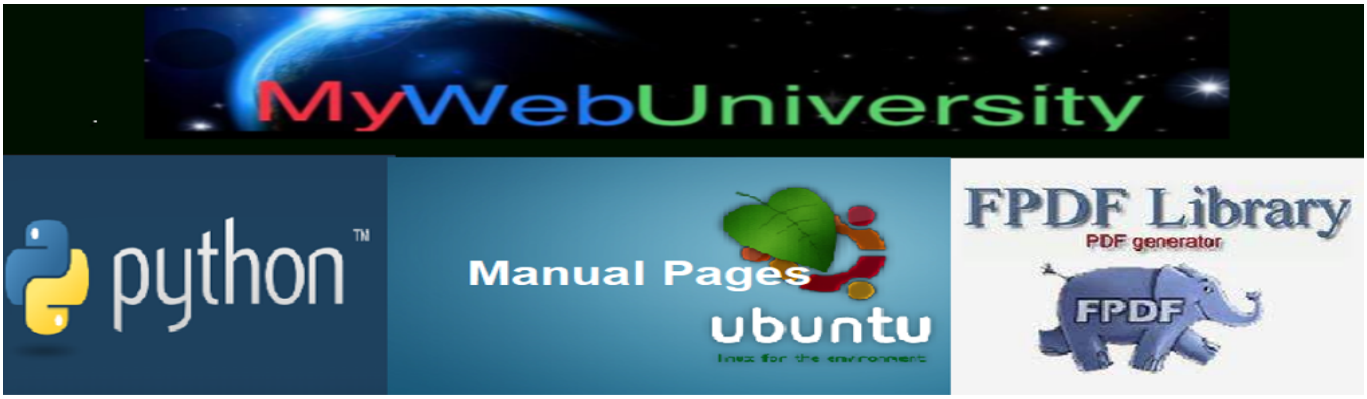

Rocky Enterprise Linux 9.2 Manual Pages on command 'Tie::Hash.3perl'

$ man Tie::Hash.3perl

Tie::Hash(3perl) Perl Programmers Reference Guide Tie::Hash(3perl)

NAME

Tie::Hash, Tie::StdHash, Tie::ExtraHash - base class definitions for tied hashes

SYNOPSIS

```
package NewHash;
require Tie::Hash;
@ISA = qw(Tie::Hash);
sub DELETE { ... }      # Provides needed method
sub CLEAR { ... }       # Overrides inherited method
package NewStdHash;
require Tie::Hash;
@ISA = qw(Tie::StdHash);
# All methods provided by default, define
# only those needing overrides
# Accessors access the storage in %{$_[0]};
# TIEHASH should return a reference to the actual storage
sub DELETE { ... }
package NewExtraHash;
require Tie::Hash;
@ISA = qw(Tie::ExtraHash);
# All methods provided by default, define
# only those needing overrides
# Accessors access the storage in %{$_[0][0]};
```

```
    # TIEHASH should return an array reference with the first element
    # being the reference to the actual storage
    sub DELETE {
      $_[0][1]->('del', $_[0][0], $_[1]); # Call the report writer
      delete $_[0][0]->{$_[1]};           #  $_[0]->SUPER::DELETE($_[1])
    }
    package main;
    tie %new_hash, 'NewHash';
    tie %new_std_hash, 'NewStdHash';
    tie %new_extra_hash, 'NewExtraHash',
        sub {warn "Doing \U$_[1]\E of $_[2].\n"};
```

DESCRIPTION

This module provides some skeletal methods for hash-tying classes. See perltie for a list of the functions required in order to tie a hash to a package. The basic Tie::Hash package provides a "new" method, as well as methods "TIEHASH", "EXISTS" and "CLEAR". The Tie::StdHash and Tie::ExtraHash packages provide most methods for hashes described in perltie (the exceptions are "UNTIE" and "DESTROY"). They cause tied hashes to behave exactly like standard hashes, and allow for selective overwriting of methods. Tie::Hash grandfathers the "new" method: it is used if "TIEHASH" is not defined in the case a class forgets to include a "TIEHASH" method.

For developers wishing to write their own tied hashes, the required methods are briefly defined below. See the perltie section for more detailed descriptive, as well as example code:

TIEHASH classname, LIST

The method invoked by the command "tie %hash, classname". Associates a new hash instance with the specified class. "LIST" would represent additional arguments (along the lines of AnyDBM_File and compatriots) needed to complete the association.

STORE this, key, value

Store datum value into key for the tied hash this.

FETCH this, key

Retrieve the datum in key for the tied hash this.

FIRSTKEY this

Return the first key in the hash.

NEXTKEY this, lastkey

Return the next key in the hash.

EXISTS this, key

Verify that key exists with the tied hash this.

The Tie::Hash implementation is a stub that simply croaks.

DELETE this, key

Delete the key key from the tied hash this.

CLEAR this

Clear all values from the tied hash this.

SCALAR this

Returns what evaluating the hash in scalar context yields.

Tie::Hash does not implement this method (but Tie::StdHash and Tie::ExtraHash do).

Inheriting from Tie::StdHash

The accessor methods assume that the actual storage for the data in the tied hash is in the hash referenced by "tied(%tiedhash)". Thus overwritten "TIEHASH" method should return a hash reference, and the remaining methods should operate on the hash referenced by the first argument:

```
package ReportHash;
our @ISA = 'Tie::StdHash';
sub TIEHASH  {
  my $storage = bless {}, shift;
  warn "New ReportHash created, stored in $storage.\n";
  $storage
}
sub STORE    {
  warn "Storing data with key $_[1] at $_[0].\n";
  $_[0]{$_[1]} = $_[2]
}
```

Inheriting from Tie::ExtraHash

The accessor methods assume that the actual storage for the data in the tied hash is in the hash referenced by "(tied(%tiedhash))->[0]". Thus overwritten "TIEHASH" method should return an array reference with the first element being a hash reference, and the remaining methods should operate on the hash "%{ $_[0]->[0] }":

```
package ReportHash;
our @ISA = 'Tie::ExtraHash';
sub TIEHASH  {
  my $class = shift;
  my $storage = bless [{}, @_], $class;
  warn "New ReportHash created, stored in $storage.\n";
  $storage;
}
sub STORE    {
  warn "Storing data with key $_[1] at $_[0].\n";
  $_[0][0]{$_[1]} = $_[2]
}
```

The default "TIEHASH" method stores "extra" arguments to tie() starting from offset 1 in the array referenced by "tied(%tiedhash)"; this is the same storage algorithm as in TIEHASH subroutine above. Hence, a typical package inheriting from Tie::ExtraHash does not need to overwrite this method.

"SCALAR", "UNTIE" and "DESTROY"

The methods "UNTIE" and "DESTROY" are not defined in Tie::Hash, Tie::StdHash, or Tie::ExtraHash. Tied hashes do not require presence of these methods, but if defined, the methods will be called in proper time, see perltie.

"SCALAR" is only defined in Tie::StdHash and Tie::ExtraHash.

If needed, these methods should be defined by the package inheriting from Tie::Hash, Tie::StdHash, or Tie::ExtraHash. See "SCALAR" in perltie to find out what happens when "SCALAR" does not exist.

MORE INFORMATION

The packages relating to various DBM-related implementations (DB_File, NDBM_File, etc.) show examples of general tied hashes, as does the Config module. While these do not utilize Tie::Hash, they serve as good working examples.