CodingStyle

Guidelines for the zealots 🙂

The CodingStyle used in an application can lead to various heated debates. To avoid those in future, the SpamAssassin team tried to come to a compromise. Not everybody likes every single detail but sometimes one has to bite the bullet to avoid long discussions 🙂

Perl CodingStyle

Mostly quite sensible Perl stuff as documented in perlstyle:

http://perldoc.perl.org/perlstyle.html

Differences from the standard Perl style are documented below.

Formatting

The main difference between our style and the standard Perl style is that we use an indentation level of 2 and try to keep everything under 80 columns (79 not counting the newline).

Braces and Parentheses

elsif and else are uncuddled, no spaces on the inside of paretheses, and try to avoid one-liner conditionals that require braces. If braces are needed, use multiple lines. Open braces should appear on the same line of the condition, unless it's a multiple line condition. For example:

```
if ($foo) {
  do_foo();
}
elsif ($bar_really_long_condition_that_lasts_longer_than_40_columns &&
       $foo_really_long_condition_that_lasts_longer_than_40_columns)
{
  do_bar();
}
else {
  do_else();
}
```

- *(status of this guideline: proposed by DanielQuinlan. everyone happy with it?)*
- *(+1 – JustinMason)*
- *(+1 – DuncanFindlay who has finally weened himself off cuddled elses)*

One liners

";" should always be at the end of a line unless its a "for", no points for squeezing more code on the same line.

- *(status of this guideline: proposed by DanielQuinlan. everyone happy with it?)*
- *(+1 – JustinMason)*
- *(+1 – DuncanFindlay)*

Function Arguments

Use @_ rather than shift if possible. For example:

```
sub foo {
  my ($foo, $bar) = @_;
```

instead of

```
sub foo {
  my $foo = shift;
  my $bar = shift;
```

- *(status of this guideline: proposed by DanielQuinlan. everyone happy with it?)*
- *(+1 – JustinMason)*
- *(+1 – DuncanFindlay - shift should be fine for one argument subroutines, IMO)*

Accessors

We don't use perl-style accessors very frequently (ie.

```
sub foo {
  my ($self, $val) = @_;
  if (defined $val) { $self->{foo} = $val; } else { return $val; }
}
```

Instead, the more wordy Java/C++ style is preferred:

```
sub get_foo { my ($self) = @_; return $val; }
sub set_foo { my ($self, $val) = @_; $self->{foo} = $val; }
```

The reason why is detailed at PerlAccessorsConsideredHarmful.

- *(status of this guideline: proposed by JustinMason. everyone happy with it?)*
- *(+1 – Daniel Quinlan except where we already do this)*

Return Values From Functions

Returns should be explicit, instead of implicit, for clarity:

```
sub foo { ...stuff...; $val = 1; return $val; }
```

instead of

```
sub foo { ...stuff...; $val = 1; }
```

reason: in the latter, `$val` is returned, but it's not explicit and not obvious. A later change could result in code being added after the assignment to `$val`, since it's not clear that the value is returned by the function.

- *(status of this guideline: proposed by JustinMason. everyone happy with it?)*
- *(+1 – Daniel Quinlan)*
- *(+1 – DuncanFindlay)*

C CodingStyle

In our C code we took the easiest way and adopted the Apache Developers' C Language Style Guide.

One addition, arising from bug 4593: if there are warnings about variables being signed/unsigned, caused by use of "int" types in system calls that accept "size_t" (or similar), it is better to *carefully* perform a cast in the call(s) to the specific system calls being warned about, instead of changing the type of variables on a global scale to be a "size_t". Bug 4593 is a good example of how this can cause bugs as a side-effect due to -1 being used as an error indicator.