NAME

Tie::Array - base class for tied arrays

SYNOPSIS

```
package Tie::NewArray;
use Tie::Array;
@ISA = ('Tie::Array');

# mandatory methods
sub TIEARRAY { ... }
sub FETCH { ... }
sub FETCHSIZE { ... }

sub STORE { ... }        # mandatory if elements writeable
sub STORESIZE { ... }    # mandatory if elements can be added/deleted
sub EXISTS { ... }       # mandatory if exists() expected to work
sub DELETE { ... }       # mandatory if delete() expected to work

# optional methods - for efficiency
sub CLEAR { ... }
sub PUSH { ... }
sub POP { ... }
sub SHIFT { ... }
sub UNSHIFT { ... }
sub SPLICE { ... }
sub EXTEND { ... }
sub DESTROY { ... }

package Tie::NewStdArray;
use Tie::Array;

@ISA = ('Tie::StdArray');

# all methods provided by default

package main;

$object = tie @somearray,Tie::NewArray;
$object = tie @somearray,Tie::StdArray;
$object = tie @somearray,Tie::NewStdArray;
```

DESCRIPTION

This module provides methods for array-tying classes. See *perltie* for a list of the functions required in order to tie an array to a package. The basic **Tie::Array** package provides stub `DESTROY`, and `EXTEND` methods that do nothing, stub `DELETE` and `EXISTS` methods that croak() if the delete() or exists() builtins are ever called on the tied array, and implementations of `PUSH`, `POP`, `SHIFT`, `UNSHIFT`, `SPLICE` and `CLEAR` in terms of basic `FETCH`, `STORE`, `FETCHSIZE`, `STORESIZE`.

The **Tie::StdArray** package provides efficient methods required for tied arrays which are implemented as blessed references to an "inner" perl array. It inherits from **Tie::Array**, and should cause tied arrays to behave exactly like standard arrays, allowing for selective overloading of methods.

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

TIEARRAY classname, LIST

The class method is invoked by the command `tie @array, classname`. Associates an array instance with the specified class. `LIST` would represent additional arguments (along the lines of *AnyDBM_File* and compatriots) needed to complete the association. The method should return an object of a class which provides the methods below.

STORE this, index, value

Store datum *value* into *index* for the tied array associated with object *this*. If this makes the array larger then class's mapping of `undef` should be returned for new positions.

FETCH this, index

Retrieve the datum in *index* for the tied array associated with object *this*.

FETCHSIZE this

Returns the total number of items in the tied array associated with object *this*. (Equivalent to `scalar(@array)`).

STORESIZE this, count

Sets the total number of items in the tied array associated with object *this* to be *count*. If this makes the array larger then class's mapping of `undef` should be returned for new positions. If the array becomes smaller then entries beyond count should be deleted.

EXTEND this, count

Informative call that array is likely to grow to have *count* entries. Can be used to optimize allocation. This method need do nothing.

EXISTS this, key

Verify that the element at index *key* exists in the tied array *this*.

The **Tie::Array** implementation is a stub that simply croaks.

DELETE this, key

Delete the element at index *key* from the tied array *this*.

The **Tie::Array** implementation is a stub that simply croaks.

CLEAR this

Clear (remove, delete, ...) all values from the tied array associated with object *this*.

DESTROY this

Normal object destructor method.

PUSH this, LIST

Append elements of LIST to the array.

POP this

Remove last element of the array and return it.

SHIFT this

Remove the first element of the array (shifting other elements down) and return it.

UNSHIFT this, LIST

Insert LIST elements at the beginning of the array, moving existing elements up to make room.

SPLICE this, offset, length, LIST

Perform the equivalent of `splice` on the array.

offset is optional and defaults to zero, negative values count back from the end of the array.

length is optional and defaults to rest of the array.

LIST may be empty.

Returns a list of the original *length* elements at *offset*.

CAVEATS

There is no support at present for tied @ISA. There is a potential conflict between magic entries needed to notice setting of @ISA, and those needed to implement 'tie'.

Very little consideration has been given to the behaviour of tied arrays when `$[` is not default value of zero.

AUTHOR

Nick Ing-Simmons <nik@tiuk.ti.com>