Class Struct < Object

Subclasses: Struct::Tms

A Struct is a convenient way to bundle a number of attributes together, using accessor methods, without having to write an explicit class.

The Struct class is a generator of specific classes, each one of which is defined to hold a set of variables and their accessors. In these examples, we'll call the generated class *Customer*, and we'll show an example instance of that class as *joe*.

Also see OpenStruct on page 787.

In the descriptions that follow, the parameter *symbol* refers to a symbol, which is either a quoted string or a Symbol (such as :name).

Mixes in

Enumerable:

`all?`, `any?`, `collect`, `count`, `cycle`, `detect`, `drop`, `drop_while`, `each_cons`, `each_slice`, `each_with_index`, `entries`, `find`, `find_all`, `find_index`, `first`, `grep`, `group_by`, `include?`, `inject`, `map`, `max`, `max_by`, `member?`, `min`, `min_by`, `minmax`, `minmax_by`, `none?`, `one?`, `partition`, `reduce`, `reject`, `select`, `sort`, `sort_by`, `take`, `take_while`, `to_a`, `zip`

Class methods

new

Struct.new(⟨ *string* ⟩ ⟨ , *symbol* $\rangle^{+}$) → *Customer*
Struct.new(⟨ *string* ⟩ ⟨ , *symbol* $\rangle^{+}$) { *block* } → *Customer*

Creates a new class, named by *string*, containing accessor methods for the given symbols. If the name *string* is omitted, an anonymous structure class will be created. Otherwise, the name of this struct will appear as a constant in class Struct, so it must be unique for all Structs in the system and should start with a capital letter. Assigning a structure class to a constant effectively gives the class the name of the constant.

Struct.new returns a new Class object, which can then be used to create specific instances of the new structure. The remaining methods listed next (class and instance) are defined for this generated class. See the description that follows for an example.

```
# Create a structure with a name in Struct
Struct.new("Customer", :name, :address)    # =>   Struct::Customer
Struct::Customer.new("Dave", "123 Main")   # =>   #<struct
                                                  Struct::Customer
                                                  name="Dave",
                                                  address="123 Main">

# Create a structure named by its constant
Customer = Struct.new(:name, :address)   # =>   Customer
Customer.new("Dave", "123 Main")         # =>   #<struct Customer
                                                name="Dave", address="123
                                                Main">
```

A block passed to the constructor is evaluated in the context of the new struct's class and hence allows you conveniently to add instance methods to the new struct.

```
Customer = Struct.new(:name, :address) do
  def to_s
    "#{self.name} lives at #{self.address}"
  end
end
Customer.new("Dave", "123 Main").to_s   # =>   "Dave lives at 123 Main"
```

new *Customer*.new(⟨ *obj* ⟩$^+$) → *joe*

Creates a new instance of a structure (the class created by Struct.new). The number of actual parameters must be less than or equal to the number of attributes defined for this class; unset parameters default to nil. Passing too many parameters will raise an ArgumentError.

```
Customer = Struct.new(:name, :address, :zip)

joe = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)
joe.name   # =>   "Joe Smith"
joe.zip    # =>   12345
```

[ ] *Customer*[⟨ *obj* ⟩$^+$] → *joe*

Synonym for new (for the generated structure).

```
Customer = Struct.new(:name, :address, :zip)

joe = Customer["Joe Smith", "123 Maple, Anytown NC", 12345]
joe.name   # =>   "Joe Smith"
joe.zip    # =>   12345
```

members *Customer*.members → *array*

1.9 Returns an array of symbols representing the names of the instance variables.

```
Customer = Struct.new("Customer", :name, :address, :zip)
Customer.members   # =>   [:name, :address, :zip]
```

Instance methods

== *joe* == *other_struct* → true or false

Equality—Returns true if *other_struct* is equal to this one: they must be of the same class as generated by Struct.new, and the values of all instance variables must be equal (according to Object#==).

```
Customer = Struct.new(:name, :address, :zip)

joe   = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)
joejr = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)
jane  = Customer.new("Jane Doe", "456 Elm, Anytown NC", 12345)

joe == joejr   # =>   true
joe == jane    # =>   false
```

[ ]

joe[*symbol*] → *obj*
joe[*integer*] → *obj*

Attribute Reference—Returns the value of the instance variable named by *symbol* or indexed ($0..length - 1$) by *int*. Raises NameError if the named variable does not exist or raises IndexError if the index is out of range.

```
Customer = Struct.new(:name, :address, :zip)
joe = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)
joe["name"]   # =>   "Joe Smith"
joe[:name]    # =>   "Joe Smith"
joe[0]        # =>   "Joe Smith"
```

[ ]=

joe[*symbol*] = *obj* → *obj*
joe[*int*] = *obj* → *obj*

Attribute Assignment—Assigns to the instance variable named by *symbol* or *int* the value *obj* and returns it. Raises a NameError if the named variable does not exist or raises an IndexError if the index is out of range.

```
Customer = Struct.new(:name, :address, :zip)
joe = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)
joe["name"] = "Luke"
joe[:zip]   = "90210"
joe.name   # =>   "Luke"
joe.zip    # =>   "90210"
```

each

joe.each {| *obj* | *block* } → *joe*

Calls *block* once for each instance variable, passing the value as a parameter.

```
Customer = Struct.new(:name, :address, :zip)
joe = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)
joe.each {|x| puts(x) }
```

produces:

```
Joe Smith
123 Maple, Anytown NC
12345
```

each_pair

joe.each_pair {| *symbol, obj* | *block* } → *joe*

Calls *block* once for each instance variable, passing the name (as a symbol) and the value as parameters.

```
Customer = Struct.new(:name, :address, :zip)
joe = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)
joe.each_pair {|name, value| puts("#{name} => #{value}") }
```

produces:

```
name => Joe Smith
address => 123 Maple, Anytown NC
zip => 12345
```

length

joe.length → *int*

Returns the number of attributes.

```
Customer = Struct.new(:name, :address, :zip)
joe = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)
joe.length   # =>  3
```

members

joe.members → *array*

Returns an array of strings representing the names of the instance variables.

```
Customer = Struct.new(:name, :address, :zip)
joe = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)
joe.members   # =>  [:name, :address, :zip]
```

size

joe.size → *int*

Synonym for Struct#length.

to_a

joe.to_a → *array*

Returns the values for this instance as an array.

```
Customer = Struct.new(:name, :address, :zip)
joe = Customer.new("Joe Smith", "123 Maple, Anytown NC", 12345)
joe.to_a[1]   # =>  "123 Maple, Anytown NC"
```

values

joe.values → *array*

Synonym for to_a.

values_at

joe.values_at(⟨ *selector* ⟩*) → *array*

Returns an array containing the elements in *joe* corresponding to the given indices. The selectors may be integer indices or ranges.

```
Lots = Struct.new(:a, :b, :c, :d, :e, :f)
l = Lots.new(11, 22, 33, 44, 55, 66)
l.values_at(1, 3, 5)     # =>  [22, 44, 66]
l.values_at(0, 2, 4)     # =>  [11, 33, 55]
l.values_at(-1, -3, -5)  # =>  [66, 44, 22]
```