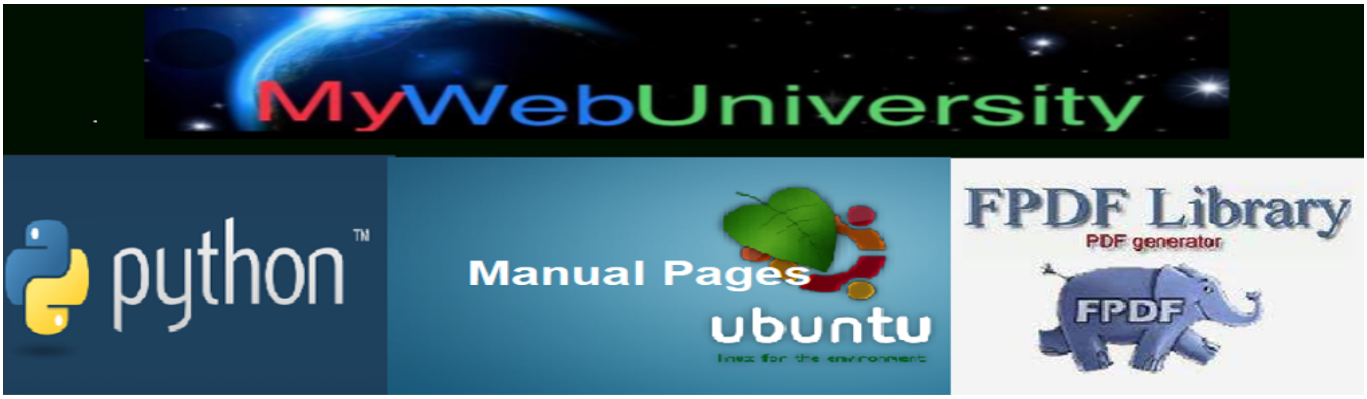

Rocky Enterprise Linux 9.2 Manual Pages on command 'autodie::Util.3perl'

$ man autodie::Util.3perl

autodie::Util(3perl) Perl Programmers Reference Guide autodie::Util(3perl)

NAME

autodie::Util - Internal Utility subroutines for autodie and Fatal

SYNOPSIS

```
# INTERNAL API for autodie and Fatal only!

use autodie::Util qw(on_end_of_compile_scope);
on_end_of_compile_scope(sub { print "Hallo world\n"; });
```

DESCRIPTION

Interal Utilities for autodie and Fatal! This module is not a part of autodie's public API.

This module contains utility subroutines for abstracting away the underlying magic of autodie and (ab)uses of "%^H" to call subs at the end of a (compile-time) scopes.

Note that due to how "%^H" works, some of these utilities are only useful during the compilation phase of a perl module and relies on the internals of how perl handles references in "%^H".

Methods

on_end_of_compile_scope

```
on_end_of_compile_scope(sub { print "Hallo world\n"; });
```

Will invoke a sub at the end of a (compile-time) scope. The sub is called once with no arguments. Can be called multiple times (even in the same "compile-time" scope) to install multiple subs. Subs are called in a "first-in-last-out"-order (FILO or "stack"-order).

fill_protos

```
fill_protos('*$$;$@')
```

Given a Perl subroutine prototype, return a list of invocation specifications. Each specification is a listref, where the first member is the (minimum) number of arguments for this invocation specification. The remaining arguments are a string representation of how to pass the arguments correctly to a sub with the given prototype, when called with the given number of arguments.

The specifications are returned in increasing order of arguments starting at 0 (e.g. ';$') or 1 (e.g. '$@'). Note that if the prototype is "slurpy" (e.g. ends with a "@"), the number of arguments for the last specification is a "minimum" number rather than an exact number. This can be detected by the last member of the last specification matching m/[@#]_/.

make_core_trampoline

```
make_core_trampoline('CORE::open', 'main', prototype('CORE::open'))
```

Creates a trampoline for calling a core sub. Essentially, a tiny sub that figures out how we should be calling our core sub, puts in the arguments in the right way, and bounces our control over to it.

If we could reliably use `goto &` on core builtins, we wouldn't need this subroutine.

install_subs

```
install_subs('My::Module', { 'read' => sub { die("Hallo\n"), ... }})
```

Given a package name and a hashref mapping names to a subroutine reference (or "undef"), this subroutine will install said subroutines on their given name in that module. If a name mapes to "undef", any subroutine with that name in the target module will be remove (possibly "unshadowing" a CORE sub of same name).

AUTHOR

LICENSE