Perl XS: Introductory Workshop

Rishi Nag

http://www.ebi.ac.uk/~rishi/2015_ensembl_retreat/

This workshop uses Perl, standard build tools, a terminal and an editor. It has notes and code sections.

```
A Code Section
```

Basics

Create a Basic XS Framework

Create the Framework

```
h2xs -A -n MyXS
```

The important files created in MyXS are the following. Have a look at these.

- Makefile.PL – libraries and headers, C compilation options
- MyXS.xs – where your XS code lives

Test the build process:

```
cd MyXS
perl Makefile.PL
make
```

For more info http://perldoc.perl.org/h2xs.html

Make a Hello World

Add a hello world function to the XS script and a perl function to call it.

MyXS.xs (whole file) will look like:

```
#include "EXTERN.h"
#include "perl.h"
#include "XSUB.h"

#include "ppport.h"

#include <stdio.h>
void print_hello()
{
  printf( "Hello From XS!\n" ) ;
}

MODULE = MyXS      PACKAGE = MyXS

void
print_hello()
```

Create a Perl file in the same directory, calling_xs.pl

```
use MyXS ;   #this will be skipped in subsequent examples

MyXS::print_hello() ;
```

Run the build process and run the file. The first line is only really needed if new files or build options are changed – but kept here as a reminder.

```
perl Makefile.PL
make
perl -Mblib calling_xs.pl
```

This, and the first following exercises use the same function name in C and XS to allow the code to be called correctly.

Passing Basic Parameters

Let's make some functions that will allow us to print more. Remember to add the XS and C parts to the correct section of the file.

The C function comes first.

```
void print_string(char* s)
{
  printf("InXS:%s\n",s) ;
}
```

The XS function now takes a parameter and type definition, with one type definition per line.

```
void
print_string(s)
  char* s
```

Edit your Perl file

```
MyXS::print_string("Need You Tonight") ;
```

Re-run the build process.

Task

Write the XS function for the print_strings function below and perl code to use it.

```
void print_strings(char* s1, char* s2)
{
  printf("InXS 1:%s\n", s1) ;
  printf("InXS 2:%s\n", s2) ;
}
```

Returning Basic Arguments

Add a C function to the XS file:

```
int treble(int x)
{
  x*=3 ;
  return x ;
}
```

Note that the XS function now has a return type associcated with it.

```
int
treble( x )
  int x
```

And the following calls the code and returns the value.

```
$a = 345 ;
print("3x$a=", &MyXS::treble($a),"\n") ;
```

Tasks

Try calling the function with a string parameter. What happens? Try with a float parameter.

Try making a multiply function in C and its XS interface.

XS Code Sections

The implementation does not have to be in the C section of an XS file. We now introduce two sections to the XS function – CODE and OUTPUT.

```
int
quadruple( x )
  int x
CODE:
  RETVAL = 4*x ;
OUTPUT:
  RETVAL
```

If no function is present a C function prototype will be required in the C section of the XS file.

```
int quadruple( int x ) ;
```

Make the library, edit the Perl module to call this function and run.

Task

Try putting a call to print_string() from the second example in the CODE section of the quadruple XS code.

Return multiple parameters

One stylistic habit Perl uses is the ability to return multiple. XS allows this to happen using the OUTLIST keyword and passing in the values. Add the XS prototype, also using NO_INIT to indicate the following variables don't need to be initialized.

```
void
tripquad_outlist( x, OUTLIST times3, OUTLIST times4 )
        int x
        int times3 = NO_INIT
        int times4 = NO_INIT
```

Add the C function to the XS file. Note the OUTLIST keyword means the values get added as pointers to that datatype.

```
void tripquad_outlist( int x, int* times3, int* times4 )
{
  *times3 = x*3 ;
  *times4 = x*4 ;
}
```

Add the calling code to the Perl function.

```
$a = 345 ;
($t3, $t4) = &MyXS::tripquad_outlist($a) ;
print("3x$a=$t3\n") ;
print("4x$a=$t4\n") ;
```

Run the build and the Perl file.

Task

What happens when you specify

1. a single scalar variable
2. an array

as the value recieving the return result from the function.

Calling A Non-Standard C Library

We have seen how to use a function from a C library – the print functions created earlier all used printf from stdio.h. In this final section of the workshop we will call a non-standard library and make a call to it from. The XS file needs to be changed in the same way – it will need to include the header file and make the function call.

Make sure you are in the MyXS directory.

Prepare the C library we are going to use for this project. This is a basic C library which will print s statement out with a single function.

```
git clone https://github.com/rishidev/xs-workshop1-c_lib.git
cd xs-workshop1-c_lib
gcc -fPIC -c simple.c -o simple.o
ar rcs libsimple.a simple.o
```

As this will be a non-standard C library alter the Makefile.PL file to indicate the location of the header file and the library needed for the XS module to build successfully. Alter the INC and LIB lines in Makefile.PL

```
INC                 => '-I. -I./xs-workshop1-c_lib',
LIBS                => ['-L./xs-workshop1-c_lib -lsimple'],
```

Add a new print_string function in the MyXS.xs file, and include simple.h

```
#include "simple.h"

void print_string2(const char* s)
{
  printf("InXS:%s\n",s) ;
  print_clib(s) ;
}
```

Add the print_string2 XS interface – this will be the same as for print_string bar the name.

Ensure a call to print_string2() is present in calling_xs.pl.

Move back to the MyXS directory (if you haven't already), compile and run. As we have edited Makefile.PL we will also need to regenerate the makefile.

```
cd ..
perl Makefile.PL
make
perl -Mblib calling_xs.pl
```

Ensure both the XS and C print output is present.