - Function prototypes should be in ANSI style
    - return-type function-name (type-name-pairs) …

Inline::C

- Which types are supported?
  - void → nothing
  - double → scalar
  - char * → string
  - SV* → Scalar
  - long → Scalar
  - float is not supported but you can support it

Inline::C

- Example

```
use Inline C;
helloWorld();
__END__
__C__
void helloWorld() {
  printf("Hello world\n");
}
```

Inline::C

- Example

```
use Inline C;
hello('World');
hello(42);
hello(42,42);
hello(undef);
__END__
__C__
void hello(char * string) {
  printf("Hello %s\n",string);
}
```

Inline::C

- Example (perl guts)

```
use Inline C;
$a = "HEY";
hello('World',1,42,undef,'Universe',$a);
__END__
__C__
void hello(SV* name1, ...) {
        Inline_Stack_Vars;
        int i = 0;
        STRLEN count;
        for (i = 0; i < Inline_Stack_Items; i++) {
                char * string = SvPV(Inline_Stack_Item(i), count);
                printf("Hello␣%s\n",string);
        }
}
```

Inline::C

- Inline Stack Access macros
  - `Inline_Stack_Vars` - put this macros at the top of the subroutine as it defines new various for Inline_Stack macros to work on
  - `Inline_Stack_Items` - how many items are on the stack
  - `Inline_Stack_Item(i)` - get the item at index i in the stack
  - see perldoc Inline::C

Perl Guts

- Types in C [5]
  - SV - Scalar Value
  - AV - Array Value
  - HV - Hash Value
  - IV - Integer Value *
  - UV - Unsigned Value *
  - NV - Double Value *
  - PV - String Value *
  - * return values

Perl Guts

- Convert Scalars [5] (these are macros)
  - `int SvIV(SV*)`
  - `unsigned int SvUV(SV*)`
  - `double SvNV(SV*)`
  - `char * SvPV(SV*, STRLEN len)`
    - puts length in len, i is the buffer in SV*
  - `char * SvPV_nolen(SV*)`

Perl Guts

- Set scalars [5] (these are macros)
  - `void sv_setiv(SV*, IV);`
  - `void sv_setuv(SV*, UV);`
  - `void sv_setnv(SV*, double);`
  - `void sv_setpv(SV*, const char*);`
  - `void sv_setpvn(SV*, const char*, int)`

Perl Guts

- Create scalars [5] (these are macros)
  - `SV* newSViv(IV);`
  - `SV* newSVuv(UV);`
  - `SV* newSVnv(double);`
  - `SV* newSVpv(const char*, int);`
  - `SV* newSVpvn(const char*, int);`
  - `SV* newSVpvf(const char*, ...);`
  - `SV* newSVsv(SV*);`

Perl Guts

- Arrays [5]
  - Make an array
    - create - `AV* newAV();`
    - create - `AV* av_make(I32 num, SV **ptr);`
  - Operations
    - push - `void av_push(AV*, SV*);`
    - pop - `SV* av_pop(AV*);`
    - shift - `SV* av_shift(AV*);`
    - unshift - `void av_unshift(AV*, I32 num);`

Perl Guts

- Arrays [5]
  - length - `I32 av_len(AV*);`
  - fetch - `SV** av_fetch(AV*, I32 key, I32 lval);`
  - store - `SV** av_store(AV*, I32 key, SV* val);`
  - clear - `void av_clear(AV*);`
  - destroy - `void av_undef(AV*);`
  - expand - `void av_extend(AV*, I32 key);`

Perl Guts

- Hashes
  - `HV* newHV();`
  - `SV** hv_store(HV*, const char* key, U32 klen, SV* val, U32 hash);`
  - `SV** hv_fetch(HV*, const char* key, U32 klen, I32 lval);`
  - `bool hv_exists(HV*, const char* key, U32 klen);`
  - `SV* hv_delete(HV*, const char* key, U32 klen, I32 flags);`
  - `void hv_clear(HV*);`
  - `void hv_undef(HV*);`

Perl Guts

- Hash iteration
  - `I32 hv_iterinit(HV*);`
  - `HE* hv_iternext(HV*);`
  - `char* hv_iterkey(HE* entry, I32* retlen);`
  - `SV* hv_iterval(HV*, HE* entry);`
  - `SV* hv_iternextsv(HV*, char** key, I32* retlen);`

Perl Guts

- Package Vars [5]
  - `SV* get_sv("package::varname", TRUE);`
  - `AV* get_av("package::varname", TRUE);`
  - `HV* get_hv("package::varname", TRUE);`

Perl Guts

- Mortality
  - If you have temporary values such as those that are returned or used temporarily you want the garbage collection to handle it properley.
  - `SV* sv_newmortal()`
  - `SV* sv_2mortal(SV* sv)`

XS

- What is XS?
  - interface description format
  - Integrate Perl and C code / libs
  - A language that is used to integrate C w/ Perl
  - Wraps C calls

XS

- Into the fire: [4] - Some of this is ripped from the perlxstut
  - h2xs -A -n PackageName
  - Edit PackageName.xs
  - perl Makefile.PL
  - make
  - Make a test script in the PackageName directory
  - make install – only do this if your test script has problems finding PackageName

XS

- The .xs file
  - Prototypes are done differently, see perldoc perlxs and perlxstut
  - First line, return type
  - Second line, prototype in K&R
  - Following lines, type the parameters (newline is the delimiter)
  - ```
    int
    strcmp(str1,str2)
            char *str1
            char *str2
    ```
    ```

XS

- Lets wrap a already existing function (add this to your .xs)
  - strcmp
  - ```
    #include <string.h>
    int
    strcmp(str1,str2)
            char *str1
            char *str2
    ```
  - That's it
    ```

XS

- Lets write a new function (add this to your .xs)
  - hello
  - ```
    void
    hello()
    CODE:
    printf("Hello World\n");
    ```
  - That's it
    ```

XS

- Lets see our test driver (after make install)

```
#!/usr/bin/perl
use Hello;

Hello::hello();

print join("␣",
                Hello::strcmp("a","b"),
                Hello::strcmp("a","a"),
                Hello::strcmp("bbbb","aaaa")
        ),$/;
```

Get Help

- Where to get Help?
  - perldoc Inline
  - perldoc Inline::C
  - perldoc Inline::C-Cookbook
  - perldoc perlguts
  - perldoc perlapi
  - perldoc perlxs
  - perldoc perlxstut
  - perldoc perlembed

References

[1] INGERSON, B. Inline::c - write perl subroutines in c.

[2] INGERSON, B., AND WATKISS, N. Inline - write perl subroutines in other programming languages.

[3] MACEACHERN, D., AND ORWANT, J. perlembed - how to embed perl in your c program.

[4] OKAMOTO, J. perlxstut - tutorial for writing xsubs.

[5] PORTERS, P. . perlguts - introduction to the perl api.

[6] ROEHRICH, D., OKAMOTO, J., AND STUHL, B. perlapi - autogenerated documentation for the perl public api.

[7] ROEHRICH, D., AND PORTERS, T. P. perlxs - xs language reference manual.