

## NAME

Tie::Memoize - add data to hash when needed

## SYNOPSIS

```
require Tie::Memoize;
tie %hash, 'Tie::Memoize',
    \&fetch,    # The rest is optional
    $DATA, \&exists,
    {%ini_value}, {%ini_existence};
```

## DESCRIPTION

This package allows a tied hash to autoload its values on the first access, and to use the cached value on the following accesses.

Only read-accesses (via fetching the value or `exists`) result in calls to the functions; the modify-accesses are performed as on a normal hash.

The required arguments during `tie` are the hash, the package, and the reference to the `FETCHing` function. The optional arguments are an arbitrary scalar `$data`, the reference to the `EXISTS` function, and initial values of the hash and of the existence cache.

Both the `FETCHing` function and the `EXISTS` functions have the same signature: the arguments are `$key`, `$data`; `$data` is the same value as given as argument during `tie()`ing. Both functions should return an empty list if the value does not exist. If `EXISTS` function is different from the `FETCHing` function, it should return a `TRUE` value on success. The `FETCHing` function should return the intended value if the key is valid.

## Inheriting from Tie::Memoize

The structure of the `tied()` data is an array reference with elements

```
0:  cache of known values
1:  cache of known existence of keys
2:  FETCH function
3:  EXISTS function
4:  $data
```

The rest is for internal usage of this package. In particular, if `TIEHASH` is overwritten, it should call `SUPER::TIEHASH`.

## EXAMPLE

```
sub slurp {
    my ($key, $dir) = shift;
    open my $h, '<', "$dir/$key" or return;
    local $/; <$h>    # slurp it all
}
sub exists { my ($key, $dir) = shift; return -f "$dir/$key" }
```

  

```
tie %hash, 'Tie::Memoize', \&slurp, $directory, \&exists,
    { fake_file1 => $content1, fake_file2 => $content2 },
    { pretend_does_not_exists => 0, known_to_exist => 1 };
```

This example treats the slightly modified contents of `$directory` as a hash. The modifications are that the keys `fake_file1` and `fake_file2` fetch values `$content1` and `$content2`, and `pretend_does_not_exists` will never be accessed. Additionally, the existence of `known_to_exist` is never checked (so if it does not exist when its content is needed, the user of `%hash` may be

**BUGS**   confused).

FIRSTKEY and NEXTKEY methods go through the keys which were already read, not all the possible keys of the hash.

**AUTHOR**

Ilya Zakharevich *mailto:perl-module-hash-memoize@ilyaz.org*.