Convert::Binary::C module

C headers parser and
"Binary Data Conversion using C Types"

Yossi Itzkovich
Feb 2013

General Description

- A lot of convenience methods to retrieve information about the C types that have been parse.
- A C Preprocessor
- A Parser for C type definitions
- Using pack() and unpack() with C types instead of a string representation of the data structure for conversion of binary data from and to Perl's complex data structures

pack() function example

- *struct foo {*
  char ary[3];
  unsigned short baz;
  int bar;
};
- Perl code:

```
my @ary = (1, 2, 3);
my $baz = 40000;
my $bar = -4711;
my $binary = pack ('c3 S i', @ary, $baz, $bar);
```

- What are the disadvantages with the above coding ?

pack() function example (2)

- The disadvantages are:
  - Maintaining 2 sources (C, Perl)
  - Not practical for complicated C structures
  - Alignment issues

My typical usage of Convert::Binary::C

```
use Convert::Binary::C
my $c = new Convert::Binary::C
(optional_configuration)
$c->parse_file(file_name)
```

The pack() method example

```
my $data = {
        ary => [1, 2, 3],
       baz => 40000,
       bar => -4711,
};
 $binary = $c->pack('foo', $data);
```

The unpack() method

```
my $binary = get_data_from_memory();
my $data = $c->unpack('foo', $binary);
say "foo.ary[1] = $data->{ary}[1]";
```

Or use Data::Dumper:

```
say Dumper($data);
```

Automatic configuration using ccconfig

- As there are over 20 different configuration options, setting all of them correctly can be a lengthy and tedious task.
- The ccconfig script, which is bundled with this module, aims at automatically determining the correct compiler configuration by testing the compiler executable. It works for both, native and cross compilers.

Methods

- Important methods:
  - sizeof
  - member, offsetof
  - typeof
  - def , defined
  - enum_names, enum
  - struct_names, struct
  - union_names, union
  - typedf_names, typedef
  - macro_names, macro
  - compound_names, compound

When we use this module

- In ECI Telecom , in the EMS-XDM project we use it:
  - Parse embedded header files that define the protocol.
  - Endian conversion
  - Backward compatibility

THANK YOU

Configuration options

- 'Define' => ['DEBUGGING', 'FOO=123'],
- 'StdCVersion' => 199901,
- 'ByteOrder' => 'LittleEndian',
- 'LongSize' => 4, 'IntSize' => 4, 'DoubleSize' => 8, 'CharSize' => 1,
- 'PointerSize' => 4, 'EnumSize' => 4, 'ShortSize' => 2, , 'LongLongSize' => 8, 'LongDoubleSize' => 12, 'FloatSize' => 4,
- 'HostedC' => 1,
- 'HasMacroVAARGS' => 1,
- 'Assert' => [],
- 'UnsignedChars' => 0,
- , 'EnumType' => 'Integer‘, DisabledKeywords' => [], 'Alignment' => 1, 'KeywordMap' => {}, 'Include' => ['/usr/include'], 'HasCPPComments' => 1, 'Bitfields' => { 'Engine' => 'Generic' }, 'UnsignedBitfields' => 0, 'Warnings' => 0, 'CompoundAlignment' => 1, 'OrderMembers' => 0