

NAME

`Scalar::Util` - A selection of general-utility scalar subroutines

SYNOPSIS

```
use Scalar::Util qw(blessed dualvar isweak readonly refaddr reftype
tainted
                    weaken isvstring looks_like_number set_prototype);
```

DESCRIPTION

`Scalar::Util` contains a selection of subroutines that people have expressed would be nice to have in the perl core, but the usage would not really be high enough to warrant the use of a keyword, and the size so small such that being individual extensions would be wasteful.

By default `Scalar::Util` does not export any subroutines. The subroutines defined are

blessed EXPR

If EXPR evaluates to a blessed reference the name of the package that it is blessed into is returned. Otherwise `undef` is returned.

```
$scalar = "foo";
$class  = blessed $scalar;           # undef

$ref     = [];
$class   = blessed $ref;             # undef

$obj     = bless [], "Foo";
$class   = blessed $obj;             # "Foo"
```

dualvar NUM, STRING

Returns a scalar that has the value NUM in a numeric context and the value STRING in a string context.

```
$foo = dualvar 10, "Hello";
$num = $foo + 2;           # 12
$str = $foo . " world";    # Hello world
```

isvstring EXPR

If EXPR is a scalar which was coded as a vstring the result is true.

```
$vs    = v49.46.48;
$fmt   = isvstring($vs) ? "%vd" : "%s"; #true
printf($fmt,$vs);
```

isweak EXPR

If EXPR is a scalar which is a weak reference the result is true.

```
$ref = \$foo;
$weak = isweak($ref);           # false
weaken($ref);
$weak = isweak($ref);           # true
```

NOTE: Copying a weak reference creates a normal, strong, reference.

```
$copy = $ref;
$weak = isweak($ref);           # false
```

`looks_like_number` EXPR

Returns true if perl thinks EXPR is a number. See *"looks_like_number" in perlapi*.

`openhandle` FH

Returns FH if FH may be used as a filehandle and is open, or FH is a tied handle. Otherwise undef is returned.

```
$fh = openhandle(*STDIN); # \*STDIN
$fh = openhandle(\*STDIN); # \*STDIN
$fh = openhandle(*NOTOPEN); # undef
$fh = openhandle("scalar"); # undef
```

`readonly` SCALAR

Returns true if SCALAR is readonly.

```
sub foo { readonly($_[0]) }

$readonly = foo($bar); # false
$readonly = foo(0);    # true
```

`refaddr` EXPR

If EXPR evaluates to a reference the internal memory address of the referenced value is returned. Otherwise undef is returned.

```
$addr = refaddr "string"; # undef
$addr = refaddr \ $var;   # eg 12345678
$addr = refaddr [];      # eg 23456784

$obj = bless {}, "Foo";
$addr = refaddr $obj;    # eg 88123488
```

`reftype` EXPR

If EXPR evaluates to a reference the type of the variable referenced is returned. Otherwise undef is returned.

```
$type = reftype "string"; # undef
$type = reftype \ $var;   # SCALAR
$type = reftype [];      # ARRAY

$obj = bless {}, "Foo";
$type = reftype $obj;    # HASH
```

`set_prototype` CODEREF, PROTOTYPE

Sets the prototype of the given function, or deletes it if PROTOTYPE is undef. Returns the CODEREF.

```
set_prototype \&foo, '$$';
```

`tainted` EXPR

Return true if the result of EXPR is tainted

```
$taint = tainted("constant"); # false
$taint = tainted($ENV{PWD});  # true if running under -T
```

`weaken` REF

REF will be turned into a weak reference. This means that it will not hold a reference count on

the object it references. Also when the reference count on that object reaches zero, REF will be set to undef.

This is useful for keeping copies of references, but you don't want to prevent the object being DESTROY-ed at its usual time.

```
{
    my $var;
    $ref = \$var;
    weaken($ref);                # Make $ref a weak reference
}
# $ref is now undef
```

Note that if you take a copy of a scalar with a weakened reference, the copy will be a strong reference.

```
my $var;
my $foo = \$var;
weaken($foo);                  # Make $foo a weak reference
my $bar = $foo;                 # $bar is now a strong
reference
```

This may be less obvious in other situations, such as `grep()`, for instance when grepping through a list of weakened references to objects that may have been destroyed already:

```
@object = grep { defined } @object;
```

This will indeed remove all references to destroyed objects, but the remaining references to objects will be strong, causing the remaining objects to never be destroyed because there is now always a strong reference to them in the `@object` array.

KNOWN BUGS

There is a bug in perl5.6.0 with UV's that are $\geq 1 < 31$. This will show up as tests 8 and 9 of `dualvar.t` failing

SEE ALSO

List::Util

COPYRIGHT

Copyright (c) 1997-2006 Graham Barr <gbarr@pobox.com>. All rights reserved. This program is free software; you can redistribute it and/or modify it under the same terms as Perl itself.

Except `weaken` and `isweak` which are

Copyright (c) 1999 Tuomas J. Lukka <lukka@iki.fi>. All rights reserved. This program is free software; you can redistribute it and/or modify it under the same terms as perl itself.

BLATANT PLUG

The `weaken` and `isweak` subroutines in this module and the patch to the core Perl were written in connection with the APress book 'Tuomas J. Lukka's Definitive Guide to Object-Oriented Programming in Perl', to avoid explaining why certain things would have to be done in cumbersome ways.