Programmer’s Reference by *M Gaffiero*

gaffie@users.sourceforge.net

Class::CodeStyler

0.04

Table of Contents
Class::CodeStyler

NAME

Class::CodeStyler - Perl extension for code generation program formatting and execution.

SYNOPSIS

```
use Class::CodeStyler;

# Create a new Perl program codestyle object...
my $p2 = Class::CodeStyler::Program::Perl->new();
$p2->code("sub function_operator");
$p2->open_block();
$p2->code("my \$self = shift;");
$p2->code("my \$arg1 = shift;");
$p2->code("my \$arg2 = shift;");
$p2->code("return \$arg1->eq(\$arg2);");
$p2->close_block();

# prepare() will produce the text for the program...
$p2->prepare();

# print will return the produced text...
print $p2->print();

# Create another Perl program codestyle object...
my $p = Class::CodeStyler::Program::Perl->new(program_name => ’testing.pl’, tab_size=>4);
$p->open_block();
$p->code("package MyBinFun;");

# Set the bookmark ’subs’...
$p->bookmark(’subs’);
$p->indent_off();
# The following comment line will appear un-indented...
$p->comment("Some comment line...");
$p->indent_on();

# Add the code from $p2 object to $p codestyle object...
$p->add($p2);

# and anther codestyle object...
my $p3 = Class::CodeStyler::Program::Perl->new();
$p3->code("sub function_operator_2");
$p3->open_block();
$p3->code("my \$self = shift;");
$p3->code("my \$arg1 = shift;");
$p3->code("my \$arg2 = shift;");
$p3->code("return \$arg1->eq(\$arg2);");
$p3->close_block();

# Jump to the bookmark position ’subs’...
$p->jump(’subs’);

# Add the code from $p3 codestyle object to $p object at the bookmark position...
$p->add($p3);

# Return to the last position prior to last jump...
$p->return();

# Add a divider line...
$p->divider();

# Add a comment line...
$p->comment(’Next function follows...’);

my $p4 = Class::CodeStyler::Program::Perl->new();
$p4->code("sub function_operator_3");
$p4->open_block();
$p4->code("my \$self = shift;");
$p4->code("my \$arg1 = shift;");
$p4->code("my \$arg2 = shift;");
$p4->code("return \$arg1->eq(\$arg2);");
$p4->close_block();
$p->add($p4);
$p->close_block();

# Process the $p codestyle object to produce the code text...
$p->prepare();

# Print the produced code text...
```

```
print $p->print();

# The following is displayed to stdout...

>{
>    package MyBinFun;
>    sub function_operator_2
>    {
>      my $self = shift;
>      my $arg1 = shift;
>      my $arg2 = shift;
>      return $arg1->eq($arg2);
>    }
>    # BOOKMARK ---- subs
>#Some comment line...
>    sub function_operator
>    {
>      my $self = shift;
>      my $arg1 = shift;
>      my $arg2 = shift;
>      return $arg1->eq($arg2);
>    }
>    --------------------------------------------------------------------
>    #Next function follows...
>    sub function_operator_3
>    {
>      my $self = shift;
>      my $arg1 = shift;
>      my $arg2 = shift;
>      return $arg1->eq($arg2);
>    }
>}

# display() will output the code text with line numbers prefixed on each line...
$p->display();

# Save the code text in the program_file...
$p->save();

# Run the syntax checker for the produced program text...
$p->syntax_check();

# execute the generated program file, passing on any stdin. Process control will pass on to
# the generated program...
$p->exec();
```

DESCRIPTION

Class::CodeStyler is an object oriented API to be used by code-gererators in producing formatted code (Perl, C, other). The produced code file can also be syntax checked, displayed with line numbers, executed, and eval‘d. Code can be inserted anywhere in the generated program using a system of bookmarks and jumps.

FUNCTION REFERENCE

Constructor

Use the *new()* function to construct a ***CodeStyle*** object. The following properties may be specified:

program_name

A *program_name* must be specified before the generated code text can be *saved*, *executed*, *eval‘d*, or *syntax checked*.

suppress_comments

Set this property to non-zero to suppress the inclusion of comment text in the generated code text.

tab_size

If ***tab_type*** is `‘spaces’` then the ***tab_size*** indicates the number of spaces per *indent*. If ***tab_type*** is `‘hard’` then the ***tab_size*** specifies the number of *tabs* per *indent*.

tab_type

Specify `'spaces'` or `'hard'`. Defaults to `'spaces'`.

divider_length
Specify the length in characters of the *divider* text line. The default length is *70*.

divider_char
Specify the character to be used when printing a *divider* line. The default character is `-`.

debug
Turn on *debug* messages printed to *stderr*.

code
Add the code text specified in the argument.

open_block
Begin a new block using the bracket characters pecified in the argument (defaults to `{}`). The code that follows will be indented.

close_block
End the block. Move back one indent and print the appropriate closing bracket.

over
Increase the indent by the number specified in the argument (default is one).

back
Decrease the indent by the size specified in the last ***over()***.

indent_off
Turn off indenting. The code that follows and up to the ***indent_on()*** will appear without any indentation.

indent_on
Turn on indenting.

add
Add the ***CodeStyle*** object or objects that are specified in the arguments to this ***CodeStyle*** object.

bookmark
Set the bookmark at the current position with the name as specified by the argument.

jump
Jump to the position bookmarked with the name as specified in the argument.

return

Return to the last position prior to the previous ***jump()***.

newline_on

Turn on *newline*. A newline will be added after every code addition code.

newline_off

Turn off *newline*.

comment

Add a *comment* line. The appropriate language comment prefix will be prefixed.

divider

Add a *divider* line to the code text.

clear

Clear the *prepared* code text. Whenever the ***CodeStyle*** object has been modified after a ***prepare***, the ***clear*** function must be called before *preparing* again.

prepare

Process the ***CodeStyle*** object and produce the code text ready for printing.

print

Print the code text to stdout.

save

Save the generated code text to the file as specified in the ***program_name*** member.

display

Display the generated code text to stdout with line numbers.

syntax_check

Check the syntax of the generated program.

exec

Execute the generated program. The generated program will be executed in place of this (caller) program.

eval

Execute the generated program with the Perl ***eval*** function.

PREREQUISITES

Class::STL::Containers version 0.18 or above is required.

SEE ALSO

AUTHOR

m gaffiero, <gaffie@users.sourceforge.net<gt>

COPYRIGHT AND LICENSE