NAME

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

SYNOPSIS

```
# Restricted hashes

use Hash::Util qw(
fieldhash fieldhashes

all_keys
lock_keys unlock_keys
lock_value unlock_value
lock_hash unlock_hash
lock_keys_plus
hash_locked hash_unlocked
hashref_locked hashref_unlocked
hidden_keys legal_keys

lock_ref_keys unlock_ref_keys
lock_ref_value unlock_ref_value
lock_hashref unlock_hashref
lock_ref_keys_plus
hidden_ref_keys legal_ref_keys

hash_seed hash_value hv_store
bucket_stats bucket_info bucket_array
lock_hash_recurse unlock_hash_recurse

hash_traversal_mask
);

%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;
```

```
my $int_hash_value = hash_value( 'string' );

my $mask= hash_traversal_mask(%hash);

hash_traversal_mask(%hash,1234);
```

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.

This method **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`.

hashref_locked
hash_locked

```
hashref_locked(\%hash) and print "Hash is locked!\n";
hash_locked(%hash) and print "Hash is locked!\n";
```

Returns true if the hash and its keys are locked.

hashref_unlocked
hash_unlocked

```
hashref_unlocked(\%hash) and print "Hash is unlocked!\n";
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 bytes used to randomise hash ordering.

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(1) 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.

Prior to Perl 5.17.6 this function returned a UV, it now returns a string, which may be of nearly any size as determined by the hash function your Perl has been built with. Possible sizes may be but are not limited to 4 bytes (for most hash algorithms) and 16 bytes (for siphash).

hash_value

```
my $hash_value = hash_value($string);
```

hash_value() returns the current perl's internal hash value for a given string.

Returns a 32 bit integer representing the hash value of the string passed in. This value is only reliable for the lifetime of the process. It may be different depending on invocation, environment variables, perl version, architectures, and build options.

Note that the hash value of a given string is sensitive information: by knowing it one can deduce the hash seed which in turn can allow one to craft a denial-of-service attack against Perl code, even remotely, see ''Algorithmic Complexity Attacks'' in perlsec(1) for more information. **Do not disclose the hash value of a string** to people who don't need to know it. See also ''PERL_HASH_SEED_DEBUG'' in perlrun.

bucket_info

Return a set of basic information about a hash.

```
my ($keys, $buckets, $used, @length_counts)= bucket_info($hash);
```

Fields are as follows:

```
0: Number of keys in the hash
1: Number of buckets in the hash
2: Number of used buckets in the hash
rest : list of counts, Kth element is the number of buckets
with K keys in it.
```

See also *bucket_stats()* and *bucket_array()*.

bucket_stats

Returns a list of statistics about a hash.

```
my ($keys, buckets, $used, $utilization_ratio, $collision_pct,
$mean, $stddev, @length_counts) = bucket_info($hashref);
```

Fields are as follows:

```
0: Number of keys in the hash
1: Number of buckets in the hash
2: Number of used buckets in the hash
3: Hash Quality Score
4: Percent of buckets used
5: Percent of keys which are in collision
6: Average bucket length
7: Standard Deviation of bucket lengths.
rest : list of counts, Kth element is the number of buckets
with K keys in it.
```

See also *bucket_info()* and *bucket_array()*.

Note that Hash Quality Score would be 1 for an ideal hash, numbers close to and below 1 indicate good hashing, and number significantly above indicate a poor score. In practice it should be around 0.95 to 1.05. It is defined as:

```
$score= sum( $count[$length] * ($length * ($length + 1) / 2) )
/
( ( $keys / 2 * $buckets ) *
( $keys + ( 2 * $buckets ) - 1 ) )
```

The formula is from the Red Dragon book (reformulated to use the data available) and is documented at <http://www.strchr.com/hash_functions>

bucket_array

```
my $array= bucket_array(\%hash);
```

Returns a packed representation of the bucket array associated with a hash. Each element of the array is either an integer K, in which case it represents K empty buckets, or a reference to another array which contains the keys that are in that bucket.

Note that the information returned by bucket_array is sensitive information: by knowing it one can directly attack perl's hash function which in turn may allow one to craft a denial-of-service attack against Perl code, even remotely, see ''Algorithmic Complexity Attacks'' in perlsec(1) for more information. **Do not disclose the output of this function** to people who don't need to know it. See also ''PERL_HASH_SEED_DEBUG'' in perlrun. This function is provided strictly for debugging and diagnostics purposes only, it is hard to imagine a reason why it would be used in production code.

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.

hash_traversal_mask

As of Perl 5.18 every hash has its own hash traversal order, and this order changes every time a new element is inserted into the hash. This functionality is provided by maintaining an unsigned integer mask (U32) which is xor'ed with the actual bucket id during a traversal of the hash buckets using *keys()*, *values()* or *each()*.

You can use this subroutine to get and set the traversal mask for a specific hash. Setting the mask ensures that a given hash will produce the same key order. **Note** that this does **not** guarantee that **two** hashes will produce the same key order for the same hash seed and traversal mask, items that collide into one bucket may have different orders regardless of this setting.

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.