NAME

MooseX::Declare::Syntax::NamespaceHandling − Handle namespaced blocks

VERSION

version 0.43

DESCRIPTION

Allows the implementation of namespaced blocks like the role and class keyword handlers.

Namespaces are automatically nested. Meaning that, for example, a `class Bar` declaration inside another `class Foo` block gives the inner one actually the name `Foo::Bar`.

METHODS

parse

```
Any Object−>parse (Object $context)
```

This is the main handling routine for namespaces. It will remove the namespace name and its options. If the handler was invoked without a name, options or a following block, it is assumed that this is an instance of an autoquoted bareword like `class => "Foo"`.

The return value of the `parse` method is also the value that is returned to the user of the keyword.

CONSUMES

- MooseX::Declare::Syntax::KeywordHandling
- MooseX::Declare::Syntax::InnerSyntaxHandling

REQUIRED METHODS

handle_missing_block

```
Object−>handle_missing_block (Object $context, Str $body, %args)
```

This must be implemented to decide what to do in case the statement is terminated rather than followed by a block. It will receive the context object, the produced code that needs to be injected, and all the arguments that were passed to the call to ‘‘inject_code_parts’’ in MooseX::Declare::Context.

The return value will be ignored.

EXTENDABLE STUB METHODS

add_namespace_customizations
add_optional_customizations

```
Object−>add_namespace_customizations (Object $context, Str $package, HashRef $o
Object−>add_optional_customizations  (Object $context, Str $package, HashRef $o
```

These will be called (in this order) by the ‘‘parse’’ method. They allow specific hooks to attach before/after/around the customizations for the namespace and the provided options that are not attached to the namespace directly.

While this distinction might seem superficial, we advise library developers facilitating this role to follow the precedent. This ensures that when another component needs to tie between the namespace and any additional customizations everything will run in the correct order. An example of this separation would be

```
class Foo is mutable ...
```

being an option of the namespace generation, while

```
class Foo with Bar ...
```

is an additional optional customization.

handle_post_parsing

```
Object−>handle_post_parsing (Object $context, Str $package, Str | Object $name)
```

Allows for additional modifications to the namespace after everything else has been done. It will receive the context, the fully qualified package name, and either a string with the name that was specified (might not be fully qualified, since namespaces can be nested) or the anonymous metaclass instance if no name was specified.

The return value of this method will be the value returned to the user of the keyword. If you always return the `$package` argument like this:

```
sub handle_post_parsing {
    my ($self, $context, $package, $name) = @_;
    return $package;
}
```

and set this up in a `foo` keyword handler, you can use it like this:

```
foo Cthulhu {

    my $fhtagn = foo Fhtagn { }
    my $anon   = foo { };

    say $fhtagn;  # Cthulhu::Fhtagn
    say $anon;    # some autogenerated package name
}
```

make_anon_metaclass

```
Class::MOP::Class Object->make_anon_metaclass ()
```

This method should be overridden if you want to provide anonymous namespaces.

It does not receive any arguments for customization of the metaclass, because the configuration and customization will be done by MooseX::Declare in the package of the generated class in the same way as in those that have specified names. This way ensures that anonymous and named namespaces are always handled equally.

If you do not extend this method (it will return nothing by default), an error will be thrown when a user attempts to declare an anonymous namespace.

SEE ALSO

- MooseX::Declare
- MooseX::Declare::Syntax::MooseSetup

AUTHOR

Florian Ragwitz <rafl AT debian DOT org>

COPYRIGHT AND LICENSE