NAME

Class::Struct - declare struct-like datatypes as Perl classes

SYNOPSIS

```
use Class::Struct;
        # declare struct, based on array:
struct( CLASS_NAME => [ ELEMENT_NAME => ELEMENT_TYPE, ... ]);
        # declare struct, based on hash:
struct( CLASS_NAME => { ELEMENT_NAME => ELEMENT_TYPE, ... });

package CLASS_NAME;
use Class::Struct;
        # declare struct, based on array, implicit class name:
struct( ELEMENT_NAME => ELEMENT_TYPE, ... );

# Declare struct at compile time
use Class::Struct CLASS_NAME => [ ELEMENT_NAME => ELEMENT_TYPE, ... ];
use Class::Struct CLASS_NAME => { ELEMENT_NAME => ELEMENT_TYPE, ... };

# declare struct at compile time, based on array, implicit class name:
package CLASS_NAME;
use Class::Struct ELEMENT_NAME => ELEMENT_TYPE, ... ;

package Myobj;
use Class::Struct;
        # declare struct with four types of elements:
struct( s => '$', a => '@', h => '%', c => 'My_Other_Class' );

$obj = new Myobj;               # constructor

                                # scalar type accessor:
$element_value = $obj->s;           # element value
$obj->s('new value');               # assign to element

                                # array type accessor:
$ary_ref = $obj->a;                 # reference to whole array
$ary_element_value = $obj->a(2);    # array element value
$obj->a(2, 'new value');            # assign to array element

                                # hash type accessor:
$hash_ref = $obj->h;                # reference to whole hash
$hash_element_value = $obj->h('x'); # hash element value
$obj->h('x', 'new value');          # assign to hash element

                                # class type accessor:
$element_value = $obj->c;           # object reference
$obj->c->method(...);               # call method of object
$obj->c(new My_Other_Class);        # assign a new object
```

DESCRIPTION

`Class::Struct` exports a single function, `struct`. Given a list of element names and types, and optionally a class name, `struct` creates a Perl 5 class that implements a "struct-like" data structure.

The new class is given a constructor method, `new`, for creating struct objects.

Each element in the struct data has an accessor method, which is used to assign to the element and to fetch its value. The default accessor can be overridden by declaring a `sub` of the same name in the package. (See Example 2.)

Each element's type can be scalar, array, hash, or class.

The struct() function

The `struct` function has three forms of parameter-list.

```
struct( CLASS_NAME => [ ELEMENT_LIST ]);
struct( CLASS_NAME => { ELEMENT_LIST });
struct( ELEMENT_LIST );
```

The first and second forms explicitly identify the name of the class being created. The third form assumes the current package name as the class name.

An object of a class created by the first and third forms is based on an array, whereas an object of a class created by the second form is based on a hash. The array-based forms will be somewhat faster and smaller; the hash-based forms are more flexible.

The class created by `struct` must not be a subclass of another class other than `UNIVERSAL`.

It can, however, be used as a superclass for other classes. To facilitate this, the generated constructor method uses a two-argument blessing. Furthermore, if the class is hash-based, the key of each element is prefixed with the class name (see *Perl Cookbook*, Recipe 13.12).

A function named `new` must not be explicitly defined in a class created by `struct`.

The *ELEMENT_LIST* has the form

```
NAME => TYPE, ...
```

Each name-type pair declares one element of the struct. Each element name will be defined as an accessor method unless a method by that name is explicitly defined; in the latter case, a warning is issued if the warning flag (**-w**) is set.

Class Creation at Compile Time

`Class::Struct` can create your class at compile time. The main reason for doing this is obvious, so your class acts like every other class in Perl. Creating your class at compile time will make the order of events similar to using any other class (or Perl module).

There is no significant speed gain between compile time and run time class creation, there is just a new, more standard order of events.

Element Types and Accessor Methods

The four element types -- scalar, array, hash, and class -- are represented by strings -- `'$'`, `'@'`, `'%'` , and a class name -- optionally preceded by a `'*'`.

The accessor method provided by `struct` for an element depends on the declared type of the element.

Scalar (`'$'` or `'*$'`)

The element is a scalar, and by default is initialized to `undef` (but see *Initializing with new*).

The accessor's argument, if any, is assigned to the element.

If the element type is `'$'`, the value of the element (after assignment) is returned. If the element type is `'*$'`, a reference to the element is returned.

Array (`'@'` or `'*@'`)

The element is an array, initialized by default to `()`.

With no argument, the accessor returns a reference to the element's whole array (whether or not the element was specified as `'@'` or `'*@'`).

With one or two arguments, the first argument is an index specifying one element of the array; the second argument, if present, is assigned to the array element. If the element type is `'@'`, the accessor returns the array element value. If the element type is `'*@'`, a reference to the array element is returned.

As a special case, when the accessor is called with an array reference as the sole argument, this causes an assignment of the whole array element. The object reference is returned.

Hash (`'%'` or `'*%'`)

The element is a hash, initialized by default to `()`.

With no argument, the accessor returns a reference to the element's whole hash (whether or not the element was specified as `'%'` or `'*%'`).

With one or two arguments, the first argument is a key specifying one element of the hash; the second argument, if present, is assigned to the hash element. If the element type is `'%'`, the accessor returns the hash element value. If the element type is `'*%'`, a reference to the hash element is returned.

As a special case, when the accessor is called with a hash reference as the sole argument, this causes an assignment of the whole hash element. The object reference is returned.

Class (`'Class_Name'` or `'*Class_Name'`)

The element's value must be a reference blessed to the named class or to one of its subclasses. The element is not initialized by default.

The accessor's argument, if any, is assigned to the element. The accessor will `croak` if this is not an appropriate object reference.

If the element type does not start with a `'*'`, the accessor returns the element value (after assignment). If the element type starts with a `'*'`, a reference to the element itself is returned.

Initializing with new

`struct` always creates a constructor called `new`. That constructor may take a list of initializers for the various elements of the new struct.

Each initializer is a pair of values: *element name* `=>` *value*. The initializer value for a scalar element is just a scalar value. The initializer for an array element is an array reference. The initializer for a hash is a hash reference.

The initializer for a class element is an object of the corresponding class, or of one of it's subclasses, or a reference to a hash containing named arguments to be passed to the element's constructor.

See Example 3 below for an example of initialization.

EXAMPLES

Example 1

Giving a struct element a class type that is also a struct is how structs are nested. Here, `Timeval` represents a time (seconds and microseconds), and `Rusage` has two elements, each of which is of type `Timeval`.

```
use Class::Struct;

struct( Rusage => {
    ru_utime => 'Timeval',  # user time used
    ru_stime => 'Timeval',  # system time used
```

```
    });

    struct( Timeval => [
        tv_secs  => '$',        # seconds
        tv_usecs => '$',        # microseconds
    ]);

        # create an object:
    my $t = Rusage->new(ru_utime=>Timeval->new(),
ru_stime=>Timeval->new());

        # $t->ru_utime and $t->ru_stime are objects of type Timeval.
        # set $t->ru_utime to 100.0 sec and $t->ru_stime to 5.0 sec.
    $t->ru_utime->tv_secs(100);
    $t->ru_utime->tv_usecs(0);
    $t->ru_stime->tv_secs(5);
    $t->ru_stime->tv_usecs(0);
```

Example 2

An accessor function can be redefined in order to provide additional checking of values, etc. Here, we want the `count` element always to be nonnegative, so we redefine the `count` accessor accordingly.

```
    package MyObj;
    use Class::Struct;

    # declare the struct
    struct ( 'MyObj', { count => '$', stuff => '%' } );

    # override the default accessor method for 'count'
    sub count {
        my $self = shift;
        if ( @_ ) {
            die 'count must be nonnegative' if $_[0] < 0;
            $self->{'MyObj::count'} = shift;
            warn "Too many args to count" if @_;
        }
        return $self->{'MyObj::count'};
    }

    package main;
    $x = new MyObj;
    print "\$x->count(5) = ", $x->count(5), "\n";
                            # prints '$x->count(5) = 5'

    print "\$x->count = ", $x->count, "\n";
                            # prints '$x->count = 5'

    print "\$x->count(-5) = ", $x->count(-5), "\n";
                            # dies due to negative argument!
```

Example 3

The constructor of a generated class can be passed a list of *element=>value* pairs, with which to initialize the struct. If no initializer is specified for a particular element, its default initialization is performed instead. Initializers for non-existent elements are silently ignored.

Note that the initializer for a nested class may be specified as an object of that class, or as a reference to a hash of initializers that are passed on to the nested struct's constructor.

```
    use Class::Struct;

    struct Breed =>
    {
        name  => '$',
        cross => '$',
    };

    struct Cat =>
    [
        name     => '$',
        kittens  => '@',
        markings => '%',
        breed    => 'Breed',
    ];

    my $cat = Cat->new( name     => 'Socks',
                        kittens  => ['Monica', 'Kenneth'],
                        markings => { socks=>1, blaze=>"white" },
                        breed    => Breed->new(name=>'short-hair',
cross=>1),
                      or:  breed    => {name=>'short-hair', cross=>1},
                         );

    print "Once a cat called ", $cat->name, "\n";
    print "(which was a ", $cat->breed->name, ")\n";
    print "had two kittens: ", join(' and ', @{$cat->kittens}), "\n";
```

Author and Modification History

Modified by Damian Conway, 2001-09-10, v0.62.

```
   Modified implicit construction of nested objects.
   Now will also take an object ref instead of requiring a hash ref.
   Also default initializes nested object attributes to undef, rather
   than calling object constructor without args
   Original over-helpfulness was fraught with problems:
       * the class's constructor might not be called 'new'
       * the class might not have a hash-like-arguments constructor
       * the class might not have a no-argument constructor
       * "recursive" data structures didn't work well:
                package Person;
                struct { mother => 'Person', father => 'Person'};
```

Modified by Casey West, 2000-11-08, v0.59.

```
    Added the ability for compile time class creation.
```

Modified by Damian Conway, 1999-03-05, v0.58.

```
    Added handling of hash-like arg list to class ctor.

    Changed to two-argument blessing in ctor to support
    derivation from created classes.
```

```
    Added classname prefixes to keys in hash-based classes
    (refer to "Perl Cookbook", Recipe 13.12 for rationale).

    Corrected behaviour of accessors for '*@' and '*%' struct
    elements.  Package now implements documented behaviour when
    returning a reference to an entire hash or array element.
    Previously these were returned as a reference to a reference
    to the element.
```

Renamed to `Class::Struct` and modified by Jim Miner, 1997-04-02.

```
    members() function removed.
    Documentation corrected and extended.
    Use of struct() in a subclass prohibited.
    User definition of accessor allowed.
    Treatment of '*' in element types corrected.
    Treatment of classes as element types corrected.
    Class name to struct() made optional.
    Diagnostic checks added.
```

Originally `Class::Template` by Dean Roehrich.

```
    # Template.pm   --- struct/member template builder
    #   12mar95
    #   Dean Roehrich
    #
    # changes/bugs fixed since 28nov94 version:
    #  - podified
    # changes/bugs fixed since 21nov94 version:
    #  - Fixed examples.
    # changes/bugs fixed since 02sep94 version:
    #  - Moved to Class::Template.
    # changes/bugs fixed since 20feb94 version:
    #  - Updated to be a more proper module.
    #  - Added "use strict".
    #  - Bug in build_methods, was using @var when @$var needed.
    #  - Now using my() rather than local().
    #
    # Uses perl5 classes to create nested data types.
    # This is offered as one implementation of Tom Christiansen's
"structs.pl"
    # idea.
```