NAME

Hash::Util - A selection of general-utility hash subroutines

SYNOPSIS

```
    # Restricted hashes

    use Hash::Util qw(
                       hash_seed all_keys
                       lock_keys unlock_keys
                       lock_value unlock_value
                       lock_hash unlock_hash
                       lock_keys_plus hash_locked
                       hidden_keys legal_keys
                     );

    %hash = (foo => 42, bar => 23);
    # Ways to restrict a hash
    lock_keys(%hash);
    lock_keys(%hash, @keyset);
    lock_keys_plus(%hash, @additional_keys);

    # Ways to inspect the properties of a restricted hash
    my @legal = legal_keys(%hash);
    my @hidden = hidden_keys(%hash);
    my $ref = all_keys(%hash,@keys,@hidden);
    my $is_locked = hash_locked(%hash);

    # Remove restrictions on the hash
    unlock_keys(%hash);

    # Lock individual values in a hash
    lock_value  (%hash, 'foo');
    unlock_value(%hash, 'foo');

    # Ways to change the restrictions on both keys and values
    lock_hash  (%hash);
    unlock_hash(%hash);

    my $hashes_are_randomised = hash_seed() != 0;
```

DESCRIPTION

`Hash::Util` and `Hash::Util::FieldHash` contain special functions for manipulating hashes that don't really warrant a keyword.

`Hash::Util` contains a set of functions that support *restricted hashes*. These are described in this document. `Hash::Util::FieldHash` contains an (unrelated) set of functions that support the use of hashes in *inside-out classes*, described in *Hash::Util::FieldHash*.

By default `Hash::Util` does not export anything.

Restricted hashes

5.8.0 introduces the ability to restrict a hash to a certain set of keys. No keys outside of this set can be added. It also introduces the ability to lock an individual key so it cannot be deleted and the ability to

ensure that an individual value cannot be changed.

This is intended to largely replace the deprecated pseudo-hashes.

lock_keys

unlock_keys

```
lock_keys(%hash);
lock_keys(%hash, @keys);
```

Restricts the given %hash's set of keys to @keys. If @keys is not given it restricts it to its current keyset. No more keys can be added. delete() and exists() will still work, but will not alter the set of allowed keys. **Note**: the current implementation prevents the hash from being bless()ed while it is in a locked state. Any attempt to do so will raise an exception. Of course you can still bless() the hash before you call lock_keys() so this shouldn't be a problem.

```
unlock_keys(%hash);
```

Removes the restriction on the %hash's keyset.

Note that if any of the values of the hash have been locked they will not be unlocked after this sub executes.

Both routines return a reference to the hash operated on.

lock_keys_plus

```
lock_keys_plus(%hash,@additional_keys)
```

Similar to `lock_keys()`, with the difference being that the optional key list specifies keys that may or may not be already in the hash. Essentially this is an easier way to say

```
lock_keys(%hash,@additional_keys,keys %hash);
```

Returns a reference to %hash

lock_value

unlock_value

```
lock_value  (%hash, $key);
unlock_value(%hash, $key);
```

Locks and unlocks the value for an individual key of a hash. The value of a locked key cannot be changed.

Unless %hash has already been locked the key/value could be deleted regardless of this setting.

Returns a reference to the %hash.

lock_hash

unlock_hash

```
lock_hash(%hash);
```

lock_hash() locks an entire hash, making all keys and values read-only. No value can be changed, no keys can be added or deleted.

```
unlock_hash(%hash);
```

unlock_hash() does the opposite of lock_hash(). All keys and values are made writable. All values can be changed and keys can be added and deleted.

Returns a reference to the %hash.

lock_hash_recurse

unlock_hash_recurse

```
lock_hash_recurse(%hash);
```

lock_hash() locks an entire hash and any hashes it references recursively, making all keys and values read-only. No value can be changed, no keys can be added or deleted.

Only recurses into hashes that are referenced by another hash. Thus a Hash of Hashes (HoH) will all be restricted, but a Hash of Arrays of Hashes (HoAoH) will only have the top hash restricted.

```
unlock_hash_recurse(%hash);
```

unlock_hash_recurse() does the opposite of lock_hash_recurse(). All keys and values are made writable. All values can be changed and keys can be added and deleted. Identical recursion restrictions apply as to lock_hash_recurse().

Returns a reference to the %hash.

hash_unlocked

```
hash_unlocked(%hash) and print "Hash is unlocked!\n";
```

Returns true if the hash and its keys are unlocked.

legal_keys

```
my @keys = legal_keys(%hash);
```

Returns the list of the keys that are legal in a restricted hash. In the case of an unrestricted hash this is identical to calling keys(%hash).

hidden_keys

```
my @keys = hidden_keys(%hash);
```

Returns the list of the keys that are legal in a restricted hash but do not have a value associated to them. Thus if 'foo' is a "hidden" key of the %hash it will return false for both `defined` and `exists` tests.

In the case of an unrestricted hash this will return an empty list.

NOTE this is an experimental feature that is heavily dependent on the current implementation of restricted hashes. Should the implementation change, this routine may become meaningless, in which case it will return an empty list.

all_keys

```
all_keys(%hash,@keys,@hidden);
```

Populates the arrays @keys with the all the keys that would pass an `exists` tests, and populates @hidden with the remaining legal keys that have not been utilized.

Returns a reference to the hash.

In the case of an unrestricted hash this will be equivalent to

```
$ref = do {
    @keys = keys %hash;
    @hidden = ();
    \%hash
};
```

NOTE this is an experimental feature that is heavily dependent on the current implementation of restricted hashes. Should the implementation change this routine may become meaningless

in which case it will behave identically to how it would behave on an unrestricted hash.

hash_seed

```
my $hash_seed = hash_seed();
```

hash_seed() returns the seed number used to randomise hash ordering. Zero means the "traditional" random hash ordering, non-zero means the new even more random hash ordering introduced in Perl 5.8.1.

Note that the hash seed is sensitive information: by knowing it one can craft a denial-of-service attack against Perl code, even remotely, see *"Algorithmic Complexity Attacks" in perlsec* for more information. **Do not disclose the hash seed** to people who don't need to know it. See also *"PERL_HASH_SEED_DEBUG" in perlrun*.

hv_store

```
my $sv = 0;
hv_store(%hash,$key,$sv) or die "Failed to alias!";
$hash{$key} = 1;
print $sv; # prints 1
```

Stores an alias to a variable in a hash instead of copying the value.

Operating on references to hashes.

Most subroutines documented in this module have equivalent versions that operate on references to hashes instead of native hashes. The following is a list of these subs. They are identical except in name and in that instead of taking a %hash they take a $hashref, and additionally are not prototyped.

lock_ref_keys

unlock_ref_keys

lock_ref_keys_plus

lock_ref_value

unlock_ref_value

lock_hashref

unlock_hashref

lock_hashref_recurse

unlock_hashref_recurse

hash_ref_unlocked

legal_ref_keys

hidden_ref_keys

CAVEATS

Note that the trapping of the restricted operations is not atomic: for example

```
eval { %hash = (illegal_key => 1) }
```

leaves the `%hash` empty rather than with its original contents.

BUGS

The interface exposed by this module is very close to the current implementation of restricted hashes. Over time it is expected that this behavior will be extended and the interface abstracted further.

AUTHOR

Michael G Schwern <schwern@pobox.com> on top of code by Nick Ing-Simmons and Jeffrey Friedl.

hv_store() is from Array::RefElem, Copyright 2000 Gisle Aas.

Additional code by Yves Orton.

SEE ALSO

Scalar::Util, *List::Util* and *"Algorithmic Complexity Attacks" in perlsec*.

Hash::Util::FieldHash.