NAME

Data::Section - read multiple hunks of data out of your DATA section

VERSION

version 0.200006

SYNOPSIS

```
package Letter::Resignation;
use Data::Section -setup;

sub quit {
my ($class, $angry, %arg) = @_;

my $template = $self->section_data(
($angry ? "angry_" : "professional_") . "letter"
);

return fill_in($$template, \%arg);
}

__DATA__
__[ angry_letter ]__
Dear jerks,

I quit!

--
{{ $name }}
__[ professional_letter ]__
Dear {{ $boss }},

I quit, jerks!


--
{{ $name }}
```

DESCRIPTION

Data::Section provides an easy way to access multiple named chunks of line-oriented data in your module's DATA section. It was written to allow modules to store their own templates, but probably has other uses.

WARNING

You will need to use `__DATA__` sections and not `__END__` sections. Yes, it matters. Who knew!

EXPORTS

To get the methods exported by Data::Section, you must import like this:

```
use Data::Section -setup;
```

Optional arguments may be given to Data::Section like this:

```
use Data::Section -setup => { ... };
```

Valid arguments are:

```
encoding - if given, gives the encoding needed to decode bytes in
data sections; default; UTF-8

the special value "bytes" will leave the bytes in the string
```

```
verbatim

inherit - if true, allow packages to inherit the data of the packages
from which they inherit; default: true

header_re - if given, changes the regex used to find section headers
in the data section; it should leave the section name in $1

default_name - if given, allows the first section to has no header and set
its name
```

Three methods are exported by Data::Section:

section_data

```
my $string_ref = $pkg->section_data($name);
```

This method returns a reference to a string containing the data from the name section, either in the invocant's `DATA` section or in that of one of its ancestors. (The ancestor must also derive from the class that imported Data::Section.)

By default, named sections are delimited by lines that look like this:

```
__[ name ]__
```

You can use as many underscores as you want, and the space around the name is optional. This pattern can be configured with the `header_re` option (see above). If present, a single leading \ is removed, so that sections can encode lines that look like section delimiters.

When a line containing only `__END__` is reached, all processing of sections ends.

section_data_names

```
my @names = $pkg->section_data_names;
```

This returns a list of all the names that will be recognized by the `section_data` method.

merged_section_data

```
my $data = $pkg->merged_section_data;
```

This method returns a hashref containing all the data extracted from the package data for all the classes from which the invocant inherits — as long as those classes also inherit from the package into which Data::Section was imported.

In other words, given this inheritance tree:

```
A
\
B C
\ /
D
```

...if Data::Section was imported by A, then when D's `merged_section_data` is invoked, C's data section will not be considered. (This prevents the read position of C's data handle from being altered unexpectedly.)

The keys in the returned hashref are the section names, and the values are **references to** the strings extracted from the data sections.

merged_section_data_names

```
my @names = $pkg->merged_section_data_names;
```

This returns a list of all the names that will be recognized by the `merged_section_data` method.

local_section_data

```
my $data = $pkg->local_section_data;
```

This method returns a hashref containing all the data extracted from the package on which the

method was invoked. If called on an object, it will operate on the package into which the object was blessed.

This method needs to be used carefully, because it's weird. It returns only the data for the package on which it was invoked. If the package on which it was invoked has no data sections, it returns an empty hashref.

local_section_data_names

```
my @names = $pkg->local_section_data_names;
```

This returns a list of all the names that will be recognized by the `local_section_data` method.

TIPS AND TRICKS

MooseX::Declare and namespace::autoclean

The namespace::autoclean library automatically cleans foreign routines from a class, including those imported by Data::Section.

MooseX::Declare does the same thing, and can also cause your `__DATA__` section to appear outside your class's package.

These are easy to address. The Sub::Exporter::ForMethods library provides an installer that will cause installed methods to appear to come from the class and avoid autocleaning. Using an explicit `package` statement will keep the data section in the correct package.

```
package Foo;

use MooseX::Declare;
class Foo {

# Utility to tell Sub::Exporter modules to export methods.
use Sub::Exporter::ForMethods qw( method_installer );

# method_installer returns a sub.
use Data::Section { installer => method_installer }, -setup;

method my_method {
my $content_ref = $self->section_data('SectionA');

print $$content_ref;
}
}

__DATA__
__[ SectionA ]__
Hello, world.
```

SEE ALSO

- article for RJBS Advent 2009 <http://advent.rjbs.manxome.org/2009/2009-12-09.html>
- Inline::Files does something that is at first look similar,

  but it works with source filters, and contains the warning:

  ```
  It is possible that this module may overwrite the source code in files that
  use it. To protect yourself against this possibility, you are strongly
  advised to use the -backup option described in "Safety first".
  ```

  Enough said.

AUTHOR

Ricardo SIGNES <rjbs@cpan.org>

COPYRIGHT AND LICENSE