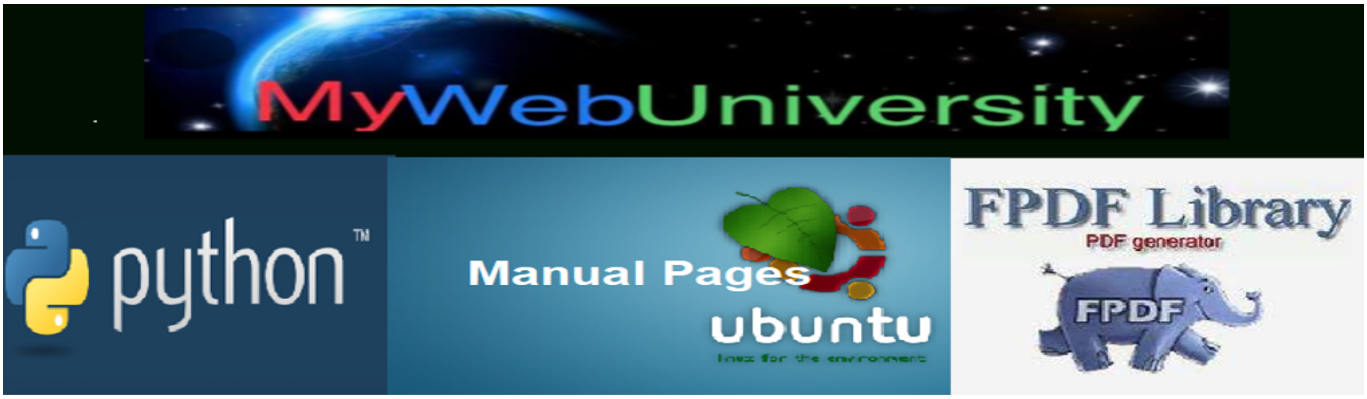

Rocky Enterprise Linux 9.2 Manual Pages on command 'Test2::Util::HashBase.3perl'

$ man Test2::Util::HashBase.3perl

NAME

Test2::Util::HashBase - Build hash based classes.

SYNOPSIS

A class:

```
package My::Class;
use strict;
use warnings;

# Generate 3 accessors
use Test2::Util::HashBase qw/foo -bar ^baz <bat >ban +boo/;

# Chance to initialize defaults
sub init {
    my $self = shift;    # No other args
    $self->{+FOO} ||= "foo";
    $self->{+BAR} ||= "bar";
    $self->{+BAZ} ||= "baz";
    $self->{+BAT} ||= "bat";
    $self->{+BAN} ||= "ban";
```

```
    $self->{+BOO} ||= "boo";
}

sub print {
    print join '', '' => map { $self->{$_} } FOO, BAR, BAZ, BAT, BAN, BOO;
}
```

Subclass it

```
package My::Subclass;
use strict;
use warnings;

# Note, you should subclass before loading HashBase.
use base 'My::Class';
use Test2::Util::HashBase qw/bub/;

sub init {
    my $self = shift;

    # We get the constants from the base class for free.
    $self->{+FOO} ||= 'SubFoo';
    $self->{+BUB} ||= 'bub';

    $self->SUPER::init();
}
```

use it:

```
package main;
use strict;
use warnings;
use My::Class;
```

```
    # These are all functionally identical
    my $one   = My::Class->new(foo => 'MyFoo', bar => 'MyBar');
    my $two   = My::Class->new({foo => 'MyFoo', bar => 'MyBar'});
    my $three = My::Class->new(['MyFoo', 'MyBar']);

    # Readers!
    my $foo = $one->foo;    # 'MyFoo'
    my $bar = $one->bar;    # 'MyBar'
    my $baz = $one->baz;    # Defaulted to: 'baz'
    my $bat = $one->bat;    # Defaulted to: 'bat'
    # '>ban' means setter only, no reader
    # '+boo' means no setter or reader, just the BOO constant

    # Setters!
    $one->set_foo('A Foo');

    #'-bar' means read-only, so the setter will throw an exception (but is defined).
    $one->set_bar('A bar');

    # '^baz' means deprecated setter, this will warn about the setter being
    # deprecated.
    $one->set_baz('A Baz');

    # '<bat' means no setter defined at all
    # '+boo' means no setter or reader, just the BOO constant

    $one->{+FOO} = 'xxx';
```

DESCRIPTION

This package is used to generate classes based on hashrefs. Using this class will give you a "new()" method, as well as generating accessors you request. Generated accessors will be getters, "set_ACCESSOR" setters will also be generated for you. You also get constants

for each accessor (all caps) which return the key into the hash for that accessor. Single inheritance is also supported.

THIS IS A BUNDLED COPY OF HASHBASE

This is a bundled copy of Object::HashBase. This file was generated using the "/home/exodist/perl5/perlbrew/perls/main/bin/hashbase_inc.pl" script.

METHODS

PROVIDED BY HASH BASE

$it = $class->new(%PAIRS)

$it = $class->new(\%PAIRS)

$it = $class->new(\@ORDERED_VALUES)

Create a new instance.

HashBase will not export "new()" if there is already a "new()" method in your packages inheritance chain.

If you do not want this method you can define your own you just have to declare it before loading Test2::Util::HashBase.

```
package My::Package;

# predeclare new() so that HashBase does not give us one.
sub new;

use Test2::Util::HashBase qw/foo bar baz/;

# Now we define our own new method.
sub new { ... }
```

This makes it so that HashBase sees that you have your own "new()" method. Alternatively you can define the method before loading HashBase instead of just declaring it, but that scatters your use statements.

The most common way to create an object is to pass in key/value pairs where each key is an attribute and each value is what you want assigned to that attribute. No checking is done to verify the attributes or values are valid, you may do that in "init()" if desired.

If you would like, you can pass in a hashref instead of pairs. When you do so the hashref will be copied, and the copy will be returned blessed as an object. There is no way to ask HashBase to bless a specific hashref.

In some cases an object may only have 1 or 2 attributes, in which case a hashref may be too verbose for your liking. In these cases you can pass in an arrayref with only values. The values will be assigned to attributes in the order the attributes were listed. When there is inheritance involved the attributes from parent classes will come before subclasses.

HOOKS

$self->init()

This gives you the chance to set some default values to your fields. The only argument is $self with its indexes already set from the constructor.

Note: Test2::Util::HashBase checks for an init using "$class->can('init')" during construction. It DOES NOT call "can()" on the created object. Also note that the result of the check is cached, it is only ever checked once, the first time an instance of your class is created. This means that adding an "init()" method AFTER the first construction will result in it being ignored.

ACCESSORS

READ/WRITE

To generate accessors you list them when using the module:

```
use Test2::Util::HashBase qw/foo/;
```

This will generate the following subs in your namespace:

foo()

Getter, used to get the value of the "foo" field.

set_foo()

Setter, used to set the value of the "foo" field.

FOO()

Constant, returns the field "foo"'s key into the class hashref. Subclasses will also get this function as a constant, not simply a method, that means it is copied into the subclass namespace.

The main reason for using these constants is to help avoid spelling mistakes and similar typos. It will not help you if you forget to prefix the '+' though.

READ ONLY

use Test2::Util::HashBase qw/-foo/;

set_foo()

Throws an exception telling you the attribute is read-only. This is exported to override any active setters for the attribute in a parent class.

DEPRECATED SETTER

use Test2::Util::HashBase qw/^foo/;

set_foo()

This will set the value, but it will also warn you that the method is deprecated.

NO SETTER

use Test2::Util::HashBase qw/<foo/;

Only gives you a reader, no "set_foo" method is defined at all.

NO READER

```
use Test2::Util::HashBase qw/>foo/;
```

Only gives you a write ("set_foo"), no "foo" method is defined at all.

CONSTANT ONLY

```
use Test2::Util::HashBase qw/+foo/;
```

This does not create any methods for you, it just adds the "FOO" constant.

SUBCLASSING

You can subclass an existing HashBase class.

```
use base 'Another::HashBase::Class';
use Test2::Util::HashBase qw/foo bar baz/;
```

The base class is added to @ISA for you, and all constants from base classes are added to subclasses automatically.

GETTING A LIST OF ATTRIBUTES FOR A CLASS

Test2::Util::HashBase provides a function for retrieving a list of attributes for an Test2::Util::HashBase class.

```
@list = Test2::Util::HashBase::attr_list($class)
@list = $class->Test2::Util::HashBase::attr_list()
```

Either form above will work. This will return a list of attributes defined on the object. This list is returned in the attribute definition order, parent class attributes are listed before subclass attributes. Duplicate attributes will be removed before the list is returned.

Note: This list is used in the "$class->new(\@ARRAY)" constructor to determine the attribute to which each value will be paired.

SOURCE

The source code repository for HashBase can be found at

http://github.com/Test-More/HashBase/.

MAINTAINERS

Chad Granum <exodist@cpan.org>

AUTHORS

Chad Granum <exodist@cpan.org>

COPYRIGHT