Perl coding standard for FCM

Last updated: 28 November 2006

Met Office
FitzRoy Road, Exeter
Devon, EX1 3PB
United Kingdom

Contents

- Main contents:
  1. Introduction
  2. Online Perl style guides
  3. Coding standard

1. Introduction

Perl (Practical Extraction and Report Language) is gaining a lot of popularity at the Met Office for developing applications that are traditionally programmed using shell script. For example, it is currently our choice of language for developing the in-house components of the FCM system.

Perl is a very powerful general purpose scripting tool that is free and portable across a huge variety of platforms including many non-Unix systems. Replacing a shell script with an equivalent Perl program often results in a massive reduction in runtime - using cleaner syntax and algorithm.

Perl is a language with a rich set of "grammar". To most people, the first impression of a piece of code written in Perl is that it is very ugly. Its lines are full of punctuation junks that nobody can hope to understand. This is often caused by poorly written programs coupled with little and often inadequte documentations. To improve readability and to reduce the overheads of maintenance, it is important for Perl programmers to write their code in a nice and consistent way.

The aim of this document is to propose ideas on how the Perl programming language should be used with FCM.

2. Online Perl style guides

There are many Perl style guides available online. Some are listed below:

- the manpages `perlstyle` and `perlmodstyle`
- Perlingo
- Perl Style Guide for Large Scale Object Oriented Development
- Perltidy

- mod_perl Coding Style Guide (PDF)

It is worth noting that the ideas in some of the Perl style guides may conflict with each other - as they are targeted to different people. However, there is a common theme to most of the best practices. The following is a summary:

- A source file, whether it is a top-level script or a module, should contain a header block with information of the source file, some general description of the code as well as standard pragmas or options for running the Perl interpreter.
- Each function should be preceded by a comment block or a block of POD to specify a synopsis for the defined function.
- Avoid using names of built-in functions to name your new functions.
- If the parameter list of a function is becoming too long, e.g. more than 3 arguments, you may want to implement the argument list with a hash, so that all the arguments are named.
- Indent to an appropriate number of spaces/tabs for code blocks.
- Open curly brackets should be on the same line as the keyword.
- Close curly brackets should line up with the keywords that started the block.
- Insert a blank line between chunks of code that do different things.
- Use space between tokens and operators to improve readability. Use space after commas. No space before commas and semi-colons.
- Line up corresponding items vertically.
- Use parentheses only if necessary.
- Restrict line length to e.g. 80 characters.
- Be consistent with style.
- Use the "strict" and "warnings" pragmas.
- options and arguments should be processed as "soon" as possible, preferably at the beginning of a source file or a function.
- Use the "__END__" directive at the end of a Perl script.
- -more later-

3. Coding standard

The main question is: what should we include in our coding standard? Although we would like to recommend using the best practices described in the last section, we would not want to impose any restriction, as Perl is a language designed to do things in many different ways. The only thing we would like to impose is a header block in each source file, and a header comment block for each function in a source file.

For a Perl executable, the header block of the source file should contain the following:

- the first line should be `"#!/usr/bin/perl"`
- the name of the executable
- a synopsis and/or a description or what the executable does
- an explanation of the options and arguments (either in separate sections or described within the "description" section)
- copyright and owner information
- no history or revision information, (as we would prefer using the Subversion log message to record such information)
- use "strict" and use "warnings" statements
- use the "__END__" directive at the end of a Perl script.

Executable header block example

```
#!/usr/bin/perl
# ------------------------------------------------------------------------------
# NAME
#   example.pl
#
# SYNOPSIS
#   example.pl [options] args
#
# DESCRIPTION
#   This is a header example, and so on and so forth.
#
# OPTIONS
#   -b [--bar] - this option does what "bar" does.
#   -f [--foo] - this option does what "foo" does.
#
# ARGUMENTS
#   args       - description of each argument.
#
# SEE ALSO
#   list of relevant commands, modules and documents
#
# COPYRIGHT
#   (C) Crown copyright Met Office. All rights reserved.
#   For further details please refer to the file COPYRIGHT.txt
#   which you should have received as part of this distribution.
# ------------------------------------------------------------------------------

# Standard pragmas
use strict;
use warnings;

# Standard modules
use Getopt::Long;

# ------------------------------------------------------------------------------

# Process options
my ($foo, $bar);
GetOptions (
  'foo|f' => \$foo,
  'bar|b' => \$bar,
);

# Process arguments
my $arg = shift @ARGV;

# Do something...
print 'This is an example: ', $arg, "\n";
print 'FOO', "\n" if $foo;
print 'BAR', "\n" if $bar;

__END__
```

For a Perl module, the header block of the source file should contain the following:

- the first line should be "`#!/usr/bin/perl`"
- the name of the module
- a synopsis and/or a description or what the module does
- copyright and owner information
- no history or revision information
- use "strict" and use "warnings" statements
- a list of exports (do not export if you are writing an OO module)

- use the "__END__" directive at the end of a Perl script, and remember to put 1 before the end of the module, so that it will return true.

Module header block example

```
#!/usr/bin/perl
# ------------------------------------------------------------------------------
# NAME
#   Metoffice::Example
#
# DESCRIPTION
#   This is a header example, and so on and so forth.
#
# SEE ALSO
#   list of relevant commands, modules and documents
#
# COPYRIGHT
#   (C) Crown copyright Met Office. All rights reserved.
#   For further details please refer to the file COPYRIGHT.txt
#   which you should have received as part of this distribution.
# ------------------------------------------------------------------------------

package Metoffice::Example;

# Standard pragmas
use strict;
use warnings;

# Exports
our (@ISA, @EXPORT, @EXPORT_OK);
require Exporter;
@ISA    = qw(Exporter);
@EXPORT = qw(
  foo
  bar
);

# ------------------------------------------------------------------------------
# ... some more Perl ...
# ------------------------------------------------------------------------------

1;

__END__
```

The header of a function (or "method" for OO code) should have the following:

- a synopsis of the function's calling interfaces.
- a description of what the function does and what it returns, and if appropriate, what each argument represents

Function header block example

```
# ... Something before the function ...
#
# ------------------------------------------------------------------------------
# SYNOPSIS
#   $result = &print_hello;
#   $result = &print_hello ($arg);
#
# DESCRIPTION
#   Print the word "hello" to standard output. If no argument is specified,
#   the word "world" will be printed after the word "hello". Otherwise, the
#   word specified by the argument $arg will follow "hello". The function
#   returns true on success.
#
# ARGUMENTS
#   $arg - optional, describe $arg if it has not been done in the above section
# ------------------------------------------------------------------------------

sub print_hello {
  my ($arg) = @_;
  $arg = defined ($arg) ? $arg : 'world';
  my $result = print 'hello ', $arg, "\n";
  return $result;
}

# ------------------------------------------------------------------------------
#
# ... Something after the function ...
```