NAME

perldbmfilter - Perl DBM Filters

SYNOPSIS

```
$db = tie %hash, 'DBM', ...

$old_filter = $db->filter_store_key  ( sub { ... } );
$old_filter = $db->filter_store_value( sub { ... } );
$old_filter = $db->filter_fetch_key  ( sub { ... } );
$old_filter = $db->filter_fetch_value( sub { ... } );
```

DESCRIPTION

The four "filter_*" methods shown above are available in all the DBM modules that ship with Perl, namely DB_File, GDBM_File, NDBM_File, ODBM_File and SDBM_File.

Each of the methods works identically, and is used to install (or uninstall) a single DBM Filter. The only difference between them is the place that the filter is installed.

To summarise:

filter_store_key

If a filter has been installed with this method, it will be invoked every time you write a key to a DBM database.

filter_store_value

If a filter has been installed with this method, it will be invoked every time you write a value to a DBM database.

filter_fetch_key

If a filter has been installed with this method, it will be invoked every time you read a key from a DBM database.

filter_fetch_value

If a filter has been installed with this method, it will be invoked every time you read a value from a DBM database.

You can use any combination of the methods from none to all four.

All filter methods return the existing filter, if present, or "undef" if not.

To delete a filter pass "undef" to it.

The Filter

When each filter is called by Perl, a local copy of $_ will contain the key or value to be filtered. Filtering is achieved by modifying the contents of $_. The return code from the filter is ignored.

An Example: the NULL termination problem.

DBM Filters are useful for a class of problems where you *always* want to make the same transformation to all keys, all values or both.

For example, consider the following scenario. You have a DBM database that you need to share with a third-party C application. The C application assumes that *all* keys and values are NULL terminated. Unfortunately when Perl writes to DBM databases it doesn't use NULL termination, so your Perl application will have to manage NULL termination itself. When you write to the database you will have to use something like this:

```
$hash{"$key\0"} = "$value\0";
```

Similarly the NULL needs to be taken into account when you are considering the length of existing keys/values.

It would be much better if you could ignore the NULL terminations issue in the main application code and have a mechanism that automatically added the terminating NULL to all keys and values whenever you write to the database and have them removed when you read from the database. As I'm sure you have already guessed, this is a problem that DBM Filters can fix very easily.

```
use strict;
use warnings;
use SDBM_File;
use Fcntl;

my %hash;
my $filename = "filt";
unlink $filename;

my $db = tie(%hash, 'SDBM_File', $filename, O_RDWR|O_CREAT, 0640)
  or die "Cannot open $filename: $!\n";

# Install DBM Filters
$db->filter_fetch_key  ( sub { s/\0$//    } );
```

```
$db->filter_store_key  ( sub { $_ .= "\0" } );
$db->filter_fetch_value(
   sub { no warnings 'uninitialized'; s/\0$// } );
$db->filter_store_value( sub { $_ .= "\0" } );

$hash{"abc"} = "def";
my $a = $hash{"ABC"};
# ...
undef $db;
untie %hash;
```

The code above uses SDBM_File, but it will work with any of the DBM modules.

Hopefully the contents of each of the filters should be self-explanatory. Both "fetch" filters remove the terminating NULL, and both "store" filters add a terminating NULL.

Another Example: Key is a C int.

Here is another real-life example. By default, whenever Perl writes to a DBM database it always writes the key and value as strings. So when you use this:

```
$hash{12345} = "something";
```

the key 12345 will get stored in the DBM database as the 5 byte string "12345". If you actually want the key to be stored in the DBM database as a C int, you will have to use "pack" when writing, and "unpack" when reading.

Here is a DBM Filter that does it:

```
use strict;
use warnings;
use DB_File;
my %hash;
my $filename = "filt";
unlink $filename;


my $db = tie %hash, 'DB_File', $filename, O_CREAT|O_RDWR, 0666,
   $DB_HASH or die "Cannot open $filename: $!\n";

$db->filter_fetch_key  ( sub { $_ = unpack("i", $_) } );
```

```
$db->filter_store_key  ( sub { $_ = pack ("i", $_) } );
$hash{123} = "def";
# ...
undef $db;
untie %hash;
```

The code above uses DB_File, but again it will work with any of the DBM modules.

This time only two filters have been used; we only need to manipulate the contents of the key, so it wasn't necessary to install any value filters.

SEE ALSO

DB_File, GDBM_File, NDBM_File, ODBM_File and SDBM_File.

AUTHOR

Paul Marquess