NAME

fields - compile-time class fields

SYNOPSIS

```
{
    package Foo;
    use fields qw(foo bar _Foo_private);
    sub new {
        my Foo $self = shift;
        unless (ref $self) {
            $self = fields::new($self);
            $self->{_Foo_private} = "this is Foo's secret";
        }
        $self->{foo} = 10;
        $self->{bar} = 20;
        return $self;
    }
}

my $var = Foo->new;
$var->{foo} = 42;

# this will generate an error
$var->{zap} = 42;

# subclassing
{
    package Bar;
    use base 'Foo';
    use fields qw(baz _Bar_private);        # not shared with Foo
    sub new {
        my $class = shift;
        my $self = fields::new($class);
        $self->SUPER::new();                # init base fields
        $self->{baz} = 10;                  # init own fields
        $self->{_Bar_private} = "this is Bar's secret";
        return $self;
    }
}
```

DESCRIPTION

The `fields` pragma enables compile-time verified class fields.

NOTE: The current implementation keeps the declared fields in the %FIELDS hash of the calling package, but this may change in future versions. Do **not** update the %FIELDS hash directly, because it must be created at compile-time for it to be fully useful, as is done by this pragma.

Only valid for perl before 5.9.0:

If a typed lexical variable holding a reference is used to access a hash element and a package with the same name as the type has declared class fields using this pragma, then the operation is turned into an array access at compile time.

The related `base` pragma will combine fields from base classes and any fields declared using the `fields` pragma. This enables field inheritance to work properly.

Field names that start with an underscore character are made private to the class and are not visible to subclasses. Inherited fields can be overridden but will generate a warning if used together with the `-w` switch.

Only valid for perls before 5.9.0:

The effect of all this is that you can have objects with named fields which are as compact and as fast arrays to access. This only works as long as the objects are accessed through properly typed variables. If the objects are not typed, access is only checked at run time.

The following functions are supported:

new

perl before 5.9.0: fields::new() creates and blesses a pseudo-hash comprised of the fields declared using the `fields` pragma into the specified class.

perl 5.9.0 and higher: fields::new() creates and blesses a restricted-hash comprised of the fields declared using the `fields` pragma into the specified class.

This function is usable with or without pseudo-hashes. It is the recommended way to construct a fields-based object.

This makes it possible to write a constructor like this:

```
    package Critter::Sounds;
    use fields qw(cat dog bird);

    sub new {
        my $self = shift;
        $self = fields::new($self) unless ref $self;
        $self->{cat} = 'meow';                          # scalar
element
        @$self{'dog','bird'} = ('bark','tweet');        # slice
        return $self;
    }
```

phash

before perl 5.9.0:

fields::phash() can be used to create and initialize a plain (unblessed) pseudo-hash. This function should always be used instead of creating pseudo-hashes directly.

If the first argument is a reference to an array, the pseudo-hash will be created with keys from that array. If a second argument is supplied, it must also be a reference to an array whose elements will be used as the values. If the second array contains less elements than the first, the trailing elements of the pseudo-hash will not be initialized. This makes it particularly useful for creating a pseudo-hash from subroutine arguments:

```
    sub dogtag {
       my $tag = fields::phash([qw(name rank ser_num)], [@_]);
    }
```

fields::phash() also accepts a list of key-value pairs that will be used to construct the pseudo hash. Examples:

```
    my $tag = fields::phash(name => "Joe",
                            rank => "captain",
                            ser_num => 42);

    my $pseudohash = fields::phash(%args);
```

perl 5.9.0 and higher:

Pseudo-hashes have been removed from Perl as of 5.10. Consider using restricted hashes or fields::new() instead. Using fields::phash() will cause an error.

SEE ALSO

base