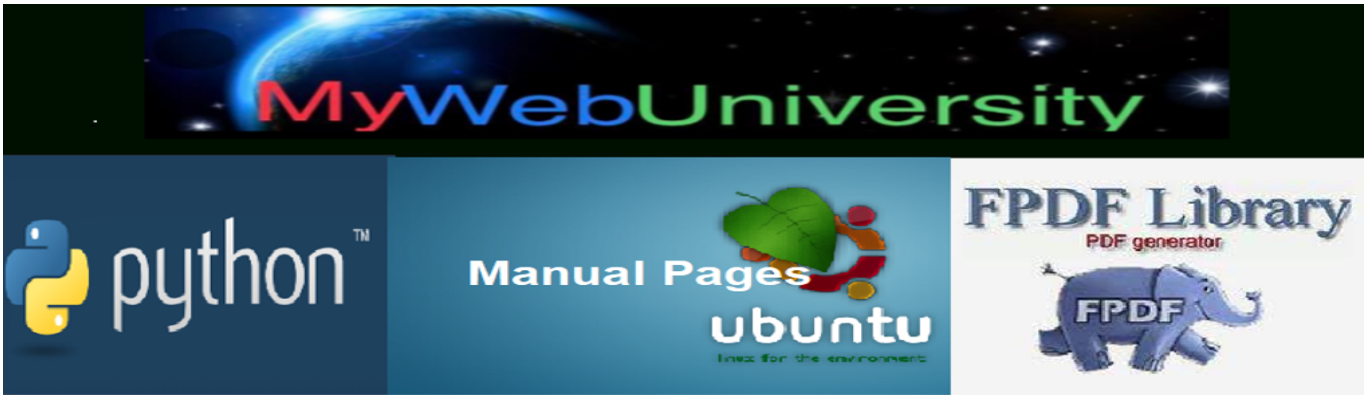




## *Rocky Enterprise Linux 9.2 Manual Pages on command 'Package::Stash.3pm'*

***$ man Package::Stash.3pm***

Package::Stash(3pm) User Contributed Perl Documentation Package::Stash(3pm)

NAME

Package::Stash - Routines for manipulating stashes

VERSION

version 0.39

SYNOPSIS

```
my $stash = Package::Stash->new('Foo');
$stash->add_symbol('%foo', {bar => 1});
# $Foo::foo{bar} == 1
$stash->has_symbol('$foo') # false
my $namespace = $stash->namespace;
*{ $namespace->{foo} }{HASH} # {bar => 1}
```

DESCRIPTION

Manipulating stashes (Perl's symbol tables) is occasionally necessary, but incredibly messy, and easy to get wrong. This module hides all of that behind a simple API.

NOTE: Most methods in this class require a variable specification that includes a sigil. If this sigil is absent, it is assumed to represent the IO slot.

Due to limitations in the typeglob API available to perl code, and to typeglob manipulation in perl being quite slow, this module provides two implementations - one in pure perl, and one using XS. The XS implementation is to be preferred for most usages; the pure perl one is provided for cases where XS modules are not a possibility. The current implementation in use can be set by setting $ENV{PACKAGE_STASH_IMPLEMENTATION} or $Package::Stash::IMPLEMENTATION before loading Package::Stash (with the environment

variable taking precedence), otherwise, it will use the XS implementation if possible, falling back to the pure perl one.

## METHODS

### new $package_name

Creates a new "Package::Stash" object, for the package given as the only argument.

### name

Returns the name of the package that this object represents.

### namespace

Returns the raw stash itself.

### add_symbol $variable $value %opts

Adds a new package symbol, for the symbol given as $variable, and optionally gives it an initial value of $value. $variable should be the name of variable including the sigil, so

```
Package::Stash->new('Foo')->add_symbol('%foo')
```

will create %Foo::foo.

Valid options (all optional) are "filename", "first_line_num", and "last_line_num". $opts{filename}, $opts{first_line_num}, and $opts{last_line_num} can be used to indicate where the symbol should be regarded as having been defined. Currently these values are only used if the symbol is a subroutine ('"&"' sigil) and only if "$^P & 0x10" is true, in which case the special %DB::sub hash is updated to record the values of "filename", "first_line_num", and "last_line_num" for the subroutine. If these are not passed, their values are inferred (as much as possible) from "caller" information.

### remove_glob $name

Removes all package variables with the given name, regardless of sigil.

### has_symbol $variable

Returns whether or not the given package variable (including sigil) exists.

### get_symbol $variable

Returns the value of the given package variable (including sigil).

### get_or_add_symbol $variable

Like "get_symbol", except that it will return an empty hashref or arrayref if the variable doesn't exist.

### remove_symbol $variable

Removes the package variable described by $variable (which includes the sigil); other variables with the same name but different sigils will be untouched.

list_all_symbols $type_filter

Returns a list of package variable names in the package, without sigils. If a "type_filter" is passed, it is used to select package variables of a given type, where valid types are the slots of a typeglob ('SCALAR', 'CODE', 'HASH', etc). Note that if the package contained any "BEGIN" blocks, perl will leave an empty typeglob in the "BEGIN" slot, so this will show up if no filter is used (and similarly for "INIT", "END", etc).

get_all_symbols $type_filter

Returns a hashref, keyed by the variable names in the package. If $type_filter is passed, the hash will contain every variable of that type in the package as values, otherwise, it will contain the typeglobs corresponding to the variable names (basically, a clone of the stash).

This is especially useful for debuggers and profilers, which use %DB::sub to determine where the source code for a subroutine can be found. See <http://perldoc.perl.org/perldebguts.html#Debugger-Internals> for more information about %DB::sub.

WORKING WITH VARIABLES

It is important to note, that when working with scalar variables, the default behavior is to copy values.

```
my $stash = Package::Stash->new('Some::Namespace');
my $variable = 1;
# $Some::Namespace::name is a copy of $variable
$stash->add_symbol('$name', $variable);
$variable++
# $Some::Namespace::name == 1 , $variable == 2
```

This will likely confuse people who expect it to work the same as typeglob assignment, which simply creates new references to existing variables.

```
my $variable = 1;
{
    no strict 'refs';
    # assign $Package::Stash::name = $variable
    *{'Package::Stash::name'} = \$variable;
}
$variable++ # affects both names
```

If this behaviour is desired when working with Package::Stash, simply pass Package::Stash a scalar ref:

```
my $stash = Package::Stash->new('Some::Namespace');
my $variable = 1;
# $Some::Namespace::name is now $variable
$stash->add_symbol('$name', \$variable);
$variable++
# $Some::Namespace::name == 2 , $variable == 2
```

This will be what you want as well if you're ever working with Readonly variables:

```
use Readonly;
Readonly my $value, 'hello';
$stash->add_symbol('$name', \$value); # reference
print $Some::Namespace::name; # hello
# Tries to modify the read-only 'hello' and dies.
$Some::Namespace::name .= " world";
$stash->add_symbol('$name', $value); # copy
print $Some::Namespace::name; # hello
# No problem, modifying a copy, not the original
$Some::Namespace::name .= " world";
```

## SEE ALSO

? Class::MOP::Package

This module is a factoring out of code that used to live here

## HISTORY

Based on code from Class::MOP::Package, by Stevan Little and the Moose Cabal.

## BUGS / CAVEATS

? Prior to perl 5.10, scalar slots are only considered to exist if they are defined

This is due to a shortcoming within perl itself. See "Making References" in perlref point 7 for more information.

? GLOB and FORMAT variables are not (yet) accessible through this module.

? Also, see the BUGS section for the specific backends (Package::Stash::XS and Package::Stash::PP)

Bugs may be submitted through the RT bug tracker <https://rt.cpan.org/Public/Dist/Display.html?Name=Package-Stash> (or

bug-Package-Stash@rt.cpan.org <mailto:bug-Package-Stash@rt.cpan.org>).

## AUTHORS

- Stevan Little <stevan.little@iinteractive.com>
- Jesse Luehrs <doy@tozt.net>

## CONTRIBUTORS

- Karen Etheridge <ether@cpan.org>
- Carlos Lima <carlos@multi>
- Dave Rolsky <autarch@urth.org>
- Justin Hunter <justin.d.hunter@gmail.com>
- Christian Walde <walde.christian@googlemail.com>
- Kent Fredric <kentfredric@gmail.com>
- Niko Tyni <ntyni@debian.org>
- Renee <reb@perl-services.de>
- Tim Bunce <Tim.Bunce@pobox.com>

## COPYRIGHT AND LICENSE