NAME

Tie::Scalar, Tie::StdScalar - base class definitions for tied scalars

SYNOPSIS

```
    package NewScalar;
    require Tie::Scalar;

    @ISA = qw(Tie::Scalar);

    sub FETCH { ... }  # Provide a needed method
    sub TIESCALAR { ... } # Overrides inherited method

    package NewStdScalar;
    require Tie::Scalar;

    @ISA = qw(Tie::StdScalar);

    # All methods provided by default, so define only what needs be
overridden
    sub FETCH { ... }

    package main;

    tie $new_scalar, 'NewScalar';
    tie $new_std_scalar, 'NewStdScalar';
```

DESCRIPTION

This module provides some skeletal methods for scalar-tying classes. See *perltie* for a list of the functions required in tying a scalar to a package. The basic **Tie::Scalar** package provides a `new` method, as well as methods `TIESCALAR`, `FETCH` and `STORE`. The **Tie::StdScalar** package provides all the methods specified in *perltie*. It inherits from **Tie::Scalar** and causes scalars tied to it to behave exactly like the built-in scalars, allowing for selective overloading of methods. The `new` method is provided as a means of grandfathering, for classes that forget to provide their own `TIESCALAR` method.

For developers wishing to write their own tied-scalar classes, the methods are summarized below. The *perltie* section not only documents these, but has sample code as well:

TIESCALAR classname, LIST

> The method invoked by the command `tie $scalar, classname`. Associates a new scalar instance with the specified class. `LIST` would represent additional arguments (along the lines of *AnyDBM_File* and compatriots) needed to complete the association.

FETCH this

> Retrieve the value of the tied scalar referenced by *this*.

STORE this, value

> Store data *value* in the tied scalar referenced by *this*.

DESTROY this

> Free the storage associated with the tied scalar referenced by *this*. This is rarely needed, as Perl manages its memory quite well. But the option exists, should a class wish to perform specific actions upon the destruction of an instance.

Tie::Scalar vs Tie::StdScalar

`Tie::Scalar` provides all the necessary methods, but one should realize they do not do anything useful. Calling `Tie::Scalar::FETCH` or `Tie::Scalar::STORE` results in a (trappable) croak. And if you inherit from `Tie::Scalar`, you *must* provide either a `new` or a `TIESCALAR` method.

If you are looking for a class that does everything for you you don't define yourself, use the `Tie::StdScalar` class, not the `Tie::Scalar` one.

MORE INFORMATION

The *perltie* section uses a good example of tying scalars by associating process IDs with priority.