# Ruby 2.2.4 Core API Reference API Reference

This is the API documentation for 'Ruby 2.2.4 Core API Reference API Reference'.











UnboundMethod
 UncaughtThrowError
 ZeroDivisionError

<sup>™</sup> <u>fatal</u>

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class ArgumentError

Raised when the arguments are wrong and there isn't a more specific <u>Exception</u> class.

Ex: passing the wrong number of arguments

```
[1, 2, 3].first(4, 5)
```

raises the exception:

ArgumentError: wrong number of arguments (2 fo

Ex: passing an argument that is not acceptable:

[1, 2, 3].first(-4)

raises the exception:

ArgumentError: negative array size

In Files

error.c

Parent StandardError Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class Array

Arrays are ordered, integer-indexed collections of any object.

Array indexing starts at 0, as in C or Java. A negative index is assumed to be relative to the end of the array—that is, an index of -1 indicates the last element of the array, -2 is the next to last element in the array, and so on.

## **Creating Arrays**

A new array can be created by using the literal constructor []. Arrays can contain different types of objects. For example, the array below contains an <u>Integer</u>, a <u>String</u> and a Float:

An array can also be created by explicitly calling <u>.:new</u> with zero, one (the initial size of the <u>Array</u>) or two arguments (the initial size and a default object).

```
ary = Array.new #=> []
Array.new(3) #=> [nil, nil, nil]
Array.new(3, true) #=> [true, true, true]
```

Note that the second argument populates the array with references to the same object. Therefore, it is only recommended in cases when you need to instantiate arrays with natively immutable objects such as Symbols, numbers, true or false.

To create an array with separate objects a block can be passed instead. This method is safe to use with mutable objects such as hashes, strings or other arrays:



•

This is also a quick way to build up multidimensional arrays:



An array can also be created by using the Array() method, provided by <u>Kernel</u>, which tries to call <u>to\_ary</u>, then <u>to\_a</u> on its argument.



## **Example Usage**

In addition to the methods it mixes in through the <u>Enumerable</u> module, the <u>Array</u> class has proprietary methods for accessing, searching and otherwise manipulating arrays.

Some of the more common ones are illustrated below.

## **Accessing Elements**

Elements in an array can be retrieved using the Array#[] method. It can take a single integer argument (a numeric index), a pair of arguments (start and length) or a range. Negative indices start counting from the end, with -1 being the last element.



Another way to access a particular array element is by using the <u>at</u> method

```
arr.at(0) #=>
```

The <u>slice</u> method works in an identical manner to Array#[].

To raise an error for indices outside of the array bounds or else to provide a default value when that happens, you can use <u>fetch</u>.



The special methods <u>first</u> and <u>last</u> will return the first and last elements of an array, respectively.



To return the first n elements of an array, use take

```
arr.take(3) #=> [1, 2, 3]
```

 $\frac{\text{drop}}{\text{elements}}$  does the opposite of  $\frac{\text{take}}{\text{have}}$ , by returning the elements after n elements have been dropped:



## **Obtaining Information about an <u>Array</u>**

Arrays keep track of their own length at all times. To query an array about the number of elements it contains, use <u>length</u>, <u>count</u> or <u>size</u>.



To check whether an array contains any elements at all

```
browsers.empty? #=> false
```

To check whether a particular item is included in the array



## **Adding Items to Arrays**

Items can be added to the end of an array by using either <u>push</u> or #<<



unshift will add a new item to the beginning of an array.

```
arr.unshift(0) #=> [0, 1, 2, 3, 4, 5, 6]
```

With <u>insert</u> you can add a new element to an array at any position.

Using the <u>insert</u> method, you can also insert multiple values at once:



## **Removing Items from an Array**

The method **pop** removes the last element in an array and returns it:

arr = [1, 2, 3, 4, 5, 6]
arr.pop #=> 6
arr #=> [1, 2, 3, 4, 5]

To retrieve and at the same time remove the first item, use shift:

arr.shift #=> 1 arr #=> [2, 3, 4, 5]

To delete an element at a particular index:

arr.delete\_at(2) #=> 4
arr #=> [2, 3, 5]

To delete a particular element anywhere in an array, use delete:



A useful method if you need to remove nil values from an array is <u>compact</u>:





Another common need is to remove duplicate elements from an array.

It has the non-destructive  $\underline{uniq}$ , and destructive method  $\underline{uniq!}$ 



## **Iterating over Arrays**

Like all classes that include the Enumerable module, Array has an each method, which defines what elements should be iterated over and how. In case of Array's <u>each</u>, all elements in the Array instance are yielded to the supplied block in sequence.

Note that this operation leaves the array unchanged.

Another sometimes useful iterator is <u>reverse\_each</u> which will iterate over the elements in the array in reverse order.



The <u>map</u> method can be used to create a new array based on the original array, but with the values modified by the supplied block:



arr.map!	{	a	a**2 ]	#=>	[1,	4,	9,	16,	25]
arr									

## Selecting Items from an Array

Elements can be selected from an array according to criteria defined in a block. The selection can happen in a destructive or a nondestructive manner. While the destructive operations will modify the array they were called on, the non-destructive methods usually return a new array with the selected elements, but leave the original array unchanged.

#### **Non-destructive Selection**



#### **Destructive Selection**

select! and reject! are the corresponding destructive methods to select and reject

Similar to <u>select</u> vs. <u>reject</u>, <u>delete\_if</u> and <u>keep\_if</u> have the exact opposite result when supplied with the same block:



## In Files

📄 array.c

arr

📄 pack.c

## Parent

Object

## **Included Modules**

Enumerable

## **Public Class Methods**

## 🏐 [](\*args)

Returns a new array populated with the given objects.



## new(size=0, default=nil)

#### 🚳 new(array)

#### new(size) {|index| block }

Returns a new array.

In the first form, if no arguments are sent, the new array will be empty. When a size and an optional default are sent, an array is created with size copies

of default. Take notice that all elements will reference the same object default.

The second form creates a copy of the array passed as a parameter (the array is generated by calling <u>#to\_ary</u> on the parameter).



In the last form, an array of the given size is created. Each element in this array is created by passing the element's index to the given block and storing the return value.

```
Array.new(3){ |index| index ** 2 }
# => [0, 1, 4]
```

#### **Common gotchas**

When sending the second parameter, the same object will be used as the value for all the array elements:



Since all the <u>Array</u> elements store the same hash, changes to one of them will affect them all.

If multiple copies are what you want, you should use the block version which uses the result of that block each time an element of the array needs to be initialized:

```
a = Array.new(2) { Hash.new }
a[0]['cat'] = 'feline'
a # => [{"cat"=>"feline"}, {}]
```

#### onvert(obj) → array or nil

Tries to convert obj into an array, using to\_ary method. Returns the converted array or nil if obj cannot be converted for any reason. This method can be used to check if an argument is an array.





## **Public Instance Methods**

#### $_{\odot}$ ary & other\_ary $\rightarrow$ new\_ary

Set Intersection — Returns a new array containing elements common to the two arrays, excluding any duplicates. The order is preserved from the original array.

It compares elements using their <u>hash</u> and <u>eql?</u> methods for efficiency.

See also #uniq.

# ary \* int → new\_ary ary \* str → new\_string Repetition — With a String argument, equivalent to ary.join(str). Otherwise, returns a new array built by concatenating the int copies of self.



ary + other\_ary → new\_ary Concatenation — Returns a new array built by concatenating the two arrays together to produce a third array.



Note that

x += y

is the same as

x = x + y

This means that it produces a new array. As a consequence, repeated use of += on arrays can be quite inefficient.

See also #concat.

#### $_{\odot}$ ary - other\_ary $\rightarrow$ new\_ary

Array Difference

Returns a new array that is a copy of the original array, removing any items that also appear in other\_ary. The order is preserved from the original array.

It compares elements using their <u>hash</u> and <u>eql?</u> methods for efficiency.



If you need set-like behavior, see the library class Set.

#### obj → ary

Append—Pushes the given object on to the end of this array. This expression returns the array itself, so several appends may be chained together.



#### $_{\odot}$ ary <=> other\_ary $\rightarrow$ -1, 0, +1 or nil

Comparison — Returns an integer (-1, 0, or +1) if this array is less than, equal to, or greater than other\_ary.

Each object in each array is compared (using the <=> operator).

Arrays are compared in an "element-wise" manner; the first element of ary is compared with the first one of other\_ary using the <=> operator, then each of the second elements, etc... As soon as the result of any such comparison is non zero (i.e. the two corresponding elements are not equal), that result is returned for the whole array comparison.

If all the elements are equal, then the result is based on a comparison of the array lengths. Thus, two arrays are "equal" according to Array#<=> if, and only if, they have the same length and the value of each element is equal to the value of the corresponding element in the other array.

nil is returned if the other\_ary is not an array or if the comparison of two elements returned nil.



#### $_{\odot}$ ary == other\_ary $\rightarrow$ bool

Equality — Two arrays are equal if they contain the same number of elements and if each element is equal to (according to Object#==) the corresponding element in other\_ary.



- ary[index] → obj or nil
- $_{\odot}$  ary[start, length]  $\rightarrow$  new\_ary or nil
- $_{\odot}$  ary[range]  $\rightarrow$  new\_ary or nil
- $\otimes$  slice(index)  $\rightarrow$  obj or nil
- $_{\odot}$  slice(start, length)  $\rightarrow$  new\_ary or nil
- $\otimes$  slice(range)  $\rightarrow$  new\_ary or nil

Element Reference — Returns the element at index, or returns a subarray starting at the start index and continuing for length elements, or returns a subarray specified by range of indices.

Negative indices count backward from the end of the array (-1 is the last element). For start and range cases the starting index is just before an element. Additionally, an empty array is returned when the starting index for an element range is at the end of the array.

Returns nil if the index (or starting index) are out of range.



ary[index] = obj → obj

ary[start, length] = obj or other\_ary or nil  $\rightarrow$   $_{\textcircled{}}$  obj or other\_ary or nil

ary[range] = obj or other\_ary or nil  $\rightarrow$  obj or  $\otimes$  other\_ary or nil

Element Assignment — Sets the element at index, or replaces a subarray from the start index for length elements, or replaces a subarray specified by the range of indices.

If indices are greater than the current capacity of the array, the array grows automatically. Elements are inserted into the array at start if length is zero.

Negative indices will count backward from the end of the array. For start and range cases the starting index is just before an element.

An IndexError is raised if a negative index points past the beginning of the array.

See also **#push**, and **#unshift**.

a = Array.new				
a[4] = "4";				
a[0, 3] = [ 'a', 'b',	'c' ]			
a[12] = [ 1, 2 ]				
a[0, 2] = "?"				
a[02] = "A"				
a[-1] = "Z"				
a[11] = nil				
a[11] = []		#=>	["A"]	



#### any? [{ |obj| block }] → true or false See also Enumerable#any?

#### assoc(obj) → new\_ary or nil

Searches through an array whose elements are also arrays comparing obj with the first element of each contained array using obj.==.

Returns the first contained array that matches (that is, the first associated array), or nil if no match is found.

See also **#rassoc** 



#### $_{\odot}$ at(index) $\rightarrow$ obj or nil

Returns the element at index. A negative index counts from the end of self. Returns nil if the index is out of range. See also Array#[].



#### $_{\odot}$ bsearch {|x| block } $\rightarrow$ elem

By using binary search, finds a value from this array

which meets the given condition in O(log n) where n is the size of the array.

You can use this method in two use cases: a findminimum mode and a find-any mode. In either case, the elements of the array must be monotone (or sorted) with respect to the block.

In find-minimum mode (this is a good choice for typical use case), the block must return true or false, and there must be an index i ( $0 \le i \le ary.size$ ) so that:

- the block returns false for any element whose index is less than i, and
- the block returns true for any element whose index is greater than or equal to i.

This method returns the i-th element. If i is equal to ary.size, it returns nil.

In find-any mode (this behaves like libc's bsearch(3)), the block must return a number, and there must be two indices i and j ( $0 \le i \le j \le ary.size$ ) so that:

- the block returns a positive number for <u>ary</u> if 0
  <= k < i,</p>
- The block returns zero for ary if  $i \le k \le j$ , and
- the block returns a negative number for <u>ary</u> if j <= k < ary.size.</p>

Under this condition, this method returns any element whose index is within i...j. If i is equal to j (i.e., there is no element that satisfies the block), this method returns nil.



You must not mix the two modes at a time; the block must always return either true/false, or always return a number. It is undefined which value is actually picked up at each iteration.

#### $\odot$ clear $\rightarrow$ ary

Removes all elements from self.



- collect { |item| block } → new\_ary
- $_{\odot}$  map { |item| block }  $\rightarrow$  new\_ary
- collect → Enumerator

#### map → Enumerator

Invokes the given block once for each element of self.

Creates a new array containing the values returned by the block.

See also Enumerable#collect.

If no block is given, an Enumerator is returned instead.



- $\odot$  collect! {|item| block }  $\rightarrow$  ary
- $_{\odot}$  map! {|item| block }  $\rightarrow$  ary
- collect! → Enumerator
- map! → Enumerator

Invokes the given block once for each element of self, replacing the element with the value returned by the block.

See also Enumerable#collect.

If no block is given, an Enumerator is returned instead.



#### rightarrow combination(n) { |c| block } → ary rightarrow combination(n) → Enumerator

When invoked with a block, yields all combinations of length n of elements from the array and then returns the array itself.

The implementation makes no guarantees about the order in which the combinations are yielded.

If no block is given, an Enumerator is returned instead.

Examples:

a = [1, 2, 3, 4]	
<pre>a.combination(1).to_a</pre>	
a.combination(2).to_a	
a.combination(3).to_a	
a.combination(4).to_a	
a.combination(0).to_a	
a.combination(5).to_a	
4	► F

## <u>↓</u>

#### ompact → new\_ary

Returns a copy of self with all nil elements removed.



#### $_{\odot}$ compact! $\rightarrow$ ary or nil

Removes nil elements from the array.

Returns nil if no changes were made, otherwise returns the array.



#### $\odot$ concat(other\_ary) $\rightarrow$ ary

Appends the elements of other\_ary to self.



See also Array#+.

# count → int count(obj) → int count { |item| block } → int Returns the number of elements. If an argument is given, counts the number of elements which equal obj using ==.

If a block is given, counts the number of elements for which the block returns a true value.



## sycle(n=nil) { |obj| block } → nil Sycle(n=nil) → Enumerator

Calls the given block for each element n times or forever if nil is given.

Does nothing if a non-positive number is given or the array is empty.

Returns nil if the loop has finished without getting interrupted.

If no block is given, an <u>Enumerator</u> is returned instead.



## objective (obj) → item or nil

↔ delete(obj) { block } → item or result of block

Deletes all items from self that are equal to obj.

Returns the last deleted item, or nil if no matching item is found.

If the optional code block is given, the result of the block is returned if the item is not found. (To remove nil elements and get an informative return value, use #compact!)


#### $\odot$ delete\_at(index) $\rightarrow$ obj or nil

Deletes the element at the specified index, returning that element, or nil if the index is out of range.

#### See also #slice!



# $_{\odot}$ delete\_if { |item| block } $\rightarrow$ ary

#### o delete\_if → Enumerator

Deletes every element of self for which block evaluates to true.

The array is changed instantly every time the block is called, not after the iteration is over.

See also #reject!

If no block is given, an <u>Enumerator</u> is returned instead.



#### $\odot$ drop(n) $\rightarrow$ new\_ary

Drops first n elements from ary and returns the rest of the elements in an array.

If a negative number is given, raises an ArgumentError.

See also **#take** 



#### $_{\odot}$ drop\_while { |arr| block } → new\_ary $_{\odot}$ drop\_while → Enumerator

Drops elements up to, but not including, the first element for which the block returns nil or false and returns an array containing the remaining elements.

If no block is given, an <u>Enumerator</u> is returned instead.

See also #take\_while



#### rightarrow each { |item| block } → ary rightarrow each → Enumerator

Calls the given block once for each element in self, passing that element as a parameter.

An Enumerator is returned if no block is given.



 $_{\odot}$  each\_index { |index| block } → ary  $_{\odot}$  each\_index → Enumerator

Same as <u>#each</u>, but passes the index of the element instead of the element itself.

An Enumerator is returned if no block is given.



#### $_{\odot}$ empty? $\rightarrow$ true or false

Returns true if self contains no elements.



## $_{\odot}$ eql?(other) $\rightarrow$ true or false

Returns true if self and other are the same object, or are both arrays with the same content (according to Object#eql?).

## fetch(index) → obj

# Setting fetch (index, default) → obj Setting fetch (index) { |index| block } → obj

Tries to return the element at position index, but throws an IndexError exception if the referenced index lies outside of the array bounds. This error can be prevented by supplying a second argument, which will act as a default value.

Alternatively, if a block is given it will only be executed when an invalid index is referenced. Negative values of index count from the end of the array.



- ill(obj) → ary
- $_{\odot}$  fill(obj, start [, length])  $\rightarrow$  ary
- $_{\odot}$  fill(obj, range )  $\rightarrow$  ary
- $_{\odot}$  fill { |index| block }  $\rightarrow$  ary
- $_{\textcircled{}}$  fill(start [, length] ) { |index| block }  $_{\rightarrow}$  ary
- $_{\odot}$  fill(range) { |index| block }  $\rightarrow$  ary

The first three forms set the selected elements of self (which may be the entire array) to obj.

A start of nil is equivalent to zero.

A length of nil is equivalent to the length of the array.

The last three forms fill the array with the value of the given block, which is passed the absolute index of each element to be filled.

Negative values of start count from the end of the array, where -1 is the last element.



find\_index(obj) → int or nil
 find\_index { |item| block } → int or nil

```
    find_index → Enumerator
```

- $_{\odot}$  index(obj)  $\rightarrow$  int or nil
- index { |item| block } → int or nil
- index → Enumerator

Returns the *index* of the first object in ary such that the object is == to obj.

If a block is given instead of an argument, returns the *index* of the first object for which the block returns true. Returns nil if no match is found.

See also #rindex.

An Enumerator is returned if neither a block nor argument is given.



# first → obj or nil

#### of irst(n) → new\_ary

Returns the first element, or the first n elements, of the array. If the array is empty, the first form returns nil, and the second form returns an empty array. See also **#last** for the opposite effect.



#### $_{\odot}$ flatten $\rightarrow$ new\_ary

#### flatten(level) → new\_ary

Returns a new array that is a one-dimensional flattening of self (recursively).

That is, for every element that is an array, extract its elements into the new array.

The optional level argument determines the level of recursion to flatten.



# I atten! → ary or nil I atten!(level) → ary or nil

Flattens self in place.

Returns nil if no modifications were made (i.e., the array contains no subarrays.)

The optional level argument determines the level of recursion to flatten.



#### frozen? → true or false

Return true if this array is frozen (or temporarily frozen while being sorted). See also Object#frozen?

#### 

Compute a hash-code for this array.

Two arrays with the same content will have the same

hash code (and will compare using <u>eql?</u>). See also Object#hash.

## include?(object) → true or false

Returns true if the given object is present in self (that is, if any element == object), otherwise returns false.

a = [ "a", "b", "c" ]
a.include?("b") #=> true
a.include?("z") #=> false

```
⊕ find_index(obj) → int or nil
```

```
ind_index { |item| block } → int or nil
```

```
\odot find_index \rightarrow Enumerator
```

```
_{\odot} index(obj) \rightarrow int or nil
```

 $_{\odot}$  index { |item| block }  $\rightarrow$  int or nil

#### index → Enumerator

Returns the *index* of the first object in ary such that the object is == to obj.

If a block is given instead of an argument, returns the *index* of the first object for which the block returns true. Returns nil if no match is found.

See also #rindex.

An Enumerator is returned if neither a block nor argument is given.



## $_{\odot}$ replace(other\_ary) → ary $_{\odot}$ initialize\_copy(other\_ary) → ary

Replaces the contents of self with the contents of other\_ary, truncating or expanding if necessary.



#### $_{\odot}$ insert(index, obj...) $\rightarrow$ ary

Inserts the given values before the element with the given index.

Negative indices count backwards from the end of the array, where -1 is the last element. If a negative index is used, the given values will be inserted after that element, so using an index of -1 will insert the values at the end of the array.



#### $_{\odot}$ inspect $\rightarrow$ string

#### $\odot$ to\_s $\rightarrow$ string

Creates a string representation of self.



Also aliased as: <u>to\_s</u>

## join(separator=\$,) → str

Returns a string created by converting each element

of the array to a string, separated by the given separator. If the separator is nil, it uses current \$,. If both the separator and \$, are nil, it uses empty string.



# where the set of the set of

Deletes every element of self for which the given block evaluates to false.

See also #select!

If no block is given, an Enumerator is returned instead.

```
a = %w{ a b c d e f }
a.keep_if { |v| v =~ /[aeiou]/ } #=> ["a", "e"]
```

# last → obj or nil

#### last(n) → new\_ary

Returns the last element(s) of self. If the array is empty, the first form returns nil.

See also **#first** for the opposite effect.



## $\otimes$ length $\rightarrow$ int

Returns the number of elements in self. May be zero.



#### [].length

Also aliased as: size

```
\odot collect { |item| block } \rightarrow new_ary
```

- $\otimes$  map { |item| block }  $\rightarrow$  new\_ary
- collect → Enumerator

#### map → Enumerator

Invokes the given block once for each element of self.

Creates a new array containing the values returned by the block.

```
See also Enumerable#collect.
```

If no block is given, an Enumerator is returned instead.



```
\odot collect! {|item| block } \rightarrow ary
```

```
_{\odot} map! {|item| block } \rightarrow ary
```

- collect! → Enumerator
- map! → Enumerator

Invokes the given block once for each element of self, replacing the element with the value returned by the block.

See also Enumerable#collect.

If no block is given, an Enumerator is returned instead.





## $_{\odot}$ pack ( aTemplateString ) $\rightarrow$ aBinaryString

Packs the contents of *arr* into a binary sequence according to the directives in *aTemplateString* (see the table below) Directives "A," "a," and "Z" may be followed by a count, which gives the width of the resulting field. The remaining directives also may take a count, indicating the number of array elements to convert. If the count is an asterisk ("\*"), all remaining array elements will be converted. Any of the directives "ssiIlL" may be followed by an underscore ("\_") or exclamation mark ("!") to use the underlying platform's native size for the specified type; otherwise, they use a platform-independent size. Spaces are ignored in the template string. See also String#unpack.



Directives for pack.

Integer	Array
Directive	Element   Meaning
C	Integer   8-bit unsigned (unsigned
S	Integer   16-bit unsigned, native
L	Integer   32-bit unsigned, native
Q	Integer   64-bit unsigned, native
C S	   Integer   8-bit signed (signed cha   Integer   16-bit signed, native er

1 q	Integer Integer	32-bit signed, native er 64-bit signed, native er
S_, S! I, I_, I! L_, L! Q_, Q!	Integer Integer Integer Integer	unsigned short, native e unsigned int, native end unsigned long, native er unsigned long long, nati if the platform has no l (Q_ and Q! is available
s_, s! i, i_, i! l_, l! q_, q!	Integer Integer Integer Integer	signed short, native end signed int, native endia signed long, native endi signed long long, native if the platform has no l (q_ and q! is available
S> L> Q> s> l> q> S!> I!> L!> Q!> s!> i!> l!> q!>	Integer	same as the directives w big endian (available since Ruby 1. "S>" is same as "n" "L>" is same as "N"
S< L< Q< s< l< q< S!< I!< L!< Q!< s!< i!< l!< q!<	Integer	same as the directives v little endian (available since Ruby 1. "S<" is same as "v" "L<" is same as "V"
n N V V	Integer Integer Integer Integer	16-bit unsigned, network 32-bit unsigned, network 16-bit unsigned, VAX (li 32-bit unsigned, VAX (li
U W	Integer   Integer	UTF-8 character BER-compressed integer
Float Directive		Meaning
D, d F, f E G	Float Float Float Float Float	double-precision, native single-precision, native double-precision, little single-precision, little double-precision, networ

g	Float	single-precision, networ
String		
Directive	i	Meaning
А	String	arbitrary binary string
a	String	arbitrary binary string
Z	String	same as ``a'', except th
В	String	bit string (MSB first)
b	String	bit string (LSB first)
Н	String	hex string (high nibble
h	String	hex string (low nibble f
u	String	UU-encoded string
М	String	quoted printable, MIME e
m	String	base64 encoded string (s
	1	(if count is 0, no line
Р	String	pointer to a structure (
р	String	pointer to a null-termir
Mico		
MISC. Directive		Mooning
DTLECTIVE	1	Meaning
<u> </u>		l moves to absolute <u>positi</u>
v v		hock up a byto
		pull byto
<b>↓</b>		▶

#### $\odot$ permutation { |p| block } $\rightarrow$ ary

#### $_{\odot}$ permutation $\rightarrow$ Enumerator

#### rightarrow permutation(n) { |p| block } → ary rightarrow permutation(n) → Enumerator

When invoked with a block, yield all permutations of length n of the elements of the array, then return the array itself.

If n is not specified, yield all permutations of all elements.

The implementation makes no guarantees about the order in which the permutations are yielded.

If no block is given, an Enumerator is returned

instead.

Examples:

a = [1, 2, 3]	
a.permutation.to_a  #=>	[[1,2,3],[1,3,2],[2,1,3
a.permutation(1).to_a <b>#=&gt;</b>	[[1],[2],[3]]
a.permutation(2).to_a <b>#=&gt;</b>	[[1,2],[1,3],[2,1],[2,3
a.permutation(3).to_a #=>	[[1,2,3],[1,3,2],[2,1,3
a.permutation(0).to_a <b>#=&gt;</b>	<pre>[[]] # one permutation</pre>
a.permutation(4).to_a <b>#=&gt;</b>	<pre>[] # no permutations</pre>
<u>الم</u>	

#### 

# Removes the last element from self and returns it, or

nil if the array is empty.

If a number n is given, returns an array of the last n elements (or less) just like array.slice!(-n, n) does. See also **#push** for the opposite effect.



#### $_{\odot}$ product(other\_ary, ...) → new\_ary $_{\odot}$ product(other\_ary, ...) { |p| block } → ary

Returns an array of all combinations of elements from all arrays.

The length of the returned array is the product of the length of self and the argument arrays.

If given a block, product will yield all combinations and return self instead.





#### $_{\odot}$ push(obj, ... ) $\rightarrow$ ary

Append — Pushes the given object(s) on to the end of this array. This expression returns the array itself, so several appends may be chained together. See also <u>#pop</u> for the opposite effect.



#### orassoc(obj) → new\_ary or nil

Searches through the array whose elements are also arrays.

Compares obj with the second element of each contained array using obj.==.

Returns the first contained array that matches obj.

See also #assoc.



# Some interpret of the second seco

Returns a new array containing the items in self for which the given block is not true.

See also #delete\_if

If no block is given, an Enumerator is returned instead.

#### reject! { |item| block } → ary or nil reject! → Enumerator

Equivalent to <u>#delete\_if</u>, deleting elements from self for which the block evaluates to true, but returns nil if no changes were made.

The array is changed instantly every time the block is called, not after the iteration is over.

See also Enumerable#reject and #delete\_if.

If no block is given, an <u>Enumerator</u> is returned instead.

#### $repeated_combination(n) { |c| block } → ary$ $<math>repeated_combination(n) → Enumerator$

When invoked with a block, yields all repeated combinations of length n of elements from the array and then returns the array itself.

The implementation makes no guarantees about the order in which the repeated combinations are yielded.

If no block is given, an Enumerator is returned instead.

Examples:

#### $repeated_permutation(n) { |p| block } → ary$ $repeated_permutation(n) → Enumerator$

When invoked with a block, yield all repeated permutations of length n of the elements of the array, then return the array itself.

The implementation makes no guarantees about the order in which the repeated permutations are yielded.

If no block is given, an <u>Enumerator</u> is returned instead.

Examples:

4



#### $_{\odot}$ replace(other\_ary) → ary $_{\odot}$ initialize\_copy(other\_ary) → ary

Replaces the contents of self with the contents of other\_ary, truncating or expanding if necessary.



#### ⊚ reverse → new\_ary

Returns a new array containing self's elements in

reverse order.



#### $\odot$ reverse! $\rightarrow$ ary

Reverses self in place.



#### $reverse_each { |item| block } → ary$ reverse each → Enumerator

Same as #each, but traverses self in reverse order.

```
a = [ "a", "b", "c" ]
a.reverse_each {|x| print x, " " }
```

produces:

#### os rindex(obj) → int or nil

#### on rindex { |item| block } → int or nil

#### index → Enumerator

Returns the *index* of the last object in self == to obj.

If a block is given instead of an argument, returns the *index* of the first object for which the block returns true, starting from the last object.

Returns nil if no match is found.

See also <u>#index</u>.

If neither block nor argument is given, an Enumerator

is returned instead.



#### or rotate(count=1) → new\_ary

Returns a new array by rotating self so that the element at count is the first element of the new array.

If count is negative then it rotates in the opposite direction, starting from the end of self where -1 is the last element.



# orotate!(count=1) → ary

Rotates self in place so that the element at count comes first, and returns self.

If count is negative then it rotates in the opposite direction, starting from the end of the array where -1 is the last element.



```
_{\odot} sample → obj
_{\odot} sample(random: rng) → obj
```

#### $_{\odot}$ sample(n) → new\_ary $_{\odot}$ sample(n, random: rng) → new\_ary

Choose a random element or n random elements from the array.

The elements are chosen by using random and unique indices into the array in order to ensure that an element doesn't repeat itself unless the array already contained duplicate elements.

If the array is empty the first form returns nil and the second form returns an empty array.

The optional rng argument will be used as the random number generator.



#### $_{\odot}$ select { |item| block } → new\_ary $_{\odot}$ select → Enumerator

Returns a new array containing all elements of ary for which the given block returns a true value.

If no block is given, an Enumerator is returned instead.



See also Enumerable#select.

 select! {|item| block } → ary or nil select! → Enumerator Invokes the given block passing in successive elements from self, deleting elements for which the block returns a false value.

If changes were made, it will return self, otherwise it returns nil.

See also #keep\_if

If no block is given, an <u>Enumerator</u> is returned instead.

#### 

Removes the first element of self and returns it (shifting all other elements down by one). Returns nil if the array is empty.

If a number n is given, returns an array of the first n elements (or less) just like array.slice!(0, n) does. With ary containing only the remainder elements, not including what was shifted to new\_ary. See also #unshift for the opposite effect.



## Shuffle → new\_ary

#### $\otimes$ shuffle(random: rng) $\rightarrow$ new\_ary

Returns a new array with elements of self shuffled.



The optional rng argument will be used as the random number generator.



#### $_{\odot}$ shuffle! → ary $_{\odot}$ shuffle!(random: rng) → ary

Shuffles elements in self in place.



The optional rng argument will be used as the random number generator.

```
a.shuffle!(random: Random.new(1))  #=> [1, 3, 2]
```

```
a size()
Alias for: <u>length</u>
```

а

```
ary[index] → obj or nil
ary[start, length] → new_ary or nil
ary[range] → new_ary or nil
slice(index) → obj or nil
slice(start, length) → new_ary or nil
slice(range) → new_ary or nil
Element Reference — Returns the element at index, or returns a subarray starting at the start index and continuing for length elements, or returns a subarray specified by range of indices.
```

Negative indices count backward from the end of the array (-1 is the last element). For start and range cases the starting index is just before an element. Additionally, an empty array is returned when the starting index for an element range is at the end of the array.

Returns nil if the index (or starting index) are out of range.



```
  slice!(index) \rightarrow obj or nil
```

```
  slice!(start, length) → new_ary or nil   slice!(range) → new_ary or nil
```

Deletes the element(s) given by an index (optionally up to length elements) or by a range.

Returns the deleted object (or objects), or nil if the index is out of range.



# omega sort → new\_ary omega sort { |a, b| block } → new\_ary

Returns a new array created by sorting self.

Comparisons for the sort will be done using the <=> operator or using an optional code block.

The block must implement a comparison between a and b, and return -1, when a follows b, 0 when a and b are equivalent, or +1 if b follows a.

See also Enumerable#sort\_by.



# sort! → ary sort! { |a, b| block } → ary Sorts solf in place

Sorts self in place.

Comparisons for the sort will be done using the <=> operator or using an optional code block.

The block must implement a comparison between a and b, and return -1, when a follows b, 0 when a and b are equivalent, or +1 if b follows a.

See also Enumerable#sort\_by.



 $\odot$  sort\_by! { |obj| block }  $\rightarrow$  ary

#### sort\_by! → Enumerator

Sorts self in place using a set of keys generated by mapping the values in self through the given block.

If no block is given, an Enumerator is returned instead.

#### rightarrow take(n) $\rightarrow$ new\_ary

Returns first n elements from the array.

If a negative number is given, raises an ArgumentError.

See also #drop



#### take\_while { |arr| block } → new\_ary take\_while → Enumerator

Passes elements to the block until the block returns nil or false, then stops iterating and returns an array of all prior elements.

If no block is given, an <u>Enumerator</u> is returned instead.

See also #drop\_while



#### to\_a → ary

Returns self.

If called on a subclass of <u>Array</u>, converts the receiver to an Array object.

#### ⊚ to\_ary → ary

Returns self.

#### osto\_h → hash

Returns the result of interpreting *ary* as an array of [key, value] pairs.

[[:foo, :bar], [1, 2]].to\_h
 # => {:foo => :bar, 1 => 2}

(a to\_s()
 Alias for: inspect

#### $\odot$ transpose $\rightarrow$ new\_ary

Assumes that self is an array of arrays and transposes the rows and columns.



If the length of the subarrays don't match, an IndexError is raised.

#### $rig → new_ary$ $rig uniq { |item| ... } → new_ary$

Returns a new array by removing duplicate values in self.

If a block is given, it will use the return value of the block for comparison.

It compares values using their <u>hash</u> and <u>eql?</u> methods for efficiency.



# $_{\odot}$ uniq! → ary or nil $_{\odot}$ uniq! { |item| ... } → ary or nil

Removes duplicate elements from self.

If a block is given, it will use the return value of the block for comparison.

It compares values using their  $\underline{hash}$  and  $\underline{eql?}$  methods for efficiency.

Returns nil if no changes are made (that is, no duplicates are found).



# $_{\odot}$ unshift(obj, ...) $\rightarrow$ ary

Prepends objects to the front of self, moving other elements upwards. See also  $\frac{\text{#shift}}{\text{for the opposite}}$  effect.



#### $\odot$ values\_at(selector, ...) $\rightarrow$ new\_ary

Returns an array containing the elements in self corresponding to the given selector(s).

The selectors may be either integer indices or ranges.

See also **#select**.



#### $_{\odot}$ zip(arg, ...) → new\_ary $_{\odot}$ zip(arg, ...) { |arr| block } → nil

Converts any arguments to arrays, then merges elements of self with corresponding elements from each argument.

This generates a sequence of ary.size *n*-element arrays, where *n* is one more than the count of arguments.

If the size of any argument is less than the size of the initial array, nil values are supplied.

If a block is given, it is invoked for each output array, otherwise an array of arrays is returned.



#### ary | other\_ary → new\_ary

Set Union — Returns a new array by joining ary with other\_ary, excluding any duplicates and preserving the order from the original array.

It compares elements using their  $\underline{hash}$  and  $\underline{eql?}$  methods for efficiency.



See also <u>#uniq</u>.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class BasicObject

BasicObject is the parent class of all classes in Ruby. It's an explicit blank class.

BasicObject can be used for creating object hierarchies independent of Ruby's object hierarchy, proxy objects like the Delegator class, or other uses where namespace pollution from Ruby's methods and classes must be avoided.

To avoid polluting <u>BasicObject</u> for other users an appropriately named subclass of <u>BasicObject</u> should be created instead of directly modifying BasicObject:

# class MyObjectSystem < BasicObject end</pre>

BasicObject does not include Kernel (for methods like puts) and BasicObject is outside of the namespace of the standard library so common classes will not be found without using a full class path.

A variety of strategies can be used to provide useful portions of the standard library to subclasses of <u>BasicObject</u>. A subclass could include Kernel to Obtain puts, exit, etc. A custom Kernel-like module could be created and included or delegation can be used via method\_missing:

```
class MyObjectSystem < BasicObject
DELEGATE = [:puts, :p]
def method_missing(name, *args, &block)
super unless DELEGATE.include? name
::Kernel.send(name, *args, &block)
end
def respond_to_missing?(name, include_private
DELEGATE.include?(name) or super
end
end
```

Access to classes and modules from the Ruby standard library can be obtained in a <u>BasicObject</u> subclass by referencing the desired constant from the root like ::File or ::Enumerator. Like method\_missing, const\_missing can be used to delegate constant lookup to object:

```
class MyObjectSystem < BasicObject
  def self.const_missing(name)
    ::Object.const_get(name)
  end
end</pre>
```

# **In Files**

- 📄 class.c
- 📄 gc.c
- object.c
- 📄 vm\_eval.c

# Parent

# **Public Class Methods**

new() Not documented

# **Public Instance Methods**

● !obj → true or false Boolean negate.

#### obj != other → true or false

Returns true if two objects are not-equal, otherwise false.

- $\odot$  obj == other  $\rightarrow$  true or false
- $_{\odot}$  equal?(other)  $\rightarrow$  true or false

```
_{\odot} eql?(other) \rightarrow true or false
```

Equality — At the object level, == returns true only if obj and other are the same object. Typically, this method is overridden in descendant classes to provide class-specific meaning.

Unlike ==, the equal? method should never be overridden by subclasses as it is used to determine object identity (that is, a.equal?(b) if and only if a is the same object as b):

```
obj = "a"
other = obj.dup
```



The eql? method returns true if obj and other refer to the same hash key. This is used by <u>Hash</u> to test members for equality. For objects of class object, eql? is synonymous with ==. Subclasses normally continue this tradition by aliasing eql? to their overridden == method, but there are exceptions. Numeric types, for example, perform type conversion across ==, but not across eql?, so:



# id\_\_\_\_id\_\_\_→ integer ∞ object\_id → integer

Returns an integer identifier for obj.

The same number will be returned on all calls to object\_id for a given object, and no two active objects will share an id.

Note: that some objects of builtin classes are reused for optimization. This is the case for immediate values and frozen string literals.

Immediate values are not passed by reference but are passed by value: nil, true, false, Fixnums, Symbols, and some Floats.



```
    send(symbol [, args...]) → obj
    send__(symbol [, args...]) → obj
    send(string [, args...]) → obj
    send__(string [, args...]) → obj
    Invokes the method identified by symbol, passing it any arguments specified. You can use __send__ if the name send clashes with an existing method in obj.
    When the method is identified by a string, the string is converted to a symbol.
```



#### rightarrow obj == other → true or false rightarrow equal?(other) → true or false rightarrow eql?(other) → true or false

Equality — At the object level, == returns true only if obj and other are the same object. Typically, this method is overridden in descendant classes to provide class-specific meaning.

Unlike ==, the equal? method should never be overridden by subclasses as it is used to determine object identity (that is, a.equal?(b) if and only if a is the same object as b):

```
obj = "a"
other = obj.dup
obj == other  #=> true
obj.equal? other #=> false
obj.equal? obj #=> true
```

The eql? method returns true if obj and other refer to the same hash key. This is used by <u>Hash</u> to test members for equality. For objects of class object, eql? is synonymous with ==. Subclasses normally continue this tradition by aliasing eql? to their overridden == method, but there are exceptions. Numeric types, for example, perform type conversion across ==, but not across eql?, so:

1 == 1.0 #=> true 1.eql? 1.0 #=> false

#### instance\_eval(string [, filename [, lineno]] ) $_{\textcircled{}} \rightarrow obj$

#### $_{\odot}$ instance\_eval {|obj| block } $\rightarrow$ obj

Evaluates a string containing Ruby source code, or the given block, within the context of the receiver (*obj*). In order to set the context, the variable self is set to *obj* while the code is executing, giving the code access to *obj*'s instance variables and private methods.

When instance\_eval is given a block, *obj* is also passed in as the block's only argument.

When instance\_eval is given a string, the optional second and third parameters supply a filename and starting line number that are used when reporting compilation errors.

```
class KlassWithSecret
  def initialize
    @secret = 99
  end
  private
  def the_secret
    "Ssssh! The secret is #{@secret}."
```



#### $\odot$ instance\_exec(arg...) {|var...| block } $\rightarrow$ obj

Executes the given block within the context of the receiver (*obj*). In order to set the context, the variable self is set to *obj* while the code is executing, giving the code access to *obj*'s instance variables. Arguments are passed as block parameters.



Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.
# class Bignum

Bignum objects hold integers outside the range of Fixnum. Bignum objects are created automatically when integer calculations would otherwise overflow a Fixnum. When a calculation involving Bignum objects returns a result that will fit in a Fixnum, the result is automatically converted.

For the purposes of the bitwise operations and [], a Bignum is treated as if it were an infinite-length bitstring with 2's complement representation.

While <u>Fixnum</u> values are immediate, <u>Bignum</u> objects are not—assignment and parameter passing work with references to objects, not the objects themselves.

#### **In Files**

📄 bignum.c

### Parent

Integer

### Constants

#### **GMP\_VERSION**

The version of loaded GMP.

### **Public Instance Methods**

# big % other → Numeric modulo(other) → Numeric

Returns big modulo other. See <u>Numeric#divmod</u> for more information.

#### big & numeric → integer

Performs bitwise and between big and numeric.

#### obig \* other → Numeric

Multiplies big and other, returning the result.

#### ig \*\* exponent → numeric

Raises *big* to the *exponent* power (which may be an integer, float, or anything that will coerce to a number). The result may be a Fixnum, Bignum, or Float



#### ig + other → Numeric

Adds big and other, returning the result.

big - other → Numeric

Subtracts other from big, returning the result.

#### osig → integer

Unary minus (returns an integer whose value is 0-big)

#### big / other → Numeric

Performs division: the class of the resulting object depends on the class of numeric and on the magnitude of the result.

#### $_{\odot}$ big < real → true or false

Returns true if the value of big is less than that of real.

#### big << numeric → integer</p>

Shifts big left *numeric* positions (right if *numeric* is negative).

#### $_{\odot}$ big <= real → true or false

Returns true if the value of big is less than or equal to that of real.

#### $_{\odot}$ big <=> numeric $\rightarrow$ -1, 0, +1 or nil

Comparison—Returns -1, 0, or +1 depending on whether big is less than, equal to, or greater than numeric. This is the basis for the tests in Comparable.

nil is returned if the two values are incomparable.

#### big == obj → true or false

Returns true only if *obj* has the same value as *big*. Contrast this with Bignum#eql?, which requires *obj* to be a Bignum.

```
68719476736 == 68719476736.0 #=> true
```

#### big == obj → true or false

Returns true only if *obj* has the same value as *big*. Contrast this with Bignum#eq1?, which requires *obj* to be a Bignum.

68719476736 == 68719476736.0 #=> true

#### $_{\odot}$ big > real → true or false

Returns true if the value of big is greater than that of real.

#### $_{\odot}$ big >= real → true or false

Returns true if the value of big is greater than or equal to that of real.

#### big >> numeric → integer

Shifts big right *numeric* positions (left if *numeric* is negative).

#### $\odot$ big[n] $\rightarrow$ 0, 1

Bit Reference—Returns the *n*th bit in the (assumed) binary representation of *big*, where <u>*big*</u> is the least significant bit.

a = 9\*\*15 50.downto(0) do |n|



#### big ^ numeric → integer

Performs bitwise +exclusive or+ between *big* and *numeric*.

### ⊚abs → aBignum

#### ⊚ magnitude → aBignum

Returns the absolute value of big.



### bit\_length → integer integer

Returns the number of bits of the value of *int*.

"the number of bits" means that the bit position of the highest bit which is different to the sign bit. (The bit position of the bit 2\*\*n is n+1.) If there is no such bit (zero or minus one), zero is returned.

I.e. This method returns ceil(log2(int < 0 ? -int : int+1)).

<pre>(-2**10000-1).bit_length (-2**10000).bit_length (-2**10000+1).bit_length</pre>	
(-2**1000-1).bit_length (-2**1000).bit_length (-2**1000+1).bit_length	
(2**1000-1).bit_length	

(2**1000).bit_length	#=> 1001
(2**1000+1).bit_length	#=> 1001
(2**10000-1).bit_length (2**10000).bit_length (2**10000+1).bit_length	

This method can be used to detect overflow in Array#pack as follows.

```
if n.bit_length < 32
    [n].pack("1") # no overflow
else
    raise "overflow"
end</pre>
```

#### oerce(numeric) → array

Returns an array with both a numeric and a big represented as Bignum objects.

This is achieved by converting numeric to a Bignum.

A <u>TypeError</u> is raised if the numeric is not a <u>Fixnum</u> or Bignum type.



#### oiv(other) → integer

Performs integer division: returns integer value.

#### omega divmod(numeric) → array

See Numeric#divmod.

#### eql?(obj) → true or false

Returns true only if *obj* is a Bignum with the same value as *big*. Contrast this with Bignum#==, which

performs type conversions.



# Returns the floating point result of dividing *big* by *numeric*.



### 

Compute a hash based on the value of big.

See also Object#hash.

```
(a) inspect(p1 = v1)
```

Alias for: to\_s

#### abs → aBignum

#### magnitude → aBignum

Returns the absolute value of big.



### $_{\odot}$ big % other → Numeric

modulo(other) → Numeric

Returns big modulo other. See Numeric#divmod for

more information.

#### odd? → true or false

Returns true if *big* is an odd number.

#### oremainder(numeric) → number

Returns the remainder after dividing big by numeric.



#### integer → integer

Returns the number of bytes in the machine representation of *big*.

(256**10	1).size	
(256**20	1).size	
(256**40	1).size	

#### $\odot$ to\_f $\rightarrow$ float

Converts *big* to a Float. If *big* doesn't fit in a Float, the result is infinity.

#### osto\_s(base=10) → string

Returns a string containing the representation of *big* radix *base* (2 through 36).

12345654321.to\_s #=> "12345654321" 12345654321.to\_s(2) #=> "10110111111011011: 12345654321.to\_s(8) #=> "133766736061" 12345654321.to\_s(16) #=> "2dfdbbc31" 78546939656932.to\_s(36) #=> "rubyrules" Also aliased as: inspect

#### ⓑ big | numeric → integer

Performs bitwise or between big and numeric.

#### 

Inverts the bits in big. As Bignums are conceptually infinite length, the result acts as if it had an infinite number of one bits to the left. In hex representations, this is displayed as two periods to the left of the digits.



# class Binding

Objects of class Binding encapsulate the execution context at some particular place in the code and retain this context for future use. The variables, methods, value of self, and possibly an iterator block that can be accessed in this context are all retained. <u>Binding</u> objects can be created using Kernel#binding, and are made available to the callback of Kernel#set\_trace\_func.

These binding objects can be passed as the second argument of the Kernel#eval method, establishing an environment for the evaluation.

```
class Demo
  def initialize(n)
    @secret = n
  end
  def get_binding
    return binding()
  end
end
k1 = Demo.new(99)
b1 = k1.get_binding
k2 = Demo.new(-3)
b2 = k2.get_binding
eval("@secret", b1) #=> 99
eval("@secret", b2) #=> -3
eval("@secret") #=> nil
```

Binding objects have no class-specific methods.

### **In Files**

proc.c

### Parent

Object

### **Public Instance Methods**

● eval(string [, filename [,lineno]]) → obj Evaluates the Ruby expression(s) in string, in the binding's context. If the optional filename and lineno parameters are present, they will be used when reporting syntax errors.

```
def get_binding(param)
   return binding
end
b = get_binding("hello")
b.eval("param") #=> "hello
```

### $\odot$ local\_variable\_defined?(symbol) $\rightarrow$ obj

Returns a true if a local variable symbol exists.



This method is short version of the following code.

```
binding.eval("defined?(#{symbol}) == 'local-varia
```

۲

#### $\odot$ local\_variable\_get(symbol) $\rightarrow$ obj

Returns a value of local variable symbol.

This method is short version of the following code.

```
binding.eval("#{symbol}")
```

4

#### $\odot$ local\_variable\_set(symbol, obj) $\rightarrow$ obj

Set local variable named symbol as obj.



This method is a similar behavior of the following code

#### binding.eval("#{symbol} = #{obj}")

if obj can be dumped in Ruby code.

### $_{\odot}$ local\_variables $\rightarrow$ Array

Returns the symbol names of the binding's local

variables



This method is short version of the following code.

```
binding.eval("local_variables")
```

Seturns the bound receiver of the binding object.

# class Class

Classes in Ruby are first-class objects—each is an instance of class class.

Typically, you create a new class by using:



When a new class is created, an object of type <u>Class</u> is initialized and assigned to a global constant (Name in this case).

When Name.new is called to create a new object, the new method in class is run by default. This can be demonstrated by overriding new in class:

```
class Class
  alias old_new new
  def new(*args)
    print "Creating a new ", self.name, "\n"
    old_new(*args)
    end
end
class Name
end
n = Name.new
```

produces:

#### Creating a new Name

Classes, modules, and objects are interrelated. In the diagram that follows, the vertical arrows represent inheritance, and the parentheses metaclasses. All metaclasses are instances of the class `Class'.



### In Files

📄 class.c

📄 object.c

#### Parent

Module

### **Public Class Methods**

# mew(super\_class=Object) → a\_class new(super\_class=Object) { |mod| ... } → a\_class

Creates a new anonymous (unnamed) class with the given superclass (or object if no parameter is given). You can give a class a name by assigning the class object to a constant.

If a block is given, it is passed the class object, and the block is evaluated in the context of this class using class\_eval.



Assign the class to a constant (name starting uppercase) if you want to treat it like a regular class.

### **Public Instance Methods**

#### allocate() → obj

Allocates space for a new object of *class*'s class and does not call initialize on the new instance. The returned object must be an instance of *class*.



#### obj new(args, ...) → obj

Calls allocate to create a new object of *class*'s class, then invokes that object's initialize method, passing it *args*. This is the method that ends up getting called whenever an object is constructed using .new.

#### superclass → a\_super\_class or nil

Returns the superclass of *class*, or nil.



Returns nil when the given class does not have a parent class:



# module Comparable

The comparable mixin is used by classes whose objects may be ordered. The class must define the <=> operator, which compares the receiver against another object, returning -1, 0, or +1 depending on whether the receiver is less than, equal to, or greater than the other object. If the other object is not comparable then the <=> operator should return nil. Comparable uses <=> to implement the conventional comparison operators (<, <=, ==, >=, and >) and the method between?.

```
class SizeMatters
  include Comparable
  attr :str
  def <=>(anOther)
    str.size <=> an0ther.str.size
  end
  def initialize(str)
    @str = str
  end
  def inspect
    @str
  end
end
s1 = SizeMatters.new("Z")
s2 = SizeMatters.new("YY")
s3 = SizeMatters.new("XXX")
s4 = SizeMatters.new("WWWW")
s5 = SizeMatters.new("VVVVV")
s1 < s2
```



### In Files

📄 compar.c

### **Public Instance Methods**

#### $_{\odot}$ obj < other $\rightarrow$ true or false

Compares two objects based on the receiver's <=> method, returning true if it returns -1.

#### $\odot$ obj <= other $\rightarrow$ true or false

Compares two objects based on the receiver's <=> method, returning true if it returns -1 or 0.

#### $_{\odot}$ obj == other $\rightarrow$ true or false

Compares two objects based on the receiver's <=> method, returning true if it returns 0. Also returns true if *obj* and *other* are the same object.

Even if *obj* <=> *other* raised an exception, the exception is ignored and returns false.

#### $\odot$ obj > other $\rightarrow$ true or false

Compares two objects based on the receiver's <=> method, returning true if it returns 1.

#### $_{\odot}$ obj >= other $\rightarrow$ true or false

Compares two objects based on the receiver's <=> method, returning true if it returns 0 or 1.

#### $_{\odot}$ between?(min, max) → true or false

Returns false if *obj* <=> *min* is less than zero or if an<u>Object</u> <=> max is greater than zero, true otherwise.



# class Complex

A complex number can be represented as a paired real number with imaginary unit; a+bi. Where a is real part, b is imaginary part and i is imaginary unit. Real a equals complex a+0i mathematically.

In ruby, you can create complex object with Complex, ::rect, ::polar or <u>#to\_c</u> method.



You can also create complex object from floating-point numbers or strings.



A complex object is either an exact or an inexact number.

Complex(1, 1) / 2 #=> ((1/2)+(1/2)\*i) Complex(1, 1) / 2.0 #=> (0.5+0.5i)

### **In Files**

complex.c

### Parent

Numeric

### Constants

Ι

The imaginary unit.

### **Public Class Methods**

### $_{\odot}$ polar(abs[, arg]) $\rightarrow$ complex

Returns a complex object which denotes the given polar form.



◎ rect(real[, imag]) → complex
 ◎ rectangular(real[, imag]) → complex
 Returns a complex object which denotes the given

rectangular form.



cmp \*\* numeric → complex

Performs exponentiation.



↔ cmp + numeric → complex Performs addition.



Complex(-2, 9)	+	Complex(-9,	2)	#=> (-11+11i)
Complex(9, 8)	+	4		
Complex(20, 9)	+	9.8		

#### omp - numeric → complex

Performs subtraction.

Complex(2, 3)- Complex(2, 3)#=> (0+0i)Complex(900)- Complex(1)#=> (899+0i)Complex(-2, 9)- Complex(-9, 2)#=> (7+7i)Complex(9, 8)- 4#=> (5+8i)Complex(20, 9)- 9.8#=> (10.2+9i)

#### $\odot$ -cmp $\rightarrow$ complex

Returns negation of the value.



#### $\odot$ cmp == object $\rightarrow$ true or false

Returns true if cmp equals object numerically.

Complex(2, 3)	== Complex(2, 3)	
Complex(5)	== 5	
Complex(0)	== 0.0	
Complex('1/3')	== 0.33	
Complex('1/2')	== '1/2'	

#### $rightarrow abs \rightarrow real$ $rightarrow magnitude \rightarrow real$

Returns the absolute part of its polar form.



#### $_{\odot}$ abs2 $\rightarrow$ real

Returns square of the absolute value.





conj → complex
 conjugate → complex
 Returns the complex conjugate.

#### Complex(1, 2).conjugate #=> (1-2i)

#### ⊚ conj → complex

#### onjugate → complex

Returns the complex conjugate.



#### og denominator → integer

Returns the denominator (lcm of both denominator - real and imag).

See numerator.

#### of fdiv(numeric) → complex

Performs division as each part is a float, never returns a float.



```
⊚imag → real
```

#### $\odot$ imaginary $\rightarrow$ real

Returns the imaginary part.



⊚imag → real

#### $_{\odot}$ imaginary $\rightarrow$ real

Returns the imaginary part.



#### $\odot$ inspect $\rightarrow$ string

Returns the value as a string for inspection.



#### ⊚abs → real

#### magnitude → real

Returns the absolute part of its polar form.



#### onumerator → numeric

Returns the numerator.



See denominator.

 $rig \rightarrow float$  $rig angle \rightarrow float$ 

#### → phase → float

Returns the angle part of its polar form.



#### 

Returns an array; [cmp.abs, cmp.arg].



# cmp / numeric → complex a quo(numeric) → complex

Performs division.



### $_{\odot}$ rationalize([eps]) $\rightarrow$ rational

Returns the value as a rational if possible (the imaginary part should be exactly zero).



See to\_r.

#### $rightarrow real \rightarrow real$

Returns the real part.





#### $\odot$ to\_f $\rightarrow$ float

Returns the value as a float if possible (the imaginary part should be exactly zero).

Complex(1, 0).to\_f #=> 1.0
Complex(1, 0.0).to\_f # RangeError
Complex(1, 2).to\_f # RangeError

 $\odot$  to\_i  $\rightarrow$  integer

Returns the value as an integer if possible (the imaginary part should be exactly zero).



#### oto\_r → rational

Returns the value as a rational if possible (the imaginary part should be exactly zero).



See rationalize.

#### os to\_s → string

Returns the value as a string.



#### onj → complex

#### onjugate → complex

Returns the complex conjugate.

Complex(1, 2).conjugate #=> (1-2i)

# class Complex::compatible

### **In Files**

complex.c

### Parent

Object

# class Continuation

Continuation objects are generated by Kernel#callcc, after having +require+d continuation. They hold a return address and execution context, allowing a nonlocal return to the end of the callcc block from anywhere within a program. Continuations are somewhat analogous to a structured version of C's setjmp/longjmp (although they contain more state, so you might consider them closer to threads).

For instance:

```
require "continuation"
arr = [ "Freddie", "Herbie", "Ron", "Max", "R:
callcc{|cc| $cc = cc}
puts(message = arr.shift)
$cc.call unless message =~ /Max/
```

produces:



This (somewhat contrived) example allows the inner loop to abandon processing early:

```
require "continuation"
callcc {|cont|
  for i in 0..4
    print "\n#{i}: "
    for j in i*5...(i+1)*5
        cont.call() if j == 17
        printf "%3d", j
    end
    end
}
puts
```

produces:

0:	0	1	2	3	4
1:	5	6	7	8	9
2:	10	11	12	13	14
3:	15	16			

### **In Files**

📄 cont.c

### Parent

Object

### **Public Instance Methods**

### ⊚ call(args, ...)

### cont[args, ...]

Invokes the continuation. The program continues from the end of the callcc block. If no arguments are given, the original callcc returns nil. If one argument

is given, callcc returns it. Otherwise, an array containing *args* is returned.



### ⊚ call(args, ...) ⊚ cont[args, ...]

Invokes the continuation. The program continues from the end of the callcc block. If no arguments are given, the original callcc returns nil. If one argument is given, callcc returns it. Otherwise, an array containing *args* is returned.



## class Data

This is a recommended base class for C extensions using Data\_Make\_Struct or Data\_Wrap\_Struct, see README.EXT for details.

### **In Files**

object.c

### Parent

Object

# class Dir

Objects of class Dir are directory streams representing directories in the underlying file system. They provide a variety of ways to list directories and their contents. See also File.

The directory used in these examples contains the two regular files (config.h and main.rb), the parent directory (...), and the directory itself (.).

### **In Files**

📄 dir.c

### Parent

Object

### **Included Modules**

Enumerable

### **Public Class Methods**

 $\odot$  chdir( [ string] )  $\rightarrow$  0
Chdir([string]) {| path | block } → anObject Changes the current working directory of the process to the given string. When called without an argument, changes the directory to the value of the environment variable HOME, Or LOGDIR. SystemCallError (probably Errno::ENOENT) if the target directory does not exist.

If a block is given, it is passed the name of the new current directory, and the block is executed with that as the current directory. The original working directory is restored when the block exits. The return value of chdir is the value of the block. chdir blocks can be nested, but in a multi-threaded program an error will be raised if a thread attempts to open a chdir block while another thread has one open.

```
Dir.chdir("/var/spool/mail")
puts Dir.pwd
Dir.chdir("/tmp") do
   puts Dir.pwd
   Dir.chdir("/usr") do
      puts Dir.pwd
   end
   puts Dir.pwd
end
puts Dir.pwd
```

produces:

/var/spool/mail		
/tmp		
/usr		
/tmp		
/var/spool/mail		

#### rightarrow chroot( string ) $\rightarrow$ 0

Changes this process's idea of the file system root. Only a privileged process may make this call. Not available on all platforms. On Unix systems, see chroot(2) for more information.

# $_{\odot}$ delete( string ) $\rightarrow$ 0

#### $_{\odot}$ rmdir( string ) $\rightarrow$ 0

#### $_{\odot}$ unlink( string ) $\rightarrow$ 0

Deletes the named directory. Raises a subclass of SystemCallError if the directory isn't empty.

## $_{\odot}$ entries( dirname ) $\rightarrow$ array

#### $_{\odot}$ entries( dirname, encoding: enc ) $\rightarrow$ array

Returns an array containing all of the filenames in the given directory. Will raise a systemCallError if the named directory doesn't exist.

The optional *enc* argument specifies the encoding of the directory. If not specified, the filesystem encoding is used.



#### $\otimes$ exist?(file\_name) $\rightarrow$ true or false

Returns true if the named file is a directory, false otherwise.

#### $\otimes$ exists?(file\_name) $\rightarrow$ true or false

Deprecated method. Don't use.

foreach( dirname ) {| filename | block } → nil
 foreach( dirname, encoding: enc ) {| filename
 block } → nil

- $| \mathsf{DIOCK} \rangle \to \mathsf{IIII}$
- $_{\odot}$  foreach( dirname )  $\rightarrow$  an\_enumerator

# foreach( dirname, encoding: enc ) $\rightarrow$ $_{\textcircled{}}$ an\_enumerator

Calls the block once for each entry in the named directory, passing the filename of each entry as a parameter to the block.

If no block is given, an enumerator is returned instead.



#### getwd → string

#### 

Returns the path to the current working directory of this process as a string.



# glob( pattern, [flags] ) → matches glob( pattern, [flags] ) { |filename| block } → nil

Expands pattern, which is an <u>Array</u> of patterns or a pattern <u>String</u>, and returns the results as matches or as arguments given to the block.

Note that this pattern is not a regexp, it's closer to a shell glob. See <u>File.fnmatch</u> for the meaning of the flags parameter. Note that case sensitivity depends

on your system (so File::FNM\_CASEFOLD is ignored), as does the order in which the results are returned.

\*

Matches any file. Can be restricted by other values in the glob. Equivalent to / .\* /x in regexp.

Matches all files

с\*

Matches all files beginning with c

\*c

Matches all files ending with c

\*C\*

Match all files that have c in them (including at the beginning or end).

Note, this will not match Unix-like hidden files (dotfiles). In order to include those in the match results, you must use the File::FNM\_DOTMATCH flag or something like "{\*, .\*}".

\* \*

Matches directories recursively.

?

Matches any one character. Equivalent to  $/.{1}/in$  regexp.

#### [set]

Matches any one character in set. Behaves exactly like character sets in <u>Regexp</u>, including set negation ([^a-z]).

 $\{p,q\}$ 

Matches either literal p or literal q. Equivalent to pattern alternation in regexp.

Matching literals may be more than one character

in length. More than two literals may be specified.

Escapes the next metacharacter.

Note that this means you cannot use backslash on windows as part of a glob, i.e. Dir["c:\foo\*"] will not work, use Dir["c:/foo\*"] instead.

Examples:

\

<pre>Dir["config.?"] Dir.glob("config.?") Dir.glob("*.[a-z][a-z]") Dir.glob("*.[^r]*") Dir.glob("*.{rb,h}") Dir.glob("*", File::FNM_DOTMATCH)</pre>	
<pre>rbfiles = File.join("**", "*.rb") Dir.glob(rbfiles)</pre>	
libdirs = File.join("**", "lib") Dir.glob(libdirs)	
<pre>librbfiles = File.join("**", "lib", Dir.glob(librbfiles)</pre>	"**", "*.rb") #=> ["lib/sor # "lib/sor
<pre>librbfiles = File.join("**", "lib", Dir.glob(librbfiles)</pre>	"*.rb") #=> ["lib/son

# 

home("root") → "/root"

Returns the home directory of the current user or the named user if given.

#### $_{\odot}$ mkdir( string [, integer] ) $\rightarrow$ 0

Makes a new directory named by string, with

permissions specified by the optional parameter an<u>Integer</u>. The permissions may be modified by the value of File::umask, and are ignored on NT. Raises a SystemCallError if the directory cannot be created. See also the discussion of permissions in the class documentation for File.



#### mew( string ) → aDir

#### $_{\odot}$ new( string, encoding: enc ) $\rightarrow$ aDir

Returns a new directory object for the named directory.

The optional *enc* argument specifies the encoding of the directory. If not specified, the filesystem encoding is used.

```
\odot open( string ) \rightarrow aDir
```

 $_{\odot}$  open( string, encoding: enc )  $\rightarrow$  aDir

open( string ) {| aDir | block } → anObject open( string, encoding: enc ) {| aDir | block }

```
        → anObject
```

The optional *enc* argument specifies the encoding of the directory. If not specified, the filesystem encoding is used.

With no block, open is a synonym for Dir::new. If a block is present, it is passed a<u>Dir</u> as a parameter. The directory is closed at the end of the block, and Dir::open returns the value of the block.

#### getwd → string

#### ⊚ pwd → string

Returns the path to the current working directory of this process as a string.



```
rightarrow rmdir( string ) \rightarrow 0
```

#### $_{\odot}$ unlink( string ) $\rightarrow$ 0

Deletes the named directory. Raises a subclass of SystemCallError if the directory isn't empty.

```
\odot delete( string ) \rightarrow 0
```

```
\odot rmdir( string ) \rightarrow 0
```

#### $_{\odot}$ unlink( string ) $\rightarrow$ 0

Deletes the named directory. Raises a subclass of SystemCallError if the directory isn't empty.

## **Public Instance Methods**

#### $\odot$ close $\rightarrow$ nil

Closes the directory stream. Any further attempts to access *dir* will raise an IDError.

```
d = Dir.new("testdir")
d.close #=> nil
```

#### each { |filename| block } → dir

 $_{\odot}$  each  $\rightarrow$  an\_enumerator

Calls the block once for each entry in this directory,

passing the filename of each entry as a parameter to the block.

If no block is given, an enumerator is returned instead.



produces:

Got . Got .. Got config.h Got main.rb

#### integer → integer

Returns the file descriptor used in dir.

d = Dir.new("..")
d.fileno #=> 8

This method uses dirfd() function defined by POSIX 2008. <u>NotImplementedError</u> is raised on other platforms, such as Windows, which doesn't provide the function.

#### inspect → string

Return a string describing this Dir object.

#### $_{\odot}$ path $\rightarrow$ string or nil

#### $\odot$ to\_path $\rightarrow$ string or nil

Returns the path parameter passed to *dir*'s constructor.



#### $\odot$ pos $\rightarrow$ integer

#### $\odot$ tell $\rightarrow$ integer

Returns the current position in *dir*. See also Dir#seek.

<pre>d = Dir.new("testdir")</pre>			
d.tell			
d.read			
d.tell			

#### opos = integer → integer

Synonym for Dir#seek, but returns the position parameter.



#### read → string or nil

Reads the next entry from *dir* and returns it as a string. Returns nil at the end of the stream.



#### $\odot$ rewind $\rightarrow$ dir

Repositions *dir* to the first entry.

d = Dir.new	w("testdir")
d.read	
d.rewind	
d.read	

#### $_{\odot}$ seek( integer ) $\rightarrow$ dir

Seeks to a particular location in *dir. integer* must be a value returned by Dir#tell.



#### 

#### $\odot$ tell $\rightarrow$ integer

Returns the current position in *dir*. See also Dir#seek.

```
d = Dir.new("testdir")
d.tell #=> 0
d.read #=> "."
d.tell #=> 12
```

# righting path $\rightarrow$ string or nil

#### $_{\odot}$ to\_path $\rightarrow$ string or nil

Returns the path parameter passed to *dir*'s constructor.



Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class ENV

 $\underline{\mathsf{ENV}}$  is a hash-like accessor for environment variables.

# In Files

📄 hash.c

## Parent

**Object** 

# **Public Class Methods**

#### Set in the set of the set of

Retrieves the value for environment variable name as a <u>String</u>. Returns nil if the named variable does not exist.

#### $_{\odot}$ ENV[name] = value $_{\odot}$ store(name, value) → value

Sets the environment variable name to value. If the value given is nil the environment variable is deleted.

#### assoc(name) → Array or nil

Returns an Array of the name and value of the

environment variable with name or nil if the name cannot be found.

#### 🚳 clear

Removes every environment variable.

#### $_{\odot}$ delete(name) → value $_{\odot}$ delete(name) { |name| } → value

Deletes the environment variable with name and returns the value of the variable. If a block is given it will be called when the named environment does not exist.

#### 

Deletes every environment variable for which the block evaluates to true.

If no block is given an enumerator is returned instead.

```
each { |name, value| } → Hash
each → Enumerator
each_pair { |name, value| } → Hash
each_pair → Enumerator
Yields each environment variable name and value.
If no block is given an Enumerator is returned.

each_key { |name| } → Hash
each_key → Enumerator
Yields each environment variable name.
```

An Enumerator is returned if no block is given.

#### $_{\odot}$ each { |name, value| } $\rightarrow$ Hash

- each → Enumerator
- $_{\odot}$  each\_pair { |name, value| }  $\rightarrow$  Hash

#### each\_pair → Enumerator

Yields each environment variable name and value.

If no block is given an Enumerator is returned.

#### each\_value { |value| } → Hash each\_value → Enumerator

Yields each environment variable value.

An Enumerator is returned if no block was given.

#### empty? → true or false

Returns true when there are no environment variables

#### fetch(name) → value

#### $_{\odot}$ fetch(name, default) → value

fetch(name) { |missing\_name| ... } → value
 Retrieves the environment variable name.

If the given name does not exist and neither default nor a block a provided an <u>IndexError</u> is raised. If a block is given it is called with the missing name to provide a value. If a default value is given it will be returned when no block is given.

## $\otimes$ key?(name) $\rightarrow$ true or false

#### $_{\odot}$ include?(name) $\rightarrow$ true or false

has\_key?(name) → true or false

#### $_{\odot}$ member?(name) $\rightarrow$ true or false

Returns true if there is an environment variable with the given name.

#### rightarrow value?(value) → true or false rightarrow has\_value?(value) → true or false

Returns true if there is an environment variable with the given value.

- $\otimes$  key?(name)  $\rightarrow$  true or false
- include?(name) → true or false
- has\_key?(name) → true or false
- $_{\odot}$  member?(name)  $\rightarrow$  true or false

Returns true if there is an environment variable with the given name.

#### $\otimes$ index(value) $\rightarrow$ key

Deprecated method that is equivalent to ::key

#### inspect → string

Returns the contents of the environment as a String.

#### invert → Hash

Returns a new hash created by using environment variable names as values and values as names.

#### $_{\odot}$ keep\_if { |name, value| } → Hash

#### ⊚ keep\_if → Enumerator

Deletes every environment variable where the block evaluates to false.

Returns an enumerator if no block was given.

#### $_{\odot}$ key(value) $\rightarrow$ name

Returns the name of the environment variable with value. If the value is not found nil is returned.

#### $\otimes$ key?(name) $\rightarrow$ true or false

- include?(name) → true or false
- $_{\odot}$  has\_key?(name)  $\rightarrow$  true or false

#### member?(name) → true or false

Returns true if there is an environment variable with the given name.

#### Skeys → Array

Returns every environment variable name in an Array

#### 🚳 length

#### 🚳 size

Returns the number of environment variables.

#### $_{\odot}$ key?(name) $\rightarrow$ true or false

- $_{\odot}$  include?(name)  $\rightarrow$  true or false
- $_{\odot}$  has\_key?(name) → true or false

#### $_{\odot}$ member?(name) $\rightarrow$ true or false

Returns true if there is an environment variable with the given name.

#### rassoc(value)

Returns an <u>Array</u> of the name and value of the environment variable with value or nil if the value cannot be found.

#### 🚳 rehash

Re-hashing the environment variables does nothing. It is provided for compatibility with Hash.

#### $reject { |name, value| } → Hash$ reject → Enumerator

Same as ENV#delete\_if, but works on (and returns) a copy of the environment.

#### reject! { |name, value| } → ENV or nil reject! → Enumerator

Equivalent to ENV#delete\_if but returns nil if no changes were made.

Returns an Enumerator if no block was given.

#### $_{\odot}$ replace(hash) $\rightarrow$ env

Replaces the contents of the environment variables with the contents of hash.

#### select { |name, value| } → Hash select → Enumerator

Returns a copy of the environment for entries where the block returns true.

Returns an Enumerator if no block was given.

#### select! { |name, value| } → ENV or nil select! → Enumerator

Equivalent to ENV#keep\_if but returns nil if no changes were made.

#### Shift → Array or nil

Removes an environment variable name-value pair from <u>ENV</u> and returns it as an <u>Array</u>. Returns nil if when the environment is empty.

#### 🚳 length

#### 🚳 size

Returns the number of environment variables.

#### rightarrow ENV[name] = valuerightarrow store(name, value) → value

Sets the environment variable name to value. If the value given is nil the environment variable is deleted.

#### oto\_a → Array

Converts the environment variables into an array of names and value arrays.



# to\_hash → hash

#### osto\_h → hash

Creates a hash with a copy of the environment variables.

#### oto\_hash → hash

#### osto\_h → hash

Creates a hash with a copy of the environment variables.

#### $\odot$ to\_s $\rightarrow$ "ENV"

Returns "ENV"

# □ update(hash) → Hash □ update(hash) { |name, old\_value, new\_value| □ } → Hash

Adds the contents of hash to the environment variables. If no block is specified entries with duplicate keys are overwritten, otherwise the value of each duplicate name is determined by calling the block with the key, its value from the environment and its value from the hash.

#### rightarrow value?(value) → true or false rightarrow has value?(value) → true or false

Returns true if there is an environment variable with the given value.

#### os values → Array

Returns every environment variable value as an Array

#### $\otimes$ values\_at(name, ...) $\rightarrow$ Array

Returns an array containing the environment variable values associated with the given names. See also ::select.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class EOFError

Raised by some <u>IO</u> operations when reaching the end of file. Many <u>IO</u> methods exist in two forms,

one that returns nil when the end of file is reached, the other raises EOFError EOFError.

EOFError is a subclass of IOError.



# **In Files**

📄 io.c

## Parent

**IOError** 

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class Encoding

An <u>Encoding</u> instance represents a character encoding usable in Ruby. It is defined as a constant under the <u>Encoding</u> namespace. It has a name and optionally, aliases:



Ruby methods dealing with encodings return or accept <u>Encoding</u> instances as arguments (when a method accepts an <u>Encoding</u> instance as an argument, it can be passed an <u>Encoding</u> name or alias instead).



Encoding::ASCII\_8BIT is a special encoding that is usually used for a byte string, not a character

string. But as the name insists, its characters in the range of ASCII are considered as ASCII characters. This is useful when you use ASCII-8BIT characters with other ASCII compatible characters.

# Changing an encoding

The associated <u>Encoding</u> of a <u>String</u> can be changed in two different ways.

First, it is possible to set the <u>Encoding</u> of a string to a new <u>Encoding</u> without changing the internal byte representation of the string, with <u>String#force\_encoding</u>. This is how you can tell Ruby the correct encoding of a string.

```
string
#=> "R\xC3\xA9sum\xC3\xA9"
string.encoding
#=> #<Encoding:ISO-8859-1>
string.force_encoding(Encoding::UTF_8)
#=> "R\u00E9sum\u00E9"
```

Second, it is possible to transcode a string, i.e. translate its internal byte representation to another encoding. Its associated encoding is also set to the other encoding. See <u>String#encode</u> for the various forms of transcoding, and the <u>Encoding::Converter</u> class for additional control over the transcoding process.

```
string
#=> "R\u00E9sum\u00E9"
string.encoding
#=> #<Encoding:UTF-8>
string = string.encode!(Encoding::IS0_8859_1)
#=> "R\xE9sum\xE9"
string.encoding
```



Þ

# **Script encoding**

All Ruby script code has an associated <u>Encoding</u> which any <u>String</u> literal created in the source code will be associated to.

The default script encoding is Encoding::UTF-8 after v2.0, but it can be changed by a magic comment on the first line of the source code file (or second line, if there is a shebang line on the first). The comment must contain the word coding Or encoding, followed by a colon, space and the Encoding name or alias:

```
# encoding: UTF-8
"some string".encoding
#=> #<Encoding:UTF-8>
```

The \_\_ENCODING\_\_ keyword returns the script encoding of the file which the keyword is written:



ruby -κ will change the default locale encoding, but this is not recommended. Ruby source files should declare its script encoding by a magic comment even when they only depend on US-ASCII strings or regular expressions.

# Locale encoding

The default encoding of the environment. Usually derived from locale.

see Encoding.locale\_charmap, <u>::find('locale')</u>

# **Filesystem encoding**

The default encoding of strings from the filesystem of the environment. This is used for strings of file names or paths.

see ::find('filesystem')

# **External encoding**

Each IO object has an external encoding which indicates the encoding that Ruby will use to read its data. By default Ruby sets the external encoding of an IO object to the default external encoding. The default external encoding is set by locale encoding or the interpreter -E option. ::default\_external returns the current value of the external encoding.



The default external encoding may also be set through <u>::default\_external=</u>, but you should not do this as strings created before and after the change will have inconsistent encodings. Instead use ruby -E to invoke ruby with the correct external encoding.

When you know that the actual encoding of the data of an <u>IO</u> object is not the default external encoding, you can reset its external encoding with <u>IO#set\_encoding</u> or set it at <u>IO</u> object

creation (see <u>IO.new</u> options).

# **Internal encoding**

To process the data of an IO object which has an encoding different from its external encoding, you can set its internal encoding. Ruby will use this internal encoding to transcode the data when it is read from the IO object.

Conversely, when data is written to the <u>IO</u> object it is transcoded from the internal encoding to the external encoding of the <u>IO</u> object.

The internal encoding of an <u>IO</u> object can be set with <u>IO#set\_encoding</u> or at <u>IO</u> object creation (see <u>IO.new</u> options).

The internal encoding is optional and when not set, the Ruby default internal encoding is used. If not explicitly set this default internal encoding is nil meaning that by default, no transcoding occurs.

The default internal encoding can be set with the interpreter option -E. <u>::default\_internal</u> returns the current internal encoding.

```
$ ruby -e 'p Encoding.default_internal'
nil
$ ruby -E ISO-8859-1:UTF-8 -e "p [Encoding.default_internal]"
[#<Encoding.default_internal]"
[#<Encoding:ISO-8859-1>, #<Encoding:UTF-8>]
```

The default internal encoding may also be set through <u>::default\_internal=</u>, but you should not do this as strings created before and after the change will have inconsistent encodings. Instead use ruby -E to invoke ruby with the correct internal encoding.

# **IO** encoding example

In the following example a UTF-8 encoded string "Ru00E9sumu00E9" is transcoded for output to ISO-8859-1 encoding, then read back in and transcoded to UTF-8:

```
string = "R\u00E9sum\u00E9"
open("transcoded.txt", "w:IS0-8859-1") do |io
io.write(string)
end
puts "raw text:"
p File.binread("transcoded.txt")
puts
open("transcoded.txt", "r:IS0-8859-1:UTF-8")
puts "transcoded text:"
p io.read
end
```

While writing the file, the internal encoding is not specified as it is only necessary for reading. While reading the file both the internal and external encoding must be specified to obtain the correct result.

```
$ ruby t.rb
raw text:
"R\xE9sum\xE9"
transcoded text:
"R\u00E9sum\u00E9"
```

## **In Files**

encoding.c

transcode.c

# Parent

Object

# **Public Class Methods**

# aliases -> {"alias1" => "orig1", "alias2" $\rightarrow$ $\odot$ "orig2", ...}

Returns the hash of available encoding alias and original encoding name.



## $\odot$ compatible?(obj1, obj2) $\rightarrow$ enc or nil

Checks the compatibility of two objects.

If the objects are both strings they are compatible when they are concatenatable. The encoding of the concatenated string will be returned if they are compatible, nil if they are not.



If the objects are non-strings their encodings are compatible when they have an encoding and:

- Either encoding is US-ASCII compatible
- One of the encodings is a 7-bit encoding

#### ogeneration in the second second

Returns default external encoding.

The default external encoding is used by default for strings created from the following locations:

- CSV
- File data read from disk
- SDBM
- StringIO
- Zlib::GzipReader
- Zlib::GzipWriter
- String#inspect
- Regexp#inspect

While strings created from these locations will have this encoding, the encoding may not be valid. Be sure to check String#valid\_encoding?.

File data written to disk will be transcoded to the default external encoding when written.

The default external encoding is initialized by the locale or -E option.

#### default\_external = enc

Sets default external encoding. You should not set ::default\_external in ruby code as strings created before changing the value may have a different encoding from strings created after the value was changed., instead you should use ruby -E to invoke ruby with the correct default\_external.

See <u>::default\_external</u> for information on how the default external encoding is used.

#### ogeneration internal → enc

Returns default internal encoding. Strings will be transcoded to the default internal encoding in the following places if the default internal encoding is not nil:

- CSV
- Etc.sysconfdir and Etc.systmpdir
- File data read from disk
- File names from Dir
- Integer#chr
- String#inspect and Regexp#inspect
- Strings returned from Readline
- Strings returned from SDBM
- Time#zone
- Solution Values from ENV
- Values in ARGV including \$PROGRAM\_NAME

Additionally <u>String#encode</u> and <u>String#encode!</u> use the default internal encoding if no encoding is given.

The locale encoding (\_\_ENCODING\_\_), not ::default\_internal, is used as the encoding of created strings.

::default\_internal is initialized by the source file's internal\_encoding or -E option.

#### ofault\_internal = enc or nil

Sets default internal encoding or removes default internal encoding when passed nil. You should not set ::default\_internal in ruby code as strings created before changing the value may have a different encoding from strings created after the change. Instead you should use ruby -E to invoke ruby with the correct default\_internal.

See ::default\_internal for information on how the default internal encoding is used.

#### on find(string) → enc

Search the encoding with specified *name*. *name* should be a string.

```
Encoding.find("US-ASCII") #=> #<Encoding:US-ASCI
]
```

Names which this method accept are encoding names and aliases including following special aliases

```
"external"
```

default external encoding

```
"internal"
```

default internal encoding

```
"locale"
```

locale encoding

"filesystem" filesystem encoding

An <u>ArgumentError</u> is raised when no encoding with *name*. Only Encoding.find("internal") however returns nil when no encoding named "internal", in other words, when Ruby has no default internal encoding.
## $_{\odot}$ list $\rightarrow$ [enc1, enc2, ...]

Returns the list of loaded encodings.



### $_{\odot}$ name\_list → ["enc1", "enc2", ...]

Returns the list of available encoding names.

Encoding.name\_list
#=> ["US-ASCII", "ASCII-8BIT", "UTF-8",
 "ISO-8859-1", "Shift\_JIS", "EUC-JP",
 "Windows-31J",
 "BINARY", "CP932", "eucJP"]

# **Public Instance Methods**

ascii\_compatible? → true or false Returns whether ASCII-compatible or not.

Encoding::UTF\_8.ascii\_compatible? #=> true
Encoding::UTF\_16BE.ascii\_compatible? #=> false

## o dummy? → true or false

Returns true for dummy encodings. A dummy encoding is an encoding for which character handling is not properly implemented. It is used for stateful encodings.



### $\odot$ inspect $\rightarrow$ string

Returns a string which represents the encoding for programmers.

E E	ncoding::UTF_8.inspect	
•		► International

### $\odot$ name $\rightarrow$ string

#### os to\_s → string

Returns the name of the encoding.

Encoding::UTF\_8.name #=> "UTF-8"

#### on names → array

Returns the list of name and aliases of the encoding.



# replicate(name) → encoding

Returns a replicated encoding of *enc* whose name is *name*. The new encoding should have the same byte structure of *enc*. If *name* is used by another encoding, raise ArgumentError.

#### mame → string

#### oto\_s → string

Returns the name of the encoding.

Encoding::UTF\_8.name

#=> "UTF-8

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class Encoding::CompatibilityErrc

Raised by <u>Encoding</u> and <u>String</u> methods when the source encoding is incompatible with the target encoding.

# In Files

encoding.c

# Parent

EncodingError

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class Encoding::Converter

# In Files

encoding.c

# Parent

Data

# Constants

#### AFTER\_OUTPUT

### AFTER\_OUTPUT

Stop converting after some output is complete but before all of the input was consumed. See **#primitive\_convert** for an example.

#### CRLF\_NEWLINE\_DECORATOR

#### CRLF\_NEWLINE\_DECORATOR

Decorator for converting LF to CRLF

#### **CR\_NEWLINE\_DECORATOR**

CR\_NEWLINE\_DECORATOR

Decorator for converting LF to CR

#### INVALID\_MASK

#### INVALID\_MASK

Mask for invalid byte sequences

#### INVALID\_REPLACE

#### INVALID\_REPLACE

Replace invalid byte sequences

#### PARTIAL\_INPUT

#### PARTIAL\_INPUT

Indicates the source may be part of a larger string. See **#primitive\_convert** for an example.

#### UNDEF\_HEX\_CHARREF

#### UNDEF\_HEX\_CHARREF

Replace byte sequences that are undefined in the destination encoding with an XML hexadecimal character reference. This is valid for XML conversion.

#### UNDEF\_MASK

#### UNDEF\_MASK

Mask for a valid character in the source encoding but no related character(s) in destination encoding.

#### UNDEF\_REPLACE

#### UNDEF\_REPLACE

Replace byte sequences that are undefined in the destination encoding.

#### UNIVERSAL\_NEWLINE\_DECORATOR

#### UNIVERSAL\_NEWLINE\_DECORATOR

Decorator for converting CRLF and CR to LF

#### XML\_ATTR\_CONTENT\_DECORATOR

XML\_ATTR\_CONTENT\_DECORATOR

Escape as XML AttValue

#### XML\_ATTR\_QUOTE\_DECORATOR

XML\_ATTR\_QUOTE\_DECORATOR

Escape as XML AttValue

#### XML\_TEXT\_DECORATOR

XML\_TEXT\_DECORATOR

Escape as XML CharData

# **Public Class Methods**

Returns the corresponding ASCII compatible encoding.

Returns nil if the argument is an ASCII compatible encoding.

"corresponding ASCII compatible encoding" is an ASCII compatible encoding which can represents exactly the same characters as the given ASCII incompatible encoding. So, no conversion undefined error occurs when converting between the two encodings.



Encoding::Converter.new(convpath)

possible options elements:

hash form:				
:invalid => nil	# raise error on in			
:invalid => :replace	# replace invalid by			
:undef => nil	# raise error on und			
:undef => :replace	<pre># replace undefined</pre>			
:replace => string	<pre># replacement string</pre>			
:newline => :universal	# decorator for conv			
:newline => :crlf	# decorator for conv			
:newline => :cr	# decorator for conv			
:universal_newline => true	# decorator for conv			
:crlf_newline => true	# decorator for conv			
:cr_newline => true	# decorator for conv			
:xml => :text	# escape as XML Chai			
:xml => :attr	<pre># escape as XML Att\</pre>			
integer form:				
Encoding::Converter::INVALID_REPLACE				
Encoding::Converter::UNDEF_REPLACE				
Encoding::Converter::UNDEF_HEX_CHARREF				
Encoding::Converter::UNIVERSAL_NEWLINE_DECORAT(				
Encoding::Converter::CRLF_NEWLINE_DECORATOR				
Encoding::Converter::CR_NEWLINE_DECORATOR				
Encoding::Converter::XML_TEXT_DECORATOR				
Encoding::Converter::XML_A	TTR_CONTENT_DECORATOR			
Encoding::Converter::XML_A	TTR_QUOTE_DECORATOR			
	Þ			

::new creates an instance of Encoding::Converter.

Source\_encoding and <u>#destination\_encoding</u> should be a string or <u>Encoding</u> object.

opt should be nil, a hash or an integer.

convpath should be an array. convpath may contain

- two-element arrays which contain encodings or encoding names, or
- strings representing decorator names.

::new optionally takes an option. The option should be a hash or an integer. The option hash can contain :invalid => nil, etc. The option integer should be logical-or of constants such as Encoding::Converter::INVALID\_REPLACE, etc.

#### :invalid => nil

Raise error on invalid byte sequence. This is a default behavior.

#### :invalid => :replace

Replace invalid byte sequence by replacement string.

#### :undef => nil

Raise an error if a character in <u>#source\_encoding</u> is not defined in destination\_encoding. This is a default behavior.

#### :undef => :replace

Replace undefined character in <u>#destination\_encoding</u> with replacement string.

#### :replace => string

Specify the replacement string. If not

specified, "uFFFD" is used for Unicode encodings and "?" for others.

:universal\_newline => true Convert CRLF and CR to LF.

:crlf\_newline => true Convert LF to CRLF.

:cr\_newline => true Convert LF to CR.

```
:xml => :text
```

Escape as XML CharData. This form can be used as a HTML 4.0 #PCDATA.

- '&' -> '&'
- □ '<' -> '&lt;'
- □ '>' -> '>'
- undefined characters in <u>#destination\_encoding</u> -> hexadecimal CharRef such as &xHH;

#### :xml => :attr

Escape as XML AttValue. The converted result is quoted as "...". This form can be used as a HTML 4.0 attribute value.

'&' -> '&'
 '<' -> '&lt;'
 '>' -> '&gt;'
 '"' -> '&quot;'
 undefined characters in #destination\_encoding ->

hexadecimal CharRef such as &xHH;

Examples:



Encoding::Converter.search\_convpath(source\_ rightarrow destination\_encoding)  $\rightarrow$  ary

Encoding::Converter.search\_convpath(source\_ @ destination\_encoding, opt)  $\rightarrow$  ary

Returns a conversion path.





# **Public Instance Methods**

### $_{\odot}$ ec == other $\rightarrow$ true or false

## convert(source\_string) → destination\_string

Convert source\_string and return destination\_string.

source\_string is assumed as a part of source. i.e. :partial\_input=>true is specified internally. finish method should be used last.



If a conversion error occur,

Encoding::UndefinedConversionError or Encoding::InvalidByteSequenceError is raised. #convert doesn't supply methods to recover or restart from these exceptions. When you want to handle these conversion errors, use #primitive\_convert.

#### $\odot$ convpath $\rightarrow$ ary

Returns the conversion path of ec.

The result is an array of conversions.



Each element of the array is a pair of encodings or a string. A pair means an encoding conversion. A string means a decorator.

In the above example, [#<Encoding:ISO-8859-1>, # <Encoding:UTF-8>] means a converter from ISO-8859-1 to UTF-8. "crlf\_newline" means newline converter from LF to CRLF.

### 

Returns the destination encoding as an **Encoding** object.

## inish → string

Finishes the converter. It returns the last part of the converted string.



# insert\_output(string) → nil

Inserts string into the encoding converter. The string will be converted to the destination encoding and

output on later conversions.

If the destination encoding is stateful, string is converted according to the state and the state is updated.

This method should be used only when a conversion error occurs.

```
ec = Encoding::Converter.new("utf-8", "iso-8859-
src = "HIRAGANA LETTER A is u{3042}."
dst = ""
p ec.primitive_convert(src, dst)
puts "[#{dst.dump}, #{src.dump}]"
ec.insert_output("<err>")
p ec.primitive_convert(src, dst)
puts "[#{dst.dump}, #{src.dump}]"
ec = Encoding::Converter.new("utf-8",
                                       "iso-2022
src = "\u{306F 3041 3068 2661 3002}"
dst = ""
p ec.primitive_convert(src, dst)
puts "[#{dst.dump}, #{src.dump}]"
ec.insert_output "?"
p ec.primitive_convert(src, dst)
puts "[#{dst.dump}, #{src.dump}]"
4
```

### inspect → string

Returns a printable version of ec



### last\_error → exception or nil

Returns an exception object for the last conversion. Returns nil if the last conversion did not produce an error.

"error" means that

Encoding::InvalidByteSequenceError and Encoding::UndefinedConversionError for #convert and :invalid\_byte\_sequence, :incomplete\_input and :undefined\_conversion for #primitive\_convert.



primitive\_convert(source\_buffer,
destination\_buffer) → symbol primitive\_convert(source\_buffer, destination\_buffer, destination\_byteoffset)
→ symbol primitive\_convert(source\_buffer, destination\_buffer, destination\_byteoffset,
destination\_bytesize) → symbol primitive\_convert(source\_buffer, destination\_buffer, destination\_byteoffset,
destination\_bytesize, opt) → symbol possible opt elements:

hash form:				
:partial_input => true				
:after_output => true				
integer form:				
Encoding::Converter::PARTIAL_INPUT				
Encoding::Converter::AFTER_OUTPUT				
	•			

possible results:

```
:invalid_byte_sequence
:incomplete_input
:undefined_conversion
:after_output
```

#### :destination\_buffer\_full :source\_buffer\_empty :finished

#primitive\_convert converts source\_buffer into
destination\_buffer.

source\_buffer should be a string or nil. nil means an empty string.

destination\_buffer should be a string.

destination\_byteoffset should be an integer or nil. nil means the end of destination\_buffer. If it is omitted, nil is assumed.

destination\_bytesize should be an integer or nil. nil means unlimited. If it is omitted, nil is assumed.

opt should be nil, a hash or an integer. nil means no flags. If it is omitted, nil is assumed.

#primitive\_convert converts the content of source\_buffer from beginning and store the result into destination\_buffer.

destination\_byteoffset and destination\_bytesize specify the region which the converted result is stored. destination\_byteoffset specifies the start position in destination\_buffer in bytes. If destination\_byteoffset is nil, destination\_buffer.bytesize is used for appending the result. destination\_bytesize specifies maximum number of bytes. If destination\_bytesize is nil, destination size is unlimited. After conversion, destination\_buffer is resized to destination\_byteoffset + actually produced number of bytes. Also destination\_buffer's encoding is set to destination\_encoding.

#primitive\_convert drops the converted part of
source\_buffer. the dropped part is converted in
destination\_buffer or buffered in Encoding::Converter

object.

#primitive\_convert stops conversion when one of following condition met.

- invalid byte sequence found in source buffer (:invalid\_byte\_sequence) primitive\_errinfo and last\_error methods returns the detail of the error.
- unexpected end of source buffer (:incomplete\_input) this occur only when :partial\_input is not specified. primitive\_errinfo and last\_error methods returns the detail of the error.
- character not representable in output encoding (:undefined\_conversion) primitive\_errinfo and last\_error methods returns the detail of the error.
- after some output is generated, before input is done (:after\_output) this occur only when :after\_output is specified.
- destination buffer is full (:destination\_buffer\_full) this occur only when destination\_bytesize is non-nil.
- source buffer is empty (:source\_buffer\_empty) this occur only when :partial\_input is specified.
- conversion is finished (:finished)

#### example:

```
ec = Encoding::Converter.new("UTF-8", "UTF-16BE")
ret = ec.primitive_convert(src="pi", dst="", nil,
p [ret, src, dst] #=> [:finished, "", "\x00p\x00]
ec = Encoding::Converter.new("UTF-8", "UTF-16BE")
ret = ec.primitive_convert(src="pi", dst="", nil,
p [ret, src, dst] #=> [:destination_buffer_full,
ret = ec.primitive_convert(src, dst="", nil, 1)
```



### ◎ primitive\_errinfo → array

<u>#primitive\_errinfo</u> returns important information regarding the last error as a 5-element array:



result is the last result of primitive\_convert.

Other elements are only meaningful when result is :invalid\_byte\_sequence, :incomplete\_input or :undefined\_conversion.

enc1 and enc2 indicate a conversion step as a pair of strings. For example, a converter from EUC-JP to ISO-8859-1 converts a string as follows: EUC-JP -> UTF-8 -> ISO-8859-1. So [enc1, enc2] is either ["EUC-JP", "UTF-8"] or ["UTF-8", "ISO-8859-1"].

error\_bytes and readagain\_bytes indicate the byte sequences which caused the error. error\_bytes is discarded portion. readagain\_bytes is buffered portion which is read again on next conversion.

Example:





#### putback(p1 = v1)

call-seq

ec.putback	-> string
<pre>ec.putback(max_numbytes)</pre>	-> string

Put back the bytes which will be converted.

The bytes are caused by invalid\_byte\_sequence error. When invalid\_byte\_sequence error, some bytes are discarded and some bytes are buffered to be converted later. The latter bytes can be put back. It can be observed by

Encoding::InvalidByteSequenceError#readagain\_bytes and #primitive\_errinfo.



#### or replacement → string

Returns the replacement string.



### replacement = string

Sets the replacement string.



source\_encoding → encoding

Returns the source encoding as an **Encoding** object.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class Encoding::ConverterNotFou

Raised by transcoding methods when a named encoding does not correspond with a known converter.

# In Files

encoding.c

# Parent

rb\_eEncodingError

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class Encoding::InvalidByteSeque

Raised by <u>Encoding</u> and <u>String</u> methods when the string being transcoded contains a byte invalid for the either the source or target encoding.

# In Files

encoding.c

# Parent

rb\_eEncodingError

# **Public Instance Methods**

### $\odot$ destination\_encoding $\rightarrow$ string

Returns the destination encoding as an encoding object.

### $_{\odot}$ destination\_encoding\_name $\rightarrow$ string

Returns the destination encoding name as a string.

#### error\_bytes → string Returns the discarded bytes when Encoding::InvalidByteSequenceError occurs.



### incomplete\_input? → true or false

Returns true if the invalid byte sequence error is caused by premature end of string.



## $_{\odot}$ readagain\_bytes $\rightarrow$ string

Returns the bytes to be read again when Encoding::InvalidByteSequenceError occurs.

#### source\_encoding → encoding

Returns the source encoding as an encoding object. Note that the result may not be equal to the source encoding of the encoding converter if the conversion has multiple steps.



#### source\_encoding\_name → string

Returns the source encoding name as a string.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class Encoding::UndefinedConver

Raised by <u>Encoding</u> and <u>String</u> methods when a transcoding operation fails.

# **In Files**

encoding.c

# Parent

rb\_eEncodingError

# **Public Instance Methods**

#### 

Returns the destination encoding as an encoding object.

#### $\odot$ destination\_encoding\_name $\rightarrow$ string

Returns the destination encoding name as a string.

### $\odot$ error\_char $\rightarrow$ string

Returns the one-character string which cause Encoding::UndefinedConversionError.

ec = Encoding::Converter.new("ISO-8859-1", "EUCbegin



#### source\_encoding → encoding

Returns the source encoding as an encoding object.

Note that the result may not be equal to the source encoding of the encoding converter if the conversion has multiple steps.



### source\_encoding\_name → string

Returns the source encoding name as a string.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class EncodingError

EncodingError is the base class for encoding errors.

# **In Files**

error.c

# Parent

StandardError

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# module Enumerable

The Enumerable mixin provides collection classes with several traversal and searching methods, and with the ability to sort. The class must provide a method each, which yields successive members of the collection. If Enumerable#max, #min, Or #sort is used, the objects in the collection must also implement a meaningful <=> operator, as these methods rely on an ordering between members of the collection.

# **In Files**

📄 enum.c

enumerator.c

# **Public Instance Methods**

# $_{\odot}$ all? [{ |obj| block } ] → true or false

Passes each element of the collection to the given block. The method returns true if the block never returns false or nil. If the block is not given, Ruby adds an implicit block of { |obj| obj } which will cause <u>all?</u> to return true when none of the collection members are false Or nil.

```
%w[ant bear cat].all? { |word| word.length >= 3
%w[ant bear cat].all? { |word| word.length >= 4
[nil, true, 99].all?
```

## $_{\odot}$ any? [{ |obj| block }] → true or false

4

Passes each element of the collection to the given block. The method returns true if the block ever returns a value other than false or nil. If the block is not given, Ruby adds an implicit block of { |obj| obj } that will cause <u>any?</u> to return true if at least one of the collection members is not false or nil.



# chunk { |elt| ... } → an\_enumerator chunk(initial\_state) { |elt, state| ... } → an\_enumerator (deprecated)

Enumerates over the items, chunking them together based on the return value of the block.

Consecutive elements which return the same block value are chunked together.

For example, consecutive even numbers and odd numbers can be chunked as follows.



This method is especially useful for sorted series of elements. The following example counts words for

each initial letter.



The following key values have special meaning:

- nil and :\_separator specifies that the elements should be dropped.
- :\_alone specifies that the element should be chunked by itself.

Any other symbols that begin with an underscore will raise an error:



nil and :\_separator can be used to ignore some elements.

For example, the sequence of hyphens in svn log can be eliminated as follows:





Paragraphs separated by empty lines can be parsed as follows:



:\_alone can be used to force items into their own chunk. For example, you can put lines that contain a URL by themselves, and chunk the rest of the lines together, like this:



- $\odot$  collect { |obj| block }  $\rightarrow$  array
- $_{\odot}$  map { |obj| block }  $\rightarrow$  array
- collect → an\_enumerator
- map → an\_enumerator

Returns a new array with the results of running *block* once for every element in *enum*.

If no block is given, an enumerator is returned instead.



- $\oplus$  flat\_map { |obj| block }  $\rightarrow$  array
- $\odot$  collect\_concat { |obj| block }  $\rightarrow$  array
- flat\_map → an\_enumerator

#### collect\_concat → an\_enumerator

Returns a new array with the concatenated results of running *block* once for every element in *enum*.

If no block is given, an enumerator is returned instead.



### $\odot$ count $\rightarrow$ int

#### $\odot$ count(item) $\rightarrow$ int

### ount { |obj| block } → int

Returns the number of items in enum through enumeration. If an argument is given, the number of items in enum that are equal to item are counted. If a block is given, it counts the number of elements yielding a true value.



# rightarrow cycle(n=nil) { |obj| block } → nil rightarrow cycle(n=nil) → an\_enumerator

Calls *block* for each element of *enum* repeatedly *n* times or forever if none or nil is given. If a non-positive number is given or the collection is empty,

does nothing. Returns nil if the loop has finished without getting interrupted.

<u>**#cycle**</u> saves elements in an internal array so changes to *enum* after the first pass have no effect.

If no block is given, an enumerator is returned instead.



```
↔ detect(ifnone = nil) { |obj| block } → obj or nil 
↔ find(ifnone = nil) { |obj| block } → obj or nil
```

- $\otimes$  detect(ifnone = nil)  $\rightarrow$  an enumerator
- $_{\odot}$  find(ifnone = nil) → an\_enumerator

Passes each entry in *enum* to *block*. Returns the first for which *block* is not false. If no object matches, calls *ifnone* and returns its result when it is specified, or returns nil otherwise.

If no block is given, an enumerator is returned instead.



### ombody drop(n) → array

Drops first n elements from *enum*, and returns rest elements in an array.



## $rightarrow drop_while { |arr| block } → array$ $<math>rightarrow drop_while → an_enumerator$

Drops elements up to, but not including, the first element for which the block returns nil or false and returns an array containing the remaining elements.

If no block is given, an enumerator is returned instead.

```
a = [1, 2, 3, 4, 5, 0]
a.drop_while { |i| i < 3 } #=> [3, 4, 5, 0]
```

### each\_cons(n) { ... } → nil each\_cons(n) → an\_enumerator

Iterates the given block for each array of consecutive <n> elements. If no block is given, returns an enumerator.

e.g.:



## each\_entry { |obj| block } → enum each\_entry → an\_enumerator

Calls *block* once for each element in self, passing that element as a parameter, converting multiple values from yield to an array.

If no block is given, an enumerator is returned

instead.

```
class Foo
  include Enumerable
  def each
    yield 1
    yield 1, 2
    yield
  end
end
Foo.new.each_entry{ |o| p o }
```

produces:

1 [1, 2] nil

# $_{\odot}$ each\_slice(n) { ... } → nil $_{\odot}$ each\_slice(n) → an\_enumerator

Iterates the given block for each slice of <n> elements. If no block is given, returns an enumerator.



each\_with\_index(\*args) { |obj, i| block }  $\rightarrow$   $_{\odot}$  enum

 each\_with\_index(\*args)  $\rightarrow$  an\_enumerator Calls *block* with two arguments, the item and its index, for each item in *enum*. Given arguments are passed through to each().

If no block is given, an enumerator is returned instead.


# each\_with\_object(obj) { |(\*args), memo\_obj| $_{\odot}$ ... } $_{\rightarrow}$ obj

#### $\otimes$ each\_with\_object(obj) $\rightarrow$ an\_enumerator

Iterates the given block for each element with an arbitrary object given, and returns the initially given object.

If no block is given, returns an enumerator.

## ominimistic ominimistic ominimistic ominimistic ominimistic ominimistic ominimistic ominimistic ominimistic ominimistration of the second structure of the second structure

Returns an array containing the items in enum.



obj or nil
 obj block } → obj or nil

- ↔ find(ifnone = nil) { |obj| block } → obj or nil
- $\otimes$  detect(ifnone = nil)  $\rightarrow$  an\_enumerator

```
_{\odot} find(ifnone = nil) \rightarrow an_enumerator
```

Passes each entry in enum to block. Returns the first

for which *block* is not false. If no object matches, calls *ifnone* and returns its result when it is specified, or returns nil otherwise.

If no block is given, an enumerator is returned instead.



```
_{\odot} find_all { |obj| block } → array
```

- $\otimes$  select { |obj| block }  $\rightarrow$  array
- ind\_all → an\_enumerator
- select → an\_enumerator

Returns an array containing all elements of enum for which the given block returns a true value.

If no block is given, an Enumerator is returned instead.



See also #reject.

## find\_index(value) → int or nil

ind\_index { |obj| block } → int or nil

#### ind\_index → an\_enumerator

Compares each entry in *enum* with *value* or passes to *block*. Returns the index for the first for which the evaluated value is non-false. If no object matches, returns nil

If neither block nor argument is given, an enumerator is returned instead.



## Is first → obj or nil Is first(n) → an array

Returns the first element, or the first n elements, of the enumerable. If the enumerable is empty, the first form returns nil, and the second form returns an empty array.



- $_{\odot}$  flat\_map { |obj| block }  $\rightarrow$  array
- $\odot$  collect\_concat { |obj| block }  $\rightarrow$  array
- $\otimes$  flat\_map  $\rightarrow$  an\_enumerator

```
collect_concat → an_enumerator
```

Returns a new array with the concatenated results of running *block* once for every element in *enum*.

If no block is given, an enumerator is returned instead.



## ⊚ grep(pattern) → array ⊚ grep(pattern) { |obj| block } → array

Returns an array of every element in *enum* for which Pattern === element. If the optional *block* is supplied,

each matching element is passed to it, and the block's result is stored in the output array.



## Some group\_by { |obj| block} → a\_hash group\_by → an\_enumerator

Groups the collection by result of the block. Returns a hash where the keys are the evaluated result from the block and the values are arrays of elements in the collection that correspond to the key.

If no block is given an enumerator is returned.



## rightarrow include?(obj) → true or false rightarrow member?(obj) → true or false

Returns true if any member of *enum* equals *obj*. Equality is tested using ==.



```
inject(initial, sym) → obj
inject(sym) → obj
inject(initial) { |memo, obj| block } → obj
inject { |memo, obj| block } → obj
reduce(initial, sym) → obj
```

### obj reduce(sym) → obj

## obj reduce(initial) { |memo, obj| block } → obj obj reduce { |memo, obj| block } → obj

Combines all elements of *enum* by applying a binary operation, specified by a block or a symbol that names a method or operator.

If you specify a block, then for each element in *enum* the block is passed an accumulator value (*memo*) and the element. If you specify a symbol instead, then each element in the collection will be passed to the named method of *memo*. In either case, the result becomes the new value for *memo*. At the end of the iteration, the final value of *memo* is the return value for the method.

If you do not explicitly specify an *initial* value for *memo*, then the first element of collection is used as the initial value of *memo*.



#### ⊚ lazy → lazy\_enumerator

Returns a lazy enumerator, whose methods map/collect, flat\_map/collect\_concat, select/find\_all, reject, grep, zip, take, <u>#take\_while</u>, drop, and <u>#drop\_while</u> enumerate values only on an as-needed basis. However, if a block is given to zip, values are enumerated immediately.

### Example

The following program finds pythagorean triples:





#### $rightarrow max { |a, b| block } → obj$ <math>rightarrow max(n) → obj $rightarrow max(n) { |a,b| block } → obj$

Returns the object in *enum* with the maximum value. The first form assumes all objects implement Comparable; the second uses the block to return *a* <=> *b*.



If the n argument is given, maximum n elements are returned as an array.



```
    max_by {[obj] block } → obj
    max_by → an_enumerator
    max_by(n) {[obj] block } → obj
    max_by(n) → an_enumerator
    Returns the object in enum that gives the maximum value from the given block.
    If no block is given, an enumerator is returned instead.
    a = %w(albatross dog horse)
a.max_by { |x| x.length } #> "albatross"
    If the n argument is given, minimum n elements are returned as an array.
    a = %w[albatross dog horse]
a.max_by(2) { |x| x.length } #> ["albatross", "h
```

enum.max\_by(n) can be used to implement weighted random sampling. Following example implements and use Enumerable#wsample.

F



### rightarrow include?(obj) → true or false rightarrow member?(obj) → true or false

Returns true if any member of *enum* equals *obj*. Equality is tested using ==.



min → obj
 min {| a,b | block } → obj
 min(n) → array
 min(n) {| a,b | block } → array
 Returns the object in *enum* with the minimum value. The first form assumes all objects implement comparable; the second uses the block to return a <=> b.
 a = %w(albatross dog horse)

 a.min
 a.min { |a, b| a.length <=> b.length } #=> "al
 "do

If the n argument is given, minimum n elements are returned as an array.



```
    min_by {|obj| block } → obj
    min_by → an_enumerator
    min_by(n) {|obj| block } → array
    min_by(n) → an_enumerator
```

Returns the object in *enum* that gives the minimum value from the given block.

If no block is given, an enumerator is returned instead.



If the n argument is given, minimum n elements are returned as an array.



### 

Returns two elements array which contains the minimum and the maximum value in the enumerable. The first form assumes all objects implement comparable; the second uses the block to return  $a \ll b$ .



### rightarrow minmax\_by { |obj| block } → [min, max] rightarrow minmax\_by → an\_enumerator

Returns a two element array containing the objects in *enum* that correspond to the minimum and maximum values respectively from the given block.

If no block is given, an enumerator is returned instead.



#### on none? [{ |obj| block }] → true or false

Passes each element of the collection to the given block. The method returns true if the block never returns true for all elements. If the block is not given, none? will return true only if none of the collection members is true.



## ome one? [{ |obj| block }] → true or false

Passes each element of the collection to the given block. The method returns true if the block returns true exactly once. If the block is not given, one? will return true only if exactly one of the collection members is true.



### partition { |obj| block } $\rightarrow$ [ true\_array, $_{\odot}$ false\_array ]

#### ◎ partition → an\_enumerator

Returns two arrays, the first containing the elements of *enum* for which the block evaluates to true, the

second containing the rest.

If no block is given, an enumerator is returned instead.



```
_{\odot} inject(initial, sym) \rightarrow obj
inject(sym) → obj
_{\odot} inject(initial) { |memo, obj| block } \rightarrow obj
inject { |memo, obj| block } → obj
reduce(initial, sym) \rightarrow obj
areduce(sym) → obj
\otimes reduce(initial) { |memo, obj| block } \rightarrow obj
\odot reduce { |memo, obj| block } \rightarrow obj
  Combines all elements of enum by applying a binary
  operation, specified by a block or a symbol that
  names a method or operator.
  If you specify a block, then for each element in enum
  the block is passed an accumulator value (memo)
  and the element. If you specify a symbol instead,
  then each element in the collection will be passed to
  the named method of memo. In either case, the result
```

for the method. If you do not explicitly specify an *initial* value for *memo*, then the first element of collection is used as the initial value of *memo*.

becomes the new value for *memo*. At the end of the iteration, the final value of *memo* is the return value

```
# Sum some numbers
(5..10).reduce(:+)
# Same using a block and inject
(5..10).inject { |sum, n| sum + n }
# Multiply some numbers
```



### $reject { |obj| block } → array$ $<math>reject → an_enumerator$

Returns an array for all elements of enum for which the given block returns false.

If no block is given, an <u>Enumerator</u> is returned instead.



See also <u>#find\_all</u>.

## $_{\odot}$ reverse\_each(\*args) { |item| block } → enum $_{\odot}$ reverse\_each(\*args) → an\_enumerator

Builds a temporary array and traverses that array in reverse order.

If no block is given, an enumerator is returned instead.



```
_{\odot} find_all { |obj| block } → array
```

```
_{\odot} select { |obj| block } \rightarrow array
```

- ind\_all → an\_enumerator
- select → an\_enumerator

Returns an array containing all elements of enum for which the given block returns a true value.

If no block is given, an Enumerator is returned instead.



See also #reject.

## slice\_after(pattern) → an\_enumerator slice\_after { |elt| bool } → an\_enumerator

Creates an enumerator for each chunked elements. The ends of chunks are defined by *pattern* and the block.

If *pattern* === *elt* returns true or the block returns true for the element, the element is end of a chunk.

The === and *block* is called from the first element to the last element of *enum*.

The result enumerator yields the chunked elements as an array. So each method can be called as follows:

```
enum.slice_after(pattern).each { |ary| ... }
enum.slice_after { |elt| bool }.each { |ary| ...
```

Other methods of the <u>Enumerator</u> class and Enumerable module, such as map, etc., are also usable.

For example, continuation lines (lines end with backslash) can be concatenated as follows:



Slice\_before(pattern) → an\_enumerator
 slice\_before { |elt| bool } → an\_enumerator slice\_before(initial\_state) { |elt, state| bool }
 → an enumerator (deprecated)

Creates an enumerator for each chunked elements. The beginnings of chunks are defined by *pattern* and the block.

If *pattern* === *elt* returns true or the block returns true for the element, the element is beginning of a chunk.

The === and *block* is called from the first element to the last element of *enum*. The result for the first element is ignored.

The result enumerator yields the chunked elements as an array. So each method can be called as follows:



Other methods of the <u>Enumerator</u> class and <u>Enumerable</u> module, such as map, etc., are also usable.

For example, iteration over ChangeLog entries can

be implemented as follows:



"svn proplist -R" produces multiline output for each file. They can be chunked as follows:



If the block needs to maintain state over multiple elements, local variables can be used. For example, three or more consecutive increasing numbers can be squashed as follows:



However local variables should be used carefully if the result enumerator is enumerated twice or more. The local variables should be initialized for each enumeration. Enumerator.new can be used to do it.

```
def wordwrap(words, maxwidth)
  Enumerator.new {|y|
    cols = 0
    words.slice_before { |w|
      cols += 1 if cols != 0
      cols += w.length
      if maxwidth < cols</pre>
        cols = w.length
        true
      else
        false
      end
    }.each { ws y.yield ws }
  }
end
text = (1..20).to_a.join(" ")
enum = wordwrap(text.split(/\s+/), 10)
puts "-"*10
enum.each { |ws| puts ws.join(" ") } #
puts "-"*10
enum.each { |ws| puts ws.join(" ") }
puts "-"*10
4
```

mbox contains series of mails which start with Unix From line. So each mail can be extracted by slice before Unix From line.

```
open("mbox") { |f|
  f.slice_before { |line|
    line.start_with? "From "
  }.each { |mail
    unix_from = mail.shift
    i = mail.index("\n")
    header = mail[0...i]
    body = mail[(i+1)..-1]
    body.pop if body.last == "\n"
    fields = header.slice_before { |line| !" \t"
    p unix_from
    pp fields
    pp body
  }
}
open("mbox") { |f|
  f.slice_before(emp: true) { |line, h|
    prevemp = h[:emp]
    h[:emp] = line == "\setminus n"
    prevemp && line.start_with?("From ")
  }.each { |mail|
    mail.pop if mail.last == "\n"
    pp mail
  }
4
```

## slice\_when {|elt\_before, elt\_after| bool } $\rightarrow$ $_{\textcircled{}}$ an\_enumerator

Creates an enumerator for each chunked elements. The beginnings of chunks are defined by the block.

This method split each chunk using adjacent elements, *elt\_before* and *elt\_after*, in the receiver enumerator. This method split chunks between *elt\_before* and *elt\_after* where the block returns true.

The block is called the length of the receiver enumerator minus one.

The result enumerator yields the chunked elements as an array. So each method can be called as follows:



Other methods of the <u>Enumerator</u> class and <u>Enumerable</u> module, such as to\_a, map, etc., are also usable.

For example, one-by-one increasing subsequence can be chunked as follows:



Near elements (threshold: 6) in sorted array can be chunked as follwos:



Increasing (non-decreasing) subsequence can be chunked as follows:



Adjacent evens and odds can be chunked as follows: (#chunk is another way to do it.)



Paragraphs (non-empty lines with trailing empty lines) can be chunked as follows: (See <u>#chunk</u> to ignore empty lines.)



### $_{\odot}$ sort → array $_{\odot}$ sort { |a, b| block } → array

Returns an array containing the items in *enum* sorted, either according to their own <=> method, or by using the results of the supplied block. The block should return -1, 0, or +1 depending on the comparison between *a* and *b*. As of Ruby 1.8, the method Enumerable#sort\_by implements a built-in Schwartzian Transform, useful when key computation or comparison is expensive.



#### sort\_by { |obj| block } → array sort\_by → an\_enumerator

Sorts *enum* using a set of keys generated by mapping the values in *enum* through the given block.

If no block is given, an enumerator is returned instead.



The current implementation of sort\_by generates an array of tuples containing the original collection

element and the mapped value. This makes sort\_by fairly expensive when the keysets are simple.

produces:

user Sort Sort by	system 0.180000 1.980000	total 0.000000 0.040000	real 0.180000 ( 2.020000 (	0.1 2.0
<b>↓</b>				Ð

However, consider the case where comparing the keys is a non-trivial operation. The following code sorts some files on modification time using the basic sort method.



This sort is inefficient: it generates two new File objects during every comparison. A slightly better technique is to use the Kernel#test method to generate the modification times directly.



This still generates many unnecessary Time objects. A more efficient technique is to cache the sort keys (modification times in this case) before the sort. Perl users often call this approach a Schwartzian Transform, after Randal Schwartz. We construct a temporary array, where each element is an array containing our sort key along with the filename. We sort this array, and then extract the filename from the result.



This is exactly what sort\_by does internally.

sorted =	<pre>Dir["*"].sort_by {  f  test(M, f) }</pre>
sorted	

## take(n) → array

Returns first n elements from enum.

Passes elements to the block until the block returns nil or false, then stops iterating and returns an array of all prior elements.

If no block is given, an enumerator is returned instead.

```
a = [1, 2, 3, 4, 5, 0]
a.take_while { |i| i < 3 }   #=> [1, 2]
```

```
osto_a(*args) → array
```

### ontries(\*args) → array

Returns an array containing the items in enum.



### oto\_h(\*args) → hash

Returns the result of interpreting *enum* as a list of [key, value] pairs.

%[hello world].each\_with\_index.to\_h
 # => {:hello => 0, :world => 1}

### $_{\odot}$ zip(arg, ...) → an\_array\_of\_array $_{\odot}$ zip(arg, ...) { |arr| block } → nil

Takes one element from *enum* and merges corresponding elements from each *args*. This generates a sequence of *n*-element arrays, where *n* is one more than the count of arguments. The length of the resulting sequence will be enum#size. If the size of any argument is less than enum#size, nil values are supplied. If a block is given, it is invoked for each output array, otherwise an array of arrays is returned.



Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class Enumerator

A class which allows both internal and external iteration.

An <u>Enumerator</u> can be created by the following methods.

- Kernel#to\_enum
- Kernel#enum\_for
- ::new

Most methods have two forms: a block form where the contents are evaluated for each item in the enumeration, and a non-block form which returns a new Enumerator wrapping the iteration.

```
enumerator = %w(one two three).each
puts enumerator.class # => Enumerator
enumerator.each_with_object("foo") do [item, of
puts "#{obj}: #{item}"
end
# foo: one
# foo: two
# foo: three
enum_with_obj = enumerator.each_with_object(""
puts enum_with_obj.class # => Enumerator
enum_with_obj.each do [item, obj]
puts "#{obj}: #{item}"
```



This allows you to chain Enumerators together. For example, you can map a list's elements to strings containing the index and the element as a string via:



An <u>Enumerator</u> can also be used as an external iterator. For example, <u>#next</u> returns the next value of the iterator or raises <u>StopIteration</u> if the <u>Enumerator</u> is at the end.



You can use this to implement an internal iterator as follows:

```
def ext_each(e)
  while true
    begin
    vs = e.next_values
    rescue StopIteration
        return $!.result
```

```
end
y = yield(*vs)
e.feed y
end
end
o = Object.new
def o.each
puts yield
puts yield(1)
puts yield(1, 2)
3
end
# use o.each as an internal iterator directly
puts o.each {|*x| puts x; [:b, *x] }
# => [], [:b], [1], [:b, 1], [1, 2], [:b, 1, 3]
# convert o.each to an external iterator for
# implementing an internal iterator.
puts ext_each(o.to_enum) {|*x| puts x; [:b, *;
# => [], [:b], [1], [:b, 1], [1, 2], [:b, 1, 5]
```

In Files

enumerator.c

Parent

**Object** 

## **Included Modules**

Enumerable

## **Public Class Methods**

# new(size = nil) { |yielder| ... } new(obj, method = :each, \*args)

Creates a new Enumerator object, which can be used as an Enumerable.

In the first form, iteration is defined by the given block, in which a "yielder" object, given as block parameter, can be used to yield a value by calling the yield method (aliased as +<<+):



The optional parameter can be used to specify how to calculate the size in a lazy fashion (see  $\frac{\text{#size}}{\text{size}}$ ). It can either be a value or a callable object.

In the second, deprecated, form, a generated Enumerator iterates over the given object using the given method with the given arguments passed.

Use of this form is discouraged. Use Kernel#enum\_for or Kernel#to\_enum instead.

```
e = Enumerator.new(ObjectSpace, :each_object)
    #-> ObjectSpace.enum_for(:each_object)
e.select { |obj| obj.is_a?(Class) } #=> array or
```

## **Public Instance Methods**

- $\otimes$  each { |elm| block }  $\rightarrow$  obj
- each → enum
- $_{\odot}$  each(\*appending\_args) { |elm| block }  $\rightarrow$  obj
- each(\*appending\_args) → an\_enumerator Iterates over the block according to how this <u>Enumerator</u> was constructed. If no block and no arguments are given, returns self.

#### Examples

```
"Hello, world!".scan(/\w+/)
"Hello, world!".to_enum(:scan, /\w+/).to_a
"Hello, world!".to_enum(:scan).each(/\w+/).to_a
obj = Object.new
def obj.each_arg(a, b=:b, *rest)
  yield a
  yield b
  yield rest
  :method_returned
end
enum = obj.to_enum :each_arg, :a, :x
enum.each.to a
enum.each.equal?(enum)
enum.each { |elm| elm }
enum.each(:y, :z).to_a
enum.each(:y, :z).equal?(enum)
enum.each(:y, :z) { |elm elm }
4
```

## each\_with\_index {|(\*args), idx| ... } each\_with\_index

Same as <u>#with\_index</u>, i.e. there is no starting offset.

If no block is given, a new Enumerator is returned that includes the index.

```
    each_with_object(obj) {|(*args), obj| ... }
    each_with_object(obj)
    with_object(obj) {|(*args), obj| ... }
    with_object(obj)
    Iterates the given block for each element with an arbitrary object, obj, and returns obj
```

If no block is given, returns a new Enumerator.

#### Example



### feed obj → nil

Sets the value to be returned by the next yield inside e.

If the value is not set, the yield returns nil.

This value is cleared after being yielded.





#### $\odot$ inspect $\rightarrow$ string

Creates a printable version of *e*.

#### mext → object

Returns the next object in the enumerator, and move the internal position forward. When the position reached at the end, **StopIteration** is raised.

#### Example

a = [1,2,3]		
e = a.to_enum		
p e.next		
p e.next		
p e.next		
p e.next <b>#raises</b>		

Note that enumeration sequence by next does not affect other non-external enumeration methods, unless the underlying iteration methods itself has side-effect, e.g. IO#each\_line.

#### onext\_values → array

Returns the next object as an array in the enumerator, and move the internal position forward. When the position reached at the end, <u>StopIteration</u> is raised.

This method can be used to distinguish yield and yield nil.

#### Example

<pre>o = Object.new def o.each    yield</pre>
yield 1
yield 1, 2
yield nil
yield [1, 2]
end
e = o.to_enum
p e.next_values
e = o.to_enum
p e.next

p e.next p e.next p e.next p e.next	
<pre>## yield args # yield # yield 1 # yield 1, 2 # yield nil # yield [1,</pre>	

Note that next\_values does not affect other nonexternal enumeration methods unless underlying iteration method itself has side-effect, e.g. IO#each\_line.

#### ⊚ peek → object

Returns the next object in the enumerator, but doesn't move the internal position forward. If the position is already at the end, **StopIteration** is raised.

### Example

a = [1,2,3	]	
e = a.to_enum		
p e.next		
p e.peek		
p e.peek		
p e.peek		
p e.next		
p e.next		
p e.peek		

#### ⊚ peek\_values → array

Returns the next object as an array, similar to <u>#next\_values</u>, but doesn't move the internal position forward. If the position is already at the end, <u>StopIteration</u> is raised.

Example



#### $\odot$ rewind $\rightarrow$ e

Rewinds the enumeration sequence to the beginning.

If the enclosed object responds to a "rewind" method, it is called.

#### size → int, Float::INFINITY or nil

Returns the size of the enumerator, or nil if it can't be calculated lazily.

```
(1..100).to_a.permutation(4).size # => 94109400
loop.size # => Float::INFINITY
(1..100).drop_while.size # => nil
```

## with\_index(offset = 0) {|(\*args), idx| ... } with\_index(offset = 0)

Iterates the given block for each element with an index, which starts from offset. If no block is given, returns a new Enumerator that includes the index, starting from offset

offset

the starting index to use

```
    each_with_object(obj) {|(*args), obj| ... }
    each_with_object(obj)
    with_object(obj) {|(*args), obj| ... }
    with_object(obj)
    Iterates the given block for each element with an
```

arbitrary object, obj, and returns obj

If no block is given, returns a new Enumerator.

### Example



Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class Enumerator::Generator

Lazy

**In Files** 

enumerator.c

Parent

Object

## **Included Modules**

Enumerable

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.
## class Enumerator::Lazy

Lazy

### **In Files**

enumerator.c

### Parent

Enumerator

### **Public Class Methods**

### mew(obj, size=nil) { |yielder, \*values| ... }

Creates a new <u>Lazy</u> enumerator. When the enumerator is actually enumerated (e.g. by calling force), obj will be enumerated and each value passed to the given block. The block can yield values back using yielder. For example, to create a method filter\_map in both lazy and non-lazy fashions:

```
module Enumerable
  def filter_map(&block)
    map(&block).compact
  end
end
class Enumerator::Lazy
  def filter_map
    Lazy.new(self) do [yielder, *values]
    result = yield *values
    yielder << result if result</pre>
```



### **Public Instance Methods**

🚳 chunk(\*args)

### 🏐 collect()

### collect\_concat { |obj| block } $\rightarrow$

- a\_lazy\_enumerator
- Solution flat\_map { |obj| block } → a\_lazy\_enumerator Returns a new lazy enumerator with the concatenated results of running *block* once for every element in *lazy*.



A value *x* returned by *block* is decomposed if either of the following conditions is true:



Otherwise, *x* is contained as-is in the return value.

[{a:1}, {b:2}].lazy.flat\_map {|i| i}.force
#=> [{:a=>1}, {:b=>2}]

🏐 drop(p1)

odd drop\_while()

to\_enum(method = :each, \*args) →

lazy\_enum

```
enum_for(method = :each, *args) \rightarrow
```

- lazy\_enum

Similar to Kernel#to\_enum, except it returns a lazy enumerator. This makes it easy to define <u>Enumerable</u> methods that will naturally remain lazy if called from a lazy enumerator.

For example, continuing from the example in Kernel#to\_enum:



### find\_all()

```
collect_concat { |obj| block } →

a_lazy_enumerator

flat_map { |obj| block } → a_lazy_enumerator
```

Returns a new lazy enumerator with the concatenated results of running *block* once for every element in *lazy*.



A value *x* returned by *block* is decomposed if either of the following conditions is true:



Otherwise, x is contained as-is in the return value.



🏐 grep(p1)

- 🚳 lazy()
- 🏐 map()
- 🚳 reject()
- 🚳 select()
- slice\_after(\*args)
- slice\_before(\*args)
- slice\_when(\*args)

🚳 take(p1)

take\_while()

to\_enum(method = :each, \*args) →

lazy\_enum

enum\_for(method = :each, \*args) →

lazy\_enum

to\_enum(method = :each, \*args) {|\*args|  $\otimes$  block}  $\rightarrow$  lazy\_enum

Similar to Kernel#to\_enum, except it returns a lazy enumerator. This makes it easy to define <u>Enumerable</u> methods that will naturally remain lazy if called from a lazy enumerator.

For example, continuing from the example in Kernel#to\_enum:



🐵 zip(\*args)

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class Enumerator::Yielder

Lazy

In Files

enumerator.c

### Parent

Object

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## module Errno

Ruby exception objects are subclasses of Exception. However, operating systems typically report errors using plain integers. <u>Module Errno</u> is created dynamically to map these operating system errors to Ruby classes, with each error number generating its own subclass of SystemCallError. As the subclass is created in module Errno, its name will start Errno::.

The names of the Errno:: classes depend on the environment in which Ruby runs. On a typical Unix or Windows platform, there are Errno classes such as Errno::EACCES, Errno::EAGAIN, Errno::EINTR, and so on.

The integer operating system error number corresponding to a particular error is available as the class constant Errno::error::Errno.



The full list of operating system errors on your particular platform are available as the constants of Errno.

Errno.constants	BIG, :EACCES,	

### In Files

error.c

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class Exception

Descendants of class Exception are used to communicate between Kernel#raise and rescue statements in begin ... end blocks. Exception objects carry information about the exception – its type (the exception's class name), an optional descriptive string, and optional traceback information. Exception subclasses may add additional information like NameError#name.

Programs may make subclasses of Exception, typically of <u>StandardError</u> or <u>RuntimeError</u>, to provide custom classes and add additional information. See the subclass list below for defaults for raise and rescue.

When an exception has been raised but not yet handled (in rescue, ensure, at\_exit and END blocks) the global variable \$! will contain the current exception and \$@ contains the current exception's backtrace.

It is recommended that a library should have one subclass of <u>StandardError</u> or <u>RuntimeError</u> and have specific exception types inherit from it. This allows the user to rescue a generic exception type to catch all exceptions the library may raise even if future versions of the library add new exception subclasses.

For example:

```
class MyLibrary
class Error < RuntimeError
end
class WidgetError < Error
end
class FrobError < Error
end
end
```

To handle both WidgetError and FrobError the library user can rescue MyLibrary::Error.

The built-in subclasses of **Exception** are:

- NoMemoryError
- <u>ScriptError</u>
  - LoadError
  - NotImplementedError
  - SyntaxError
- SecurityError
- SignalException
  - Interrupt
- StandardError default for rescue
  - ArgumentError
    - UncaughtThrowError
  - EncodingError
  - FiberError
  - IOError

- EOFError
- IndexError
  - KeyError
  - StopIteration
- LocalJumpError
- NameError
  - NoMethodError
- RangeError
  - FloatDomainError
- RegexpError
- **<u>RuntimeError</u>** default for raise
- SystemCallError
  - Errno::\*
- ThreadError
- TypeError
- ZeroDivisionError
- SystemExit
- SystemStackError
- fatal impossible to rescue

### In Files

error.c

### Parent

**Object** 

### **Public Class Methods**

#### $\otimes$ exception(string) $\rightarrow$ an\_exception or exc

With no argument, or if the argument is the same as the receiver, return the receiver. Otherwise, create a new exception object of the same class as the receiver, but with a message equal to string.to\_str.

#### $\otimes$ new(msg = nil) $\rightarrow$ exception

Construct a new <u>Exception</u> object, optionally passing in a message.

### **Public Instance Methods**

#### ⊚ exc == obj → true or false

Equality—If *obj* is not an Exception, returns false. Otherwise, returns true if *exc* and *obj* share same class, messages, and backtrace.

#### Solution → backtrace → array

Returns any backtrace associated with the exception. The backtrace is an array of strings, each containing either "filename:lineNo: in `method''' or "filename:lineNo."

def a raise "boom" end
def b a() end
begin b() rescue => detail

```
print detail.backtrace.join("\n")
end
```

produces:

```
prog.rb:2:in `a'
prog.rb:6:in `b'
prog.rb:10
```

#### backtrace\_locations → array

Returns any backtrace associated with the exception. This method is similar to <u>#backtrace</u>, but the backtrace is an array of

```
Thread::Backtrace::Location.
```

Now, this method is not affected by <u>#set\_backtrace</u>.

#### cause → an\_exception or nil

Returns the previous exception (\$!) at the time this exception was raised. This is useful for wrapping exceptions and retaining the original exception information.

#### $\odot$ exception(string) $\rightarrow$ an\_exception or exc

With no argument, or if the argument is the same as the receiver, return the receiver. Otherwise, create a new exception object of the same class as the receiver, but with a message equal to string.to\_str.

#### inspect → string

Return this exception's class name and message

message → string

Returns the result of invoking exception.to\_s. Normally this returns the exception's message or name. By supplying a to\_str method, exceptions are agreeing to be used where Strings are expected.

#### set\_backtrace(backtrace) → array

Sets the backtrace information associated with exc. The backtrace must be an array of <u>String</u> objects or a single String in the format described in #backtrace.

#### $\odot$ to\_s $\rightarrow$ string

Returns exception's message (or the name of the exception if no message is set).

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class FalseClass

The global value false is the only instance of class Falseclass and represents a logically false value in boolean expressions. The class provides operators allowing false to participate correctly in logical expressions.

### **In Files**

object.c

### Parent

Object

### **Public Instance Methods**

#### $_{\odot}$ false & obj → false $_{\odot}$ nil & obj → false

And—Returns false. *obj* is always evaluated as it is the argument to a method call—there is no short-circuit evaluation in this case.

#### 

Exclusive Or—If *obj* is nil Or false, returns false; otherwise, returns true.

inspect()
Alias for: to\_s

to\_s → "false"

'nuf said...

Also aliased as: inspect

#### rightarrow false | obj → true or false rightarrow nil | obj → true or false

Or—Returns false if *obj* is nil Or false; true otherwise.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class Fiber

Fibers are primitives for implementing light weight cooperative concurrency in Ruby. Basically they are a means of creating code blocks that can be paused and resumed, much like threads. The main difference is that they are never preempted and that the scheduling must be done by the programmer and not the VM.

As opposed to other stackless light weight concurrency models, each fiber comes with a small 4KB stack. This enables the fiber to be paused from deeply nested function calls within the fiber block.

When a fiber is created it will not run automatically. Rather it must be be explicitly asked to run using the Fiber#resume method. The code running inside the fiber can give up control by calling Fiber.yield in which case it yields control back to caller (the caller of the Fiber#resume).

Upon yielding or termination the Fiber returns the value of the last executed expression

For instance:

```
fiber = Fiber.new do
Fiber.yield 1
2
```

#### end

```
puts fiber.resume
puts fiber.resume
puts fiber.resume
```

produces

1 2 FiberError: dead fiber called

The Fiber#resume method accepts an arbitrary number of parameters, if it is the first call to resume then they will be passed as block arguments. Otherwise they will be the return value of the call to Fiber.yield

Example:

```
fiber = Fiber.new do |first|
  second = Fiber.yield first + 2
end
puts fiber.resume 10
puts fiber.resume 14
puts fiber.resume 18
```

produces

12 14 FiberError: dead fiber called

### **In Files**

📄 cont.c

### Parent

Object

### **Public Class Methods**

#### ourrent() → fiber

Returns the current fiber. You need to require 'fiber' before using this method. If you are not running in the context of a fiber this method will return the root fiber.

### obj wield(args, ...) → obj

Yields control back to the context that resumed the fiber, passing along any arguments that were passed to it. The fiber will resume processing at this point when resume is called next. Any arguments passed to the next resume will be the value that this Fiber.yield expression evaluates to.

### **Public Instance Methods**

#### $_{\odot}$ alive? $\rightarrow$ true or false

Returns true if the fiber can still be resumed (or transferred to). After finishing execution of the fiber block this method will always return false. You need to require 'fiber' before using this method.

#### obj resume(args, ...) → obj

Resumes the fiber from the point at which the last Fiber.yield was called, or starts running it if it is the first call to resume. Arguments passed to resume will be the value of the Fiber.yield expression or will be passed as block parameters to the fiber's block if this is the first resume.

Alternatively, when resume is called it evaluates to the arguments passed to the next Fiber.yield statement inside the fiber's block or to the block value if it runs to completion without any Fiber.yield

### transfer(args, ...) → obj

Transfer control to another fiber, resuming it from where it last stopped or starting it if it was not resumed before. The calling fiber will be suspended much like in a call to Fiber.yield. You need to require 'fiber' before using this method.

The fiber which receives the transfer call is treats it much like a resume call. Arguments passed to transfer are treated like those passed to resume.

You cannot resume a fiber that transferred control to another one. This will cause a double resume error. You need to transfer control back to this fiber before it can yield and resume.

Example:

```
fiber1 = Fiber.new do
  puts "In Fiber 1"
  Fiber.yield
end
fiber2 = Fiber.new do
  puts "In Fiber 2"
```

```
fiber1.transfer
  puts "Never see this message"
end
fiber3 = Fiber.new do
  puts "In Fiber 3"
end
fiber2.resume
fiber3.resume
```

produces

In fiber 2 In fiber 1 In fiber 3

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class FiberError

Raised when an invalid operation is attempted on a <u>Fiber</u>, in particular when attempting to call/resume a dead fiber, attempting to yield from the root fiber, or calling a fiber across threads.



## In Files

📄 cont.c

### Parent

StandardError

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class File

A File is an abstraction of any file object accessible by the program and is closely associated with class IO File includes the methods of module FileTest as class methods, allowing you to write (for example) File.exist? ("foo").

In the description of File methods, *permission bits* are a platform-specific set of bits that indicate permissions of a file. On Unix-based systems, permissions are viewed as a set of three octets, for the owner, the group, and the rest of the world. For each of these entities, permissions may be set to read, write, or execute the file:

The permission bits 0644 (in octal) would thus be interpreted as read/write for owner, and readonly for group and other. Higher-order bits may also be used to indicate the type of file (plain, directory, pipe, socket, and so on) and various other special features. If the permissions are for a directory, the meaning of the execute bit changes; when set the directory can be searched.

On non-Posix operating systems, there may be only the ability to make a file read-only or readwrite. In this case, the remaining permission bits will be synthesized to resemble typical values. For instance, on Windows NT the default permission bits are 0644, which means read/write for owner, read-only for all others. The only change that can be made is to make the file read-only, which is reported as 0444.

Various constants for the methods in <u>File</u> can be found in <u>File::Constants</u>.

### **In Files**

📄 dir.c

📄 file.c

📄 io.c

### Parent

ΙΟ

### Constants

#### ALT\_SEPARATOR

platform specific alternative separator

#### PATH\_SEPARATOR

path list separator

#### SEPARATOR

separates directory parts in path

#### Separator

separates directory parts in path

### **Public Class Methods**

# absolute\_path(file\_name [, dir\_string] ) $\rightarrow$ $_{\textcircled{}}$ abs\_file\_name

Converts a pathname to an absolute pathname. Relative paths are referenced from the current working directory of the process unless *dir\_string* is given, in which case it will be used as the starting point. If the given pathname starts with a "~" it is NOT expanded, it is treated as a normal directory name.

```
File.absolute_path("~oracle/bin") #=> "<rel
</pre>
```

### atime(file\_name) → time

Returns the last access time for the named file as a Time object).

file\_name can be an IO object.



# basename(file\_name [, suffix]) $\rightarrow$ $_{\odot}$ base\_name

Returns the last component of the filename given in *file\_name*, which can be formed using both File::SEPARATOR and File::ALT\_SEPARATOR as the separator when File::ALT\_SEPARATOR is not nil. If *suffix* is given and present at the end of *file\_name*, it is removed. If *suffix* is ".\*", any extension will be

removed.



### $_{\odot}$ birthtime(file\_name) $\rightarrow$ time

Returns the birth time for the named file.

file\_name can be an IO object.

Note that on Windows (NTFS), returns creation time (birth time).



### $_{\odot}$ blockdev?(file\_name) → true or false

Returns true if the named file is a block device.

file\_name can be an IO object.

### $_{\odot}$ chardev?(file\_name) $\rightarrow$ true or false

Returns true if the named file is a character device.

file\_name can be an IO object.

### $_{\odot}$ chmod(mode\_int, file\_name, ... ) $\rightarrow$ integer

Changes permission bits on the named file(s) to the bit pattern represented by *mode\_int*. Actual effects are operating system dependent (see the beginning of this section). On Unix systems, see chmod(2) for details. Returns the number of files processed.

File.chmod(0644, "testfile", "out") #=> 2

### chown(owner\_int, group\_int, file\_name,... ) $_{\textcircled{}}$ $\rightarrow$ integer

Changes the owner and group of the named file(s) to the given numeric owner and group id's. Only a process with superuser privileges may change the owner of a file. The current owner of a file may change the file's group to any group to which the owner belongs. A nil or -1 owner or group id is ignored. Returns the number of files processed.

File.chown(nil, 100, "testfile")

#### $\odot$ ctime(file\_name) $\rightarrow$ time

Returns the change time for the named file (the time at which directory information about the file was changed, not the file itself).

file\_name can be an IO object.

Note that on Windows (NTFS), returns creation time (birth time).



#### 

Deletes the named files, returning the number of names passed as arguments. Raises an exception on any error. See also Dir::rmdir.

### $\otimes$ directory?(file\_name) $\rightarrow$ true or false

Returns true if the named file is a directory, or a symlink that points at a directory, and false otherwise.

file\_name can be an IO object.

File.directory?(".")

### o dirname(file\_name) → dir\_name

Returns all components of the filename given in *file\_name* except the last one. The filename can be formed using both File::SEPARATOR and File::ALT\_SEPARATOR as the separator when File::ALT\_SEPARATOR is not nil.



#### executable?(file\_name) → true or false Returns true if the named file is executable by the effective user id of this process.

 executable\_real?(file\_name) → true or false Returns true if the named file is executable by the real user id of this process.

#### $\otimes$ exist?(file\_name) $\rightarrow$ true or false

Return true if the named file exists.

file\_name can be an IO object.

"file exists" means that stat() or fstat() system call is successful.

#### $\otimes$ exists?(file\_name) $\rightarrow$ true or false

Deprecated method. Don't use.

# expand\_path(file\_name [, dir\_string] ) $\rightarrow$ $_{\textcircled{}}$ abs\_file\_name

Converts a pathname to an absolute pathname. Relative paths are referenced from the current working directory of the process unless dir\_string is given, in which case it will be used as the starting point. The given pathname may start with a "~", which expands to the process owner's home directory (the environment variable HOME must be set correctly). "~user" expands to the named user's home directory.



A simple example of using dir\_string is as follows.

A more complex example which also resolves parent directory is as follows. Suppose we are in bin/mygem and want the absolute path of lib/mygem.rb.



So first it resolves the parent of \_\_\_\_\_, that is bin/, then go to the parent, the root of the project and appends lib/mygem.rb.

### $\otimes$ extname(path) $\rightarrow$ string

Returns the extension (the portion of file name in path starting from the last period).

If path is a dotfile, or starts with a period, then the starting dot is not dealt with the start of the extension.

An empty string will also be returned when the period

is the last character in path.

<pre>File.extname("test.rb")</pre>	
<pre>File.extname("a/b/d/test.rb")</pre>	
<pre>File.extname("foo.")</pre>	
<pre>File.extname("test")</pre>	
<pre>File.extname(".profile")</pre>	
<pre>File.extname(".profile.sh")</pre>	

### file?(file) → true or false

Returns true if the named file exists and is a regular file.

file can be an  $\underline{IO}$  object.

If the file argument is a symbolic link, it will resolve the symbolic link and use the file referenced by the link.

# fnmatch( pattern, path, [flags] ) $\rightarrow$ (true or $_{\textcircled{}}$ false)

# fnmatch?( pattern, path, [flags] ) $\rightarrow$ (true or $\otimes$ false)

Returns true if path matches against pattern. The pattern is not a regular expression; instead it follows rules similar to shell filename globbing. It may contain the following metacharacters:

Matches any file. Can be restricted by other values in the glob. Equivalent to / .\* /x in regexp.

\*

Matches all files regular files

с\*

Matches all files beginning with  $\ensuremath{\mathbf{c}}$ 

\*c

Matches all files ending with c

\*c\*

Matches all files that have c in them (including at the beginning or end).

To match hidden files (that start with a . set the File::FNM\_DOTMATCH flag.

\* \*

Matches directories recursively or files expansively.

?

Matches any one character. Equivalent to  $/.{1}/$  in regexp.

[set]

Matches any one character in set. Behaves exactly like character sets in <u>Regexp</u>, including set negation ( $[^a-z]$ ).

\

Escapes the next metacharacter.

a,b

Matches pattern a and pattern b if File::FNM\_EXTGLOB flag is enabled. Behaves like a Regexp union ((?:a|b)).

flags is a bitwise OR of the FNM\_XXX constants. The same glob pattern and flags are used by Dir.glob.

Examples:

File.fnmatch('cat', File.fnmatch('cat',	'cat') 'category')		
<pre>File.fnmatch('c{at,ub}s' File.fnmatch('c{at,ub}s'</pre>	, 'cats') , 'cats', File::	FNM_	_EXTGI
<pre>File.fnmatch('c?t', File.fnmatch('c?t', File.fnmatch('c?',</pre>	'cat') 'cat') 'cats')		

```
File.fnmatch('c*t',
                        'c/a/b/t')
File.fnmatch('ca[a-z]', 'cat')
File.fnmatch('ca[^t]', 'cat')
File.fnmatch('cat', 'CAT')
File.fnmatch('cat', 'CAT', File::FNM_CASEFOLD)
File.fnmatch('?', '/', File::FNM_PATHNAME)
File.fnmatch('[/]', '/', File::FNM_PATHNAME)
File.fnmatch('\?',
                     <u>'?'</u>)
File.fnmatch('\a'
                      'a')
File.fnmatch('\a', '\a', File::FNM_NOESCAPE)
                      '?')
File.fnmatch('[\?]',
                     '.profile')
File.fnmatch('*',
File.fnmatch('*',
                     '.profile', File::FNM_DOTMATC
File.fnmatch('.*',
                    '.profile')
File.fnmatch(rbfiles, 'main.rb')
File.fnmatch(rbfiles, './main.rb')
                      'lib/song.rb')
File.fnmatch(rbfiles,
File.fnmatch('**.rb',
                      'main.rb')
File.fnmatch('**.rb',
                      './main.rb')
File.fnmatch('**.rb', 'lib/song.rb')
File.fnmatch('*',
                             'dave/.profile')
pattern = <u>'*' '/' '*'</u>
File.fnmatch(pattern, 'dave/.profile', File::FNM_
File.fnmatch(pattern, 'dave/.profile', File::FNM_
File.fnmatch(pattern, 'a/b/c/foo', File::FNM_PATH
File.fnmatch(pattern, '/a/b/c/foo', File::FNM_PAT
File.fnmatch(pattern, 'c:/a/b/c/foo', File::FNM_F
File.fnmatch(pattern, 'a/.b/c/foo', File::FNM_PA
                      'a/.b/c/foo', File::FNM_PA
File.fnmatch(pattern,
4
```

fnmatch( pattern, path, [flags] )  $\rightarrow$  (true or  $\otimes$  false)

fnmatch?( pattern, path, [flags] )  $\rightarrow$  (true or

### 🚳 false)

Returns true if path matches against pattern. The pattern is not a regular expression; instead it follows rules similar to shell filename globbing. It may contain the following metacharacters:

Matches any file. Can be restricted by other values in the glob. Equivalent to /  $\ .\ *\ /x$  in regexp.

k

Matches all files regular files

с\*

Matches all files beginning with  $\ensuremath{\mathsf{c}}$ 

\*с

Matches all files ending with  $\ensuremath{\mathsf{c}}$ 

\*c\*

Matches all files that have c in them (including at the beginning or end).

To match hidden files (that start with a . set the File::FNM\_DOTMATCH flag.

\* \*

Matches directories recursively or files expansively.

?

Matches any one character. Equivalent to  $/.{1}/in$  regexp.

[set]

Matches any one character in set. Behaves exactly like character sets in <u>Regexp</u>, including set negation ( $[^a-z]$ ).

Escapes the next metacharacter.

{a,b}

\

Matches pattern a and pattern b if File::FNM\_EXTGLOB flag is enabled. Behaves like a <u>Regexp</u> union ((?:a|b)).

flags is a bitwise OR of the FNM\_XXX constants. The same glob pattern and flags are used by <u>Dir.glob</u>.

Examples:

File.fnmatch('cat', 'cat') File.fnmatch('cat', 'category') File.fnmatch('c{at,ub}s', 'cats') File.fnmatch('c{at,ub}s', 'cats', File::FNM\_EXTGI File.fnmatch('c?t', 'cat') 'cat') File.fnmatch('c??t', File.fnmatch('c\*', 'cats') File.fnmatch('c\*t', 'c/a/b/t') File.fnmatch('ca[a-z]', 'cat') File.fnmatch('ca[^t]', 'cat') File.fnmatch('cat', 'CAT') File.fnmatch('cat', 'CAT', File::FNM\_CASEFOLD) File.fnmatch('?', '/', File::FNM\_PATHNAME) File.fnmatch('\*', '/', File::FNM\_PATHNAME) File.fnmatch('[/]', '/', File::FNM\_PATHNAME) File.fnmatch('\?', '?') File.fnmatch('\a', 'a') File.fnmatch(' a', '\a', File::FNM\_NOESCAPE) File.fnmatch('[\?]', '?') File.fnmatch('\*', '.profile') File.fnmatch('\*', '.profile', File::FNM\_DOTMAT( File.fnmatch('.\*', '.profile') rbfiles = '\*\*' '/' '\*.rb' # File.fnmatch(rbfiles, 'main.rb') File.fnmatch(rbfiles, './main.rb') 'lib/song.rb') File.fnmatch(rbfiles, File.fnmatch('\*\*.rb', 'main.rb') File.fnmatch('\*\*.rb', './main.rb') File.fnmatch('\*\*.rb', 'lib/song.rb') 'dave/.profile') File.fnmatch('\*',



### $_{\odot}$ ftype(file\_name) $\rightarrow$ string

Identifies the type of the named file; the return string is one of "file", "directory", "characterSpecial", "blockSpecial", "fifo", "link", "socket", Or "unknown".



#### $\odot$ grpowned?(file\_name) $\rightarrow$ true or false

Returns true if the named file exists and the effective group id of the calling process is the owner of the file. Returns false on Windows.

file\_name can be an IO object.

### $_{\odot}$ identical?(file\_1, file\_2) → true or false

Returns true if the named files are identical.

*file\_1* and *file\_2* can be an IO object.

```
open("a", "w") {}
p File.identical?("a", "a")  #=> true
p File.identical?("a", "./a")  #=> true
File.link("a", "b")
```

<pre>p File.identical?("a",</pre>	"b")	#=> true
<pre>File.symlink("a", "c")</pre>		
<pre>p File.identical?("a",</pre>	"c")	
open("d", "w") {}		
<pre>p File.identical?("a",</pre>	"d")	

### join(string, ...) → string

Returns a new string formed by joining the strings using File::SEPARATOR.



### $_{\odot}$ lchmod(mode\_int, file\_name, ...) $\rightarrow$ integer

Equivalent to File::chmod, but does not follow symbolic links (so it will change the permissions associated with the link, not the file referenced by the link). Often not available.

### Ichown(owner\_int, group\_int, file\_name,..) → ⊚ integer

Equivalent to File::chown, but does not follow symbolic links (so it will change the owner associated with the link, not the file referenced by the link). Often not available. Returns number of files in the argument list.

### $\otimes$ link(old\_name, new\_name) $\rightarrow$ 0

Creates a new name for an existing file using a hard link. Will not overwrite *new\_name* if it already exists (raising a subclass of systemCallError). Not available on all platforms.




#### 

Same as File::stat, but does not follow the last symbolic link. Instead, reports on the link itself.



#### $_{\odot}$ mtime(file\_name) $\rightarrow$ time

Returns the modification time for the named file as a Time object.

file\_name can be an IO object.



```
    mew(filename, mode="r" [, opt]) → file
    mew(filename [, mode [, perm]] [, opt]) → file
    Opens the file named by filename according to the
```

given mode and returns a new File object.

See IO.new for a description of mode and opt.

If a file is being created, permission bits may be given in perm. These mode and permission bits are platform dependent; on Unix systems, see open(2) and chmod(2) man pages for details.

#### Examples



## $_{\odot}$ open(filename, mode="r" [, opt]) → file

open(filename [, mode [, perm]] [, opt]) → file
 open(filename, mode="r" [, opt]) {|file| block
 ) → obj

open(filename [, mode [, perm]] [, opt]) {|file|  $\otimes$  block }  $\rightarrow$  obj

With no associated block, File.open is a synonym for ::new. If the optional code block is given, it will be passed the opened file as an argument and the File object will automatically be closed when the block terminates. The value of the block will be returned from File.open.

If a file is being created, its initial permissions may be set using the perm parameter. See <u>::new</u> for further discussion.

See <u>IO.new</u> for a description of the mode and opt parameters.

#### $\odot$ owned?(file\_name) $\rightarrow$ true or false

Returns true if the named file exists and the effective used id of the calling process is the owner of the file.

file\_name can be an IO object.

## 

Returns the string representation of the path



pipe?(file\_name) → true or false

Returns true if the named file is a pipe.

file\_name can be an IO object.

## $\otimes$ readable?(file\_name) $\rightarrow$ true or false

Returns true if the named file is readable by the effective user id of this process.

#### $_{\odot}$ readable\_real?(file\_name) $\rightarrow$ true or false

Returns true if the named file is readable by the real user id of this process.

#### $\otimes$ readlink(link\_name) $\rightarrow$ file\_name

Returns the name of the file referenced by the given link. Not available on all platforms.

File.symlink("testfile", "link2test") #=> 0
File.readlink("link2test") #=> "tes

## realdirpath(pathname [, dir\_string]) $\rightarrow$ $_{\odot}$ real\_pathname

Returns the real (absolute) pathname of *pathname* in the actual filesystem. The real pathname doesn't contain symlinks or useless dots.

If *dir\_string* is given, it is used as a base directory for interpreting relative pathname instead of the current directory.

The last component of the real pathname can be nonexistent.

## realpath(pathname [, dir\_string]) →

#### real\_pathname

Returns the real (absolute) pathname of *pathname* in the actual filesystem not containing symlinks or useless dots.

If *dir\_string* is given, it is used as a base directory for interpreting relative pathname instead of the current directory.

All components of the pathname must exist when this method is called.

#### $_{\odot}$ rename(old\_name, new\_name) $\rightarrow$ 0

Renames the given file to the new name. Raises a SystemCallError if the file cannot be renamed.

File.rename("afile", "afile.bak") #=> 0

#### $\otimes$ setgid?(file\_name) $\rightarrow$ true or false

Returns true if the named file has the setgid bit set.

#### $_{\odot}$ setuid?(file\_name) → true or false

Returns true if the named file has the setuid bit set.

#### $_{\odot}$ size(file\_name) $\rightarrow$ integer

Returns the size of file\_name.

*file\_name* can be an **IO** object.

#### $\otimes$ size?(file\_name) $\rightarrow$ Integer or nil

Returns nil if file\_name doesn't exist or has zero size, the size of the file otherwise.

*file\_name* can be an **IO** object.

### $\odot$ socket?(file\_name) $\rightarrow$ true or false

Returns true if the named file is a socket.

file\_name can be an IO object.

#### $_{\odot}$ split(file\_name) $\rightarrow$ array

Splits the given string into a directory and a file component and returns them in a two-element array. See also File::dirname and File::basename.



## $_{\odot}$ stat(file\_name) $\rightarrow$ stat

Returns a File::Stat object for the named file (see File::Stat).



## sticky?(file\_name) $\rightarrow$ true or false

Returns true if the named file has the sticky bit set.

## $_{\odot}$ symlink(old\_name, new\_name) $\rightarrow$ 0

Creates a symbolic link called *new\_name* for the existing file *old\_name*. Raises a NotImplemented exception on platforms that do not support symbolic links.

File.symlink("testfile", "link2test") #=

 $\odot$  symlink?(file\_name)  $\rightarrow$  true or false

Returns true if the named file is a symbolic link.

#### $_{\odot}$ truncate(file\_name, integer) $\rightarrow$ 0

Truncates the file *file\_name* to be at most *integer* bytes long. Not available on all platforms.



### $_{\odot}$ umask() → integer $_{\odot}$ umask(integer) → integer

Returns the current umask value for this process. If the optional argument is given, set the umask to that value and return the previous value. Umask values are *subtracted* from the default permissions, so a umask of 0222 would make a file read-only for everyone.

## $_{\textcircled{}}$ delete(file\_name, ...) $\rightarrow$ integer

#### $_{\odot}$ unlink(file\_name, ...) $\rightarrow$ integer

Deletes the named files, returning the number of names passed as arguments. Raises an exception on any error. See also Dir::rmdir.

#### $_{\odot}$ utime(atime, mtime, file\_name,...) $\rightarrow$ integer

Sets the access and modification times of each named file to the first two arguments. Returns the number of file names in the argument list.

#### $\otimes$ world\_readable?(file\_name) $\rightarrow$ fixnum or nil

If *file\_name* is readable by others, returns an integer representing the file permission bits of *file\_name*. Returns nil otherwise. The meaning of the bits is platform dependent; on Unix systems, see stat(2).

*file\_name* can be an IO object.



#### $\otimes$ world\_writable?(file\_name) $\rightarrow$ fixnum or nil

If *file\_name* is writable by others, returns an integer representing the file permission bits of *file\_name*. Returns nil otherwise. The meaning of the bits is platform dependent; on Unix systems, see stat(2).

*file\_name* can be an IO object.



# Seturns true if the named file is writable by the effective user id of this process.

writable\_real?(file\_name) → true or false Returns true if the named file is writable by the real user id of this process.

#### sero?(file\_name) → true or false

Returns true if the named file exists and has a zero size.

file\_name can be an IO object.

## **Public Instance Methods**

#### atime → time

Returns the last access time (a Time object)



#### birthtime → time

Returns the birth time for file.

Note that on Windows (NTFS), returns creation time (birth time).



#### $_{\odot}$ chmod(mode\_int) $\rightarrow$ 0

Changes permission bits on *file* to the bit pattern represented by *mode\_int*. Actual effects are platform dependent; on Unix systems, see chmod(2) for details. Follows symbolic links. Also see File#lchmod.

```
f = File.new("out", "w");
f.chmod(0644)  #=> 0
```

#### $_{\odot}$ chown(owner\_int, group\_int ) $\rightarrow$ 0

Changes the owner and group of *file* to the given numeric owner and group id's. Only a process with superuser privileges may change the owner of a file. The current owner of a file may change the file's group to any group to which the owner belongs. A nil or -1 owner or group id is ignored. Follows symbolic links. See also File#lchown.

```
File.new("testfile").chown(502, 1000)
```

#### original of the second secon

Returns the change time for *file* (that is, the time directory information about the file was changed, not the file itself).

Note that on Windows (NTFS), returns creation time (birth time).



#### $\odot$ flock(locking\_constant) $\rightarrow$ 0 or false

Locks or unlocks a file according to *locking\_constant* (a logical *or* of the values in the table below). Returns false if File::LOCK\_NB is specified and the operation would otherwise have blocked. Not available on all platforms.

Locking constants (in class File):

LOCK_EX	Exclusive lock. Only one process may   exclusive lock for a given file at a
	-+
LOCK_NB	Don't block when locking. May be comm   with other lock options using logical
	-+
LOCK_SH	Shared lock. Multiple processes may e

		shared	lock	for	a	given	file	at	the	S
LOCK_UN		Unlock								
•										Þ

Example:



#### Istat → stat

Same as 10#stat, but does not follow the last symbolic link. Instead, reports on the link itself.



#### mtime → time

Returns the modification time for *file*.



## righting path → filename $righting to_path → filename$

Returns the pathname used to create *file* as a string. Does not normalize the name.



#### $\odot$ size $\rightarrow$ integer

Returns the size of *file* in bytes.

```
File.new("testfile").size #=> 66
```

### $_{\odot}$ path $\rightarrow$ filename

#### o to\_path → filename

Returns the pathname used to create *file* as a string. Does not normalize the name.



## $_{\odot}$ truncate(integer) $\rightarrow$ 0

Truncates *file* to at most *integer* bytes. The file must be opened for writing. Not available on all platforms.



Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## module File::Constants

File::Constants provides file-related constants. All possible file constants are listed in the documentation but they may not all be present on your platform.

If the underlying platform doesn't define a constant the corresponding Ruby constant is not defined.

Your platform documentations (e.g. man open(2)) may describe more detailed information.

## **In Files**

📄 dir.c

## Constants

#### APPEND

append on each write

#### BINARY

disable line code conversion

#### CREAT

create file if it does not exist

#### DIRECT

Try to minimize cache effects of the I/O to and from this file.

#### DSYNC

any write operation perform synchronously except some meta data

#### EXCL

error if **CREAT** and the file exists

#### LOCK\_EX

exclusive lock. see File#flock

#### LOCK\_NB

non-blocking lock. used with <u>LOCK\_SH</u> or <u>LOCK\_EX</u>. see <u>File#flock</u>

#### LOCK\_SH

shared lock. see File#flock

#### LOCK\_UN

unlock. see File#flock

#### NOATIME

do not change atime

#### NOCTTY

not to make opened IO the controlling terminal device

#### NOFOLLOW

do not follow symlinks

#### NONBLOCK

do not block on open or for data to become available

#### NULL

Name of the null device

#### RDONLY

open for reading only

#### RDWR

open for reading and writing

#### RSYNC

any read operation perform synchronously. used with SYNC or DSYNC.

#### SYNC

any write operation perform synchronously

#### TRUNC

truncate size to 0

#### WRONLY

open for writing only

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class File::Stat

Objects of class File::stat encapsulate common status information for File objects. The information is recorded at the moment the File::stat object is created; changes made to the file after that point will not be reflected. File::stat objects are returned by IO#stat, File::stat, File#lstat, and File::lstat. Many of these methods return platform-specific values, and not all values are meaningful on all systems. See also Kernel#test.

## **In Files**

📄 dir.c

## Parent

Object

## **Included Modules**

Comparable

## **Public Class Methods**

#### Sile::Stat.new(file\_name) → stat

Create a File::Stat object for the given file name (raising an exception if the file doesn't exist).

## **Public Instance Methods**

#### $_{\odot}$ stat <=> other\_stat $\rightarrow$ -1, 0, 1, nil

Compares File::Stat objects by comparing their respective modification times.

nil is returned if other\_stat is not a File::Stat object



#### $_{\odot}$ atime $\rightarrow$ time

Returns the last access time for this file as an object of class Time.



#### birthtime → aTime

Returns the birth time for *stat*. If the platform doesn't have birthtime, returns *ctime*.

```
File.write("testfile", "foo")
sleep 10
File.write("testfile", "bar")
sleep 10
File.chmod(0644, "testfile")
sleep 10
File.read("testfile")
File.stat("testfile").birthtime  #=> 2014-02-24
File.stat("testfile").ctime  #=> 2014-02-24
File.stat("testfile").atime  #=> 2014-02-24
```

#### blksize → integer or nil

Returns the native file system's block size. Will return nil on platforms that don't support this information.

```
File.stat("testfile").blksize #=> 4096
```

#### blockdev? → true or false

Returns true if the file is a block device, false if it isn't or if the operating system doesn't support this feature.

```
File.stat("testfile").blockdev? #=> false
File.stat("/dev/hda1").blockdev? #=> true
```

#### blocks → integer or nil

Returns the number of native file system blocks allocated for this file, or nil if the operating system doesn't support this feature.

```
File.stat("testfile").blocks #=
```

#### $_{\odot}$ chardev? $\rightarrow$ true or false

Returns true if the file is a character device, false if it isn't or if the operating system doesn't support this feature.

#### or time → aTime or time o

Returns the change time for *stat* (that is, the time directory information about the file was changed, not the file itself).

Note that on Windows (NTFS), returns creation time (birth time).



#### objective dev → fixnum

Returns an integer representing the device on which *stat* resides.

```
File.stat("testfile").dev #=> 774
```

#### objective dev\_major → fixnum

Returns the major part of File\_Stat#dev Or nil.

#### objective dev\_minor → fixnum

Returns the minor part of File\_Stat#dev Or nil.

```
File.stat("/dev/fd1").dev_minor #=> 1
File.stat("/dev/tty").dev_minor #=> 0
```

## $\otimes$ directory?(file\_name) $\rightarrow$ true or false

Returns true if the named file is a directory, or a symlink that points at a directory, and false otherwise.

file\_name can be an IO object.

File.directory?(".")

rightarrow executable?  $\rightarrow$  true or false

Returns true if *stat* is executable or if the operating system doesn't distinguish executable files from nonexecutable files. The tests are made using the effective owner of the process.

```
File.stat("testfile").executable? #=> false
```

#### $\otimes$ executable\_real? $\rightarrow$ true or false

Same as executable?, but tests using the real owner of the process.

#### ile? → true or false

Returns true if *stat* is a regular file (not a device file, pipe, socket, etc.).

```
File.stat("testfile").file? #=> true
```

#### type → string

Identifies the type of *stat*. The return string is one of: "file", "directory", "characterSpecial",

"blockSpecial", "fifo", "link", "socket", Or "unknown".

```
File.stat("/dev/tty").ftype #=> "characterSpec:
4
```

#### igid → fixnum

Returns the numeric group id of the owner of stat.

```
File.stat("testfile").gid #=> 500
```

#### $\odot$ grpowned? $\rightarrow$ true or false

Returns true if the effective group id of the process is the same as the group id of *stat*. On Windows NT,

returns false.

File.stat("testfile").grpowned? #=> true
File.stat("/etc/passwd").grpowned? #=> false

#### ino → fixnum

Returns the inode number for stat.

```
File.stat("testfile").ino #=> 1083669
```

#### $\bigcirc$ inspect $\rightarrow$ string

Produce a nicely formatted description of stat.

File.st	at("/etc/passwd").inspect
#=>	
#	
#	
#	
#	
#	
<b>↓</b>	Þ

#### mode → fixnum

Returns an integer representing the permission bits of *stat*. The meaning of the bits is platform dependent; on Unix systems, see stat(2).



#### mtime → aTime

Returns the modification time of stat.



#### on link → fixnum

Returns the number of hard links to stat.



#### $\odot$ owned? $\rightarrow$ true or false

Returns true if the effective user id of the process is the same as the owner of *stat*.

```
File.stat("testfile").owned?#=> trueFile.stat("/etc/passwd").owned?#=> false
```

#### pipe? → true or false

Returns true if the operating system supports pipes and *stat* is a pipe; false otherwise.

#### ordev → fixnum or nil

Returns an integer representing the device type on which *stat* resides. Returns nil if the operating system doesn't support this feature.



#### ordev\_major → fixnum

Returns the major part of File\_Stat#rdev Or nil.

```
File.stat("/dev/fd1").rdev_major #=> 2
File.stat("/dev/tty").rdev_major #=> 5
```

ordev\_minor → fixnum

Returns the minor part of File\_Stat#rdev Or nil.



#### $_{\odot}$ readable? $\rightarrow$ true or false

Returns true if *stat* is readable by the effective user id of this process.

```
File.stat("testfile").readable? #=> true
```

#### $\odot$ readable\_real? $\rightarrow$ true or false

Returns true if *stat* is readable by the real user id of this process.

File.stat("testfile").readable\_real? #=> true

#### $_{\odot}$ setgid? $\rightarrow$ true or false

Returns true if *stat* has the set-group-id permission bit set, false if it doesn't or if the operating system doesn't support this feature.

```
File.stat("/usr/sbin/lpc").setgid? #=> true
```

#### $_{\odot}$ setuid? $\rightarrow$ true or false

Returns true if *stat* has the set-user-id permission bit set, false if it doesn't or if the operating system doesn't support this feature.

```
File.stat("/bin/su").setuid? #=> true
```

#### Size → fixnum

Returns the size of *stat* in bytes.

#### File.stat("testfile").size #=> 66

#### integer → integer

Returns the size of *stat* in bytes.

```
File.stat("testfile").size #=> 66
```

#### $_{\odot}$ socket? $\rightarrow$ true or false

Returns true if *stat* is a socket, false if it isn't or if the operating system doesn't support this feature.

```
File.stat("testfile").socket? #=> false
```

#### $_{\odot}$ sticky? $\rightarrow$ true or false

Returns true if *stat* has its sticky bit set, false if it doesn't or if the operating system doesn't support this feature.

```
File.stat("testfile").sticky? #=> false
```

#### $_{\odot}$ symlink? $\rightarrow$ true or false

Returns true if *stat* is a symbolic link, false if it isn't or if the operating system doesn't support this feature. As File::stat automatically follows symbolic links, symlink? will always be false for an object returned by File::stat.

```
File.symlink("testfile", "alink")#=> 0File.stat("alink").symlink?#=> falseFile.lstat("alink").symlink?#=> true
```

#### o uid → fixnum

Returns the numeric user id of the owner of stat.



### $_{\odot}$ world\_readable? $\rightarrow$ fixnum or nil

If *stat* is readable by others, returns an integer representing the file permission bits of *stat*. Returns nil otherwise. The meaning of the bits is platform dependent; on Unix systems, see stat(2).



## world\_writable? → fixnum or nil

If *stat* is writable by others, returns an integer representing the file permission bits of *stat*. Returns nil otherwise. The meaning of the bits is platform dependent; on Unix systems, see stat(2).



#### $_{\odot}$ writable? $\rightarrow$ true or false

Returns true if *stat* is writable by the effective user id of this process.

```
File.stat("testfile").writable? #=> true
```

#### $\otimes$ writable\_real? $\rightarrow$ true or false

Returns true if *stat* is writable by the real user id of this process.

File.stat("testfile").writable\_real? #=> true

#### $\odot$ zero? $\rightarrow$ true or false

Returns true if *stat* is a zero-length file; false otherwise.

File.stat("testfile").zero? #=> false

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## module FileTest

FileTest implements file test operations similar to those used in File::Stat. It exists as a standalone module, and its methods are also insinuated into the File class. (Note that this is not done by inclusion: the interpreter cheats).

## **In Files**

📄 file.c

## **Public Instance Methods**

 blockdev?(file\_name) → true or false Returns true if the named file is a block device.

file\_name can be an IO object.

#### $\otimes$ chardev?(file\_name) $\rightarrow$ true or false

Returns true if the named file is a character device.

file\_name can be an IO object.

#### $\otimes$ directory?(file\_name) $\rightarrow$ true or false

Returns true if the named file is a directory, or a symlink that points at a directory, and false otherwise.

file\_name can be an IO object.

#### File.directory?(".")

 executable?(file\_name) → true or false Returns true if the named file is executable by the effective user id of this process.

#### $\odot$ executable\_real?(file\_name) $\rightarrow$ true or false

Returns true if the named file is executable by the real user id of this process.

#### $_{\odot}$ exist?(file\_name) $\rightarrow$ true or false

Return true if the named file exists.

*file\_name* can be an IO object.

"file exists" means that stat() or fstat() system call is successful.

#### $\otimes$ exists?(file\_name) $\rightarrow$ true or false

Deprecated method. Don't use.

#### $_{\odot}$ file?(file) → true or false

Returns true if the named file exists and is a regular file.

file can be an IO object.

If the file argument is a symbolic link, it will resolve the symbolic link and use the file referenced by the link.

### $\odot$ grpowned?(file\_name) $\rightarrow$ true or false

Returns true if the named file exists and the effective

group id of the calling process is the owner of the file. Returns false on Windows.

file\_name can be an IO object.

#### ⊗ identical?(file\_1, file\_2) → true or false

Returns true if the named files are identical.

*file\_1* and *file\_2* can be an IO object.

open("a", "w") {}		
<pre>p File.identical?("a",</pre>	"a")	
<pre>p File.identical?("a",</pre>	"./a")	
<pre>File.link("a", "b")</pre>		
<pre>p File.identical?("a",</pre>	"b")	
<pre>File.symlink("a", "c")</pre>		
<pre>p File.identical?("a",</pre>	"c")	
open("d", "w") {}		
<pre>p File.identical?("a",</pre>	"d")	

#### $\odot$ owned?(file\_name) $\rightarrow$ true or false

Returns true if the named file exists and the effective used id of the calling process is the owner of the file.

file\_name can be an IO object.

#### $_{\odot}$ pipe?(file\_name) → true or false

Returns true if the named file is a pipe.

file\_name can be an IO object.

#### $\otimes$ readable?(file\_name) $\rightarrow$ true or false

Returns true if the named file is readable by the effective user id of this process.

#### $\odot$ readable\_real?(file\_name) $\rightarrow$ true or false

Returns true if the named file is readable by the real user id of this process.

#### setgid?(file\_name) → true or false Returns true if the named file has the setgid bit set.

Setuid?(file\_name) → true or false Returns true if the named file has the setuid bit set.

size(file\_name) → integer
 Returns the size of file\_name.

file\_name can be an IO object.

#### size?(file\_name) $\rightarrow$ Integer or nil

Returns nil if file\_name doesn't exist or has zero size, the size of the file otherwise.

file\_name can be an IO object.

#### $_{\odot}$ socket?(file\_name) → true or false

Returns true if the named file is a socket.

file\_name can be an IO object.

#### $_{\odot}$ sticky?(file\_name) $\rightarrow$ true or false

Returns true if the named file has the sticky bit set.

#### $\odot$ symlink?(file\_name) $\rightarrow$ true or false

Returns true if the named file is a symbolic link.

world\_readable?(file\_name) → fixnum or nil If file\_name is readable by others, returns an integer representing the file permission bits of file\_name. Returns nil otherwise. The meaning of the bits is platform dependent; on Unix systems, see stat(2).

file\_name can be an IO object.



### $_{\odot}$ world\_writable?(file\_name) $\rightarrow$ fixnum or nil

If *file\_name* is writable by others, returns an integer representing the file permission bits of *file\_name*. Returns nil otherwise. The meaning of the bits is platform dependent; on Unix systems, see stat(2).

*file\_name* can be an **IO** object.



Seturns true if the named file is writable by the effective user id of this process.

writable\_real?(file\_name) → true or false Returns true if the named file is writable by the real user id of this process.

#### sero?(file\_name) → true or false

Returns true if the named file exists and has a zero size.

*file\_name* can be an <u>IO</u> object.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class Fixnum

Holds Integer values that can be represented in a native machine word (minus 1 bit). If any operation on a Fixnum exceeds this range, the value is automatically converted to a Bignum.

Fixnum objects have immediate value. This means that when they are assigned or passed as parameters, the actual object is passed, rather than a reference to that object.

Assignment does not alias Fixnum objects. There is effectively only one Fixnum object instance for any given integer value, so, for example, you cannot add a singleton method to a Fixnum. Any attempt to add a singleton method to a Fixnum object will raise a TypeError.

## **In Files**

📄 numeric.c

## Parent

Integer

## **Public Instance Methods**

## $_{\odot}$ fix % other → real $_{\odot}$ modulo(other) → real

Returns fix modulo other.

See Numeric#divmod for more information.

integer → integer\_result Bitwise AND.

## a fix \* numeric → numeric result

Performs multiplication: the class of the resulting object depends on the class of numeric and on the magnitude of the result. It may return a Bignum.

#### on fix \*\* numeric → numeric\_result

Raises fix to the power of numeric, which may be negative or fractional.



#### ⊚ fix + numeric → numeric\_result

Performs addition: the class of the resulting object depends on the class of numeric and on the magnitude of the result. It may return a Bignum.

#### fix - numeric → numeric\_result

Performs subtraction: the class of the resulting object depends on the class of numeric and on the magnitude of the result. It may return a Bignum.

#### $\odot$ -fix $\rightarrow$ integer

Negates fix, which may return a Bignum.

#### fix / numeric → numeric\_result

Performs division: the class of the resulting object depends on the class of numeric and on the magnitude of the result. It may return a Bignum.

#### $_{\odot}$ fix < real → true or false

Returns true if the value of fix is less than that of real.

#### on fix << count → integer</p>

Shifts fix left count positions, or right if count is negative.

#### is <= real → true or false</p>

Returns true if the value of fix is less than or equal to that of real.

#### In the second secon

Comparison—Returns -1, 0, +1 or nil depending on whether fix is less than, equal to, or greater than numeric.

This is the basis for the tests in the <u>Comparable</u> module.

nil is returned if the two values are incomparable.

#### $_{\odot}$ fix == other $\rightarrow$ true or false

Return true if fix equals other numerically.



## Is fix == other → true or false

Return true if fix equals other numerically.



## is fix > real → true or false

Returns true if the value of fix is greater than that of real.

## $_{\odot}$ fix >= real → true or false

Returns true if the value of fix is greater than or equal to that of real.

#### integer → integer

Shifts fix right count positions, or left if count is negative.

## fix[n] → 0, 1

Bit Reference—Returns the +n+th bit in the binary representation of fix, where fix[0] is the least significant bit.

For example:

```
a = 0b11001100101010
30.downto(0) do |n| print a[n] end
#=> 000000000000000011001100101010
```

## fix ^ integer → integer\_result

Bitwise EXCLUSIVE OR.

#### $_{\odot}$ abs $\rightarrow$ integer $_{\odot}$ magnitude $\rightarrow$ integer

Returns the absolute value of fix.

-12345.abs #=> 12345 12345.abs #=> 12345

## bit\_length → integer integer

Returns the number of bits of the value of *int*.

"the number of bits" means that the bit position of the highest bit which is different to the sign bit. (The bit position of the bit 2\*\*n is n+1.) If there is no such bit (zero or minus one), zero is returned.

I.e. This method returns ceil(log2(int < 0 ? -int : int+1)).

(-2**12-1).bit_length	
(-2**12).bit_length	
(-2**12+1).bit_length	
-0x101.bit_length	
-0x100.bit_length	
-0xff.bit_length	
-2.bit_length	
-1.bit_length	
0.bit_length	
1.bit_length	
0xff.bit_length	
0x100.bit_length	
(2**12-1).bit_length	
(2**12).bit_length	
(2**12+1).bit_length	

This method can be used to detect overflow in Array#pack as follows.
```
if n.bit_length < 32
    [n].pack("l") # no overflow
else
    raise "overflow"
end</pre>
```

## oiv(numeric) → integer

Performs integer division: returns integer result of dividing fix by numeric.

 $_{\odot}$  divmod(numeric)  $\rightarrow$  array

See Numeric#divmod.

### even? → true or false

Returns true if fix is an even number.

## fdiv(numeric) → float

Returns the floating point result of dividing fix by numeric.

```
654321.fdiv(13731)#=> 47.6528293642124654321.fdiv(13731.24)#=> 47.6519964693647
```

## inspect(p1 = v1)

Alias for: to\_s

### $\otimes$ abs $\rightarrow$ integer

#### magnitude → integer

Returns the absolute value of fix.

## $_{\odot}$ fix % other → real $_{\odot}$ modulo(other) → real

Returns fix modulo other.

See Numeric#divmod for more information.

## odd? → true or false

Returns true if fix is an odd number.

## isize → fixnum

Returns the number of bytes in the machine representation of fix.



## mext → integer

### Succ → integer

Returns the Integer equal to int + 1.

### $\odot$ to\_f $\rightarrow$ float

Converts fix to a Float.

## osto\_s(base=10) → string

Returns a string containing the representation of fix radix base (between 2 and 36).



12345.to_s(2)	#=>	"11000000111001"
12345.to_s(8)		
12345.to_s(10)		
12345.to_s(16)		
12345.to_s(36)		

Also aliased as: inspect

## rightarrow zero? $\rightarrow$ true or false

Returns true if fix is zero.

## $_{\odot}$ fix | integer $\rightarrow$ integer\_result

Bitwise OR.

## or a fix → integer

One's complement: returns a number where each bit is flipped.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class Float

Float objects represent inexact real numbers using the native architecture's double-precision floating point representation.

Floating point has a different arithmetic and is an inexact number. So you should know its esoteric system. see following:

- <u>docs.sun.com/source/806-</u> 3568/ncg\_goldberg.html
- <a>wiki.github.com/rdp/ruby\_tutorials\_core/rubytalk-faq#wiki-floats\_imprecise</a>
- en.wikipedia.org/wiki/Floating\_point#Accuracy\_pr

# **In Files**

- complex.c
- la numeric.c
- rational.c

# Parent

### Numeric

## Constants

DIG

The minimum number of significant decimal digits in a double-precision floating point.

Usually defaults to 15.

#### **EPSILON**

The difference between 1 and the smallest doubleprecision floating point number greater than 1.

Usually defaults to 2.2204460492503131e-16.

### INFINITY

An expression representing positive infinity.

## MANT\_DIG

The number of base digits for the double data type.

Usually defaults to 53.

### MAX

The largest possible integer in a double-precision floating point number.

Usually defaults to 1.7976931348623157e+308.

## MAX\_10\_EXP

The largest positive exponent in a double-precision floating point where 10 raised to this power minus 1.

Usually defaults to 308.

### MAX\_EXP

The largest possible exponent value in a doubleprecision floating point. Usually defaults to 1024.

#### MIN

<u>MIN</u>. 0.0.next\_float returns the smallest positive floating point number including denormalized numbers.

### MIN\_10\_EXP

The smallest negative exponent in a double-precision floating point where 10 raised to this power minus 1.

Usually defaults to -307.

#### MIN\_EXP

The smallest posable exponent value in a doubleprecision floating point.

Usually defaults to -1021.

### NAN

An expression representing a value which is "not a number".

### RADIX

The base of the floating point, or number of unique digits used to represent the number.

Usually defaults to 2 on most systems, which would represent a base-10 decimal.

#### ROUNDS

Rounding towards negative infinity

## **Public Instance Methods**

# rightarrow float % other → float

modulo(other) → float

Return the modulo after division of float by other.

6543.21.modulo(137)#=> 104.216543.21.modulo(137.24)#=> 92.9299999999999

## $_{\odot}$ float \* other $\rightarrow$ float

Returns a new float which is the product of float and other.

## $_{\odot}$ float \*\* other $\rightarrow$ float

Raises float to the power of other.

2.0\*\*3 #=> 8.0

### $_{\odot}$ float + other $\rightarrow$ float

Returns a new float which is the sum of float and other.

## $_{\odot}$ float - other $\rightarrow$ float

Returns a new float which is the difference of float and other.

#### Ising -float → float

Returns float, negated.

 $_{\odot}$  float / other  $\rightarrow$  float

Returns a new float which is the result of dividing float by other.

## Ioat < real → true or false</p>

Returns true if float is less than real.

The result of NaN < NaN is undefined, so the implementation-dependent value is returned.

## $_{\odot}$ float <= real → true or false

Returns true if float is less than or equal to real.

The result of NaN <= NaN is undefined, so the implementation-dependent value is returned.

## $_{\odot}$ float <=> real → -1, 0, +1 or nil

Returns -1, 0, +1 or nil depending on whether float is less than, equal to, or greater than real. This is the basis for the tests in Comparable.

The result of NaN <=> NaN is undefined, so the implementation-dependent value is returned.

nil is returned if the two values are incomparable.

### Ioat == obj → true or false

Returns true only if obj has the same value as float. Contrast this with  $\frac{\text{#eql?}}{\text{Heql?}}$ , which requires obj to be a Float.

The result of NaN == NaN is undefined, so the implementation-dependent value is returned.

## float == obj → true or false

Returns true only if obj has the same value as float. Contrast this with  $\frac{\text{#eql?}}{\text{Float.}}$ , which requires obj to be a Float.

The result of NaN == NaN is undefined, so the implementation-dependent value is returned.

1.0 == 1 #=> true

### $_{\odot}$ float > real → true or false

Returns true if float is greater than real.

The result of NaN > NaN is undefined, so the implementation-dependent value is returned.

## $_{\odot}$ float >= real $\rightarrow$ true or false

Returns true if float is greater than or equal to real.

The result of NaN >= NaN is undefined, so the implementation-dependent value is returned.

#### abs → float

#### $\odot$ magnitude $\rightarrow$ float

Returns the absolute value of float.

(-34.56).abs #=> 34.56 -34.56.abs #=> 34.56

rightarrow arg  $\rightarrow$  0 or float

#### $_{\odot}$ angle $\rightarrow$ 0 or float

#### $rightarrow phase \rightarrow 0 \text{ or float}$

Returns 0 if the value is positive, pi otherwise.

## rightarrow arg $\rightarrow$ 0 or float

## $_{\odot}$ angle $\rightarrow$ 0 or float

#### $\odot$ phase $\rightarrow$ 0 or float

Returns 0 if the value is positive, pi otherwise.

### ⊚ ceil → integer

Returns the smallest <u>Integer</u> greater than or equal to float.

1.2.ceil	
2.0.ceil	
(-1.2).ceil	
(-2.0).ceil	

## oerce(numeric) → array

Returns an array with both a numeric and a float represented as Float objects.

This is achieved by converting a numeric to a Float.



### $_{\odot}$ denominator $\rightarrow$ integer

Returns the denominator (always positive). The result is machine dependent.

See numerator.

## oivmod(numeric) → array

See <u>Numeric#divmod</u>.

42.0.divmod 6 #=> [7, 0.0] 42.0.divmod 5 #=> [8, 2.0]

## $_{\odot}$ eql?(obj) $\rightarrow$ true or false

Returns true only if obj is a <u>Float</u> with the same value as float. Contrast this with Float#==, which performs type conversions.

The result of NaN.eql?(NaN) is undefined, so the implementation-dependent value is returned.

1.0.eql?(1) #=> false

## fdiv(numeric) → float

## og quo(numeric) → float

Returns float / numeric, same as Float#/.

## inite? → true or false

Returns true if float is a valid IEEE floating point number (it is not infinite, and #nan? is false).

## $\odot$ floor $\rightarrow$ integer

Returns the largest integer less than or equal to float.

1.2.floor #=> 1 2.0.floor #=> 2 (-1.2).floor #=> -2 (-2.0).floor #=> -2

### $\otimes$ hash $\rightarrow$ integer

Returns a hash code for this float.

See also Object#hash.

 $_{\odot}$  infinite?  $\rightarrow$  nil, -1, +1

Return values corresponding to the value of float:

finite
nil
.Infinity
-1
+Infinity
1

For example:

(0.0).infinite?	
(-1.0/0.0).infinite?	
(+1.0/0.0).infinite?	

```
inspect()
Alias for: to_s
```

## ⊚abs → float

## $_{\odot}$ magnitude $\rightarrow$ float

Returns the absolute value of float.

(-34.56).abs #=> 34.56 -34.56.abs #=> 34.56

# float % other $\rightarrow$ float

## $_{\odot}$ modulo(other) $\rightarrow$ float

Return the modulo after division of float by other.



## rightarrow nan? $\rightarrow$ true or false

Returns true if float is an invalid IEEE floating point

number.

a = -1.0	
a.nan?	
a = 0.0/0.0	
a.nan?	

## onext\_float → float

Returns the next representable floating-point number.

Float::MAX.next\_float and Float::INFINITY.next\_float is Float::INFINITY.

Float::NAN.next\_float is Float::NAN.

For example:

р р р	0.01.next_float #=> 0.0 1.0.next_float #=> 1.0 100.0.next_float #=> 100	010000000000000002 0000000000000002 0.00000000
p p p	0.01.next_float - 0.01 1.0.next_float - 1.0 100.0.next_float - 100.0	<pre>#=&gt; 1.7347234759768076 #=&gt; 2.2204460492503136 #=&gt; 1.4210854715202004</pre>
f #=	= 0.01; 20.times { print	tf "%-20a %s\n", f, f.to
		010000000000000000
		0 0100000000000000000000000000000000000
		0 0100000000000000000000000000000000000
		0 0100000000000000000000000000000000000
		0 0100000000000000000000000000000000000
		0100000000000000000
		010000000000000000000000000000000000000
		01000000000000000021
		0100000000000000021
		0100000000000000023
		010000000000000020
		0100000000000000028
		01000000000000000
		LOTOCOCOCOCOCOCOCO



#### on numerator → integer

Returns the numerator. The result is machine dependent.



- $\odot$  arg  $\rightarrow$  0 or float
- $_{\odot}$  angle  $\rightarrow$  0 or float
- $\odot$  phase  $\rightarrow$  0 or float

Returns 0 if the value is positive, pi otherwise.

### $\odot$ prev\_float $\rightarrow$ float

Returns the previous representable floatint-point number.

(-Float::MAX).<u>#prev\_float</u> and (-Float::INFINITY).#prev\_float is -Float::INFINITY.

Float::NAN.prev\_float is Float::NAN.

For example:



## rightarrow fdiv(numeric) → float<math>rightarrow quo(numeric) → float

```
Returns float / numeric, same as Float#/.
```

## $_{\odot}$ rationalize([eps]) $\rightarrow$ rational

Returns a simpler approximation of the value (flt-|eps| <= result <= flt+|eps|). if the optional eps is not given, it will be chosen automatically.



See to\_r.

## opround([ndigits]) → integer or float

Rounds float to a given precision in decimal digits (default 0 digits).

Precision may be negative. Returns a floating point number when ndigits is more than zero.



#### $\odot$ to\_f $\rightarrow$ self

Since float is already a float, returns self.

### osto\_i → integer

- o to\_int → integer
- truncate → integer
   integer

Returns the float truncated to an Integer.

Synonyms are to\_i, to\_int, and truncate.

osto\_i → integer

oto\_int → integer

#### truncate → integer integer

Returns the float truncated to an Integer.

Synonyms are to\_i, to\_int, and truncate.

## oto\_r → rational

Returns the value as a rational.

NOTE: 0.3.to\_r isn't the same as '0.3'.to\_r. The latter is equivalent to '3/10'.#to\_r, but the former isn't so.

2.0.to_r	
2.5.to_r	
-0.75.to_r	
0.0.to_r	

See rationalize.

### os to\_s → string

Returns a string containing a representation of self. As well as a fixed or exponential form of the float, the call may return NaN, Infinity, and -Infinity.

Also aliased as: inspect

#### osto\_i → integer

#### to\_int → integer

truncate → integer
 integer

Returns the float truncated to an Integer.

Synonyms are to\_i, to\_int, and truncate.

### os zero? → true or false

Returns true if float is 0.0.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class FloatDomainError

Raised when attempting to convert special float values (in particular infinite or NaN) to numerical classes which don't support them.

```
Float::INFINITY.to_r
#=> FloatDomainError: Infinity
```

# In Files

📄 numeric.c

Parent

RangeError

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# module GC

The <u>GC</u> module provides an interface to Ruby's mark and sweep garbage collection mechanism.

Some of the underlying methods are also available via the <u>ObjectSpace</u> module.

You may obtain information about the operation of the <u>GC</u> through <u>GC::Profiler</u>.

**In Files** 

📄 gc.c

Constants

INTERNAL\_CONSTANTS

OPTS

## **Public Class Methods**

#### ount → Integer

The number of times GC occurred.

It returns the number of times <u>GC</u> occurred since the process started.

## $_{\odot}$ disable $\rightarrow$ true or false

Disables garbage collection, returning true if garbage collection was already disabled.

GC.disable	#=>	false
GC.disable		

## $_{\odot}$ enable $\rightarrow$ true or false

Enables garbage collection, returning true if garbage collection was previously disabled.

GC.disable	
GC.enable	
GC.enable	

## $\otimes$ latest\_gc\_info -> {:gc\_by $\rightarrow$ :newobj}

 $\otimes$  latest\_gc\_info(hash)  $\rightarrow$  hash

## $_{\odot}$ latest\_gc\_info(:major\_by) → :malloc

Returns information about the most recent garbage collection.

## malloc\_allocated\_size → Integer

Returns the size of memory allocated by malloc().

Only available if ruby was built with CALC\_EXACT\_MALLOC\_SIZE.

## malloc\_allocations → Integer

Returns the number of malloc() allocations.

Only available if ruby was built with CALC\_EXACT\_MALLOC\_SIZE.

```
⊚ start → nil
```

```
\odot garbage_collect \rightarrow nil
```

start(full\_mark: true, immediate\_sweep:

true) → nil

garbage\_collect(full\_mark: true,  $\implies$  immediate sweep: true)  $\rightarrow$  nil

Initiates garbage collection, unless manually disabled.

This method is defined with keyword arguments that default to true:



Use full\_mark: false to perform a minor <u>GC</u>. Use immediate\_sweep: false to defer sweeping (use lazy sweep).

Note: These keyword arguments are implementation and version dependent. They are not guaranteed to be future-compatible, and may be ignored if the underlying implementation does not support them.

```
⊚ stat → Hash
```

```
    stat(hash) → hash
```

## stat(:key) → Numeric

Returns a <u>Hash</u> containing information about the <u>GC</u>.

The hash includes information about internal statistics about <u>GC</u> such as:

```
{
    :count=>0,
    :heap_allocated_pages=>24,
    :heap_sorted_length=>24,
    :heap_allocatable_pages=>0,
    :heap_available_slots=>9783,
    :heap_live_slots=>7713,
```



The contents of the hash are implementation specific and may be changed in the future.

This method is only expected to work on C Ruby.

## stress → fixnum, true or false

Returns current status of GC stress mode.

## stress = flag → flag

Updates the GC stress mode.

When stress mode is enabled, the  $\underline{GC}$  is invoked at every  $\underline{GC}$  opportunity: all memory and object allocations.

Enabling stress mode will degrade performance, it is only for debugging.

flag can be true, false, or a fixnum bit-ORed following flags.

```
0x01:: no major GC
0x02:: no immediate sweep
0x04:: full mark after malloc/calloc/realloc
```

## $\odot$ verify\_internal\_consistency $\rightarrow$ nil

Verify internal consistency.

This method is implementation specific. Now this method checks generational consistency if RGenGC is supported.

## **Public Instance Methods**

```
⊚ start → nil
```

- garbage\_collect → nil start(full\_mark: true, immediate\_sweep: true) = nil
- ⊚true) → nil

garbage\_collect(full\_mark: true,

```
immediate_sweep: true) → nil
```

Initiates garbage collection, unless manually disabled.

This method is defined with keyword arguments that default to true:



Use full\_mark: false to perform a minor <u>GC</u>. Use immediate\_sweep: false to defer sweeping (use lazy sweep).

Note: These keyword arguments are implementation and version dependent. They are not guaranteed to be future-compatible, and may be ignored if the underlying implementation does not support them. Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# module GC::Profiler

The  $\underline{GC}$  profiler provides access to information on  $\underline{GC}$  runs including time, length and object space size.

Example:

GC::Profiler.enable

require 'rdoc/rdoc'

GC::Profiler.report

GC::Profiler.disable

See also <u>GC.count</u>, <u>GC.malloc\_allocated\_size</u> and <u>GC.malloc\_allocations</u>

## **In Files**

📄 gc.c

# **Public Class Methods**

## $\odot$ GC::Profiler.enable $\rightarrow$ nil

Starts the GC profiler.

## $\odot$ GC::Profiler.enabled? $\rightarrow$ true or false

The current status of GC profile mode.

## $\odot$ GC::Profiler.raw\_data $\rightarrow$ [Hash, ...]

Returns an <u>Array</u> of individual raw profile data Hashes ordered from earliest to latest by :GC\_INVOKE\_TIME.

For example:

,

The keys mean:

:GC\_TIME

Time elapsed in seconds for this GC run

#### :GC\_INVOKE\_TIME

Time elapsed in seconds from startup to when the GC was invoked

: HEAP\_USE\_SIZE Total bytes of heap used

:HEAP\_TOTAL\_SIZE Total size of heap in bytes : HEAP\_TOTAL\_OBJECTS Total number of objects

:GC\_IS\_MARKED Returns true if the GC is in mark phase

If ruby was built with GC\_PROFILE\_MORE\_DETAIL, you will also have access to the following hash keys:

- :GC\_MARK\_TIME
- :GC\_SWEEP\_TIME
- :ALLOCATE\_INCREASE
- :ALLOCATE\_LIMIT
- :HEAP\_USE\_PAGES
- :HEAP\_LIVE\_OBJECTS
- :HEAP\_FREE\_OBJECTS
- :HAVE\_FINALIZE

# GC::Profiler.report GC::Profiler.report(io)

Writes the ::result to \$stdout or the given IO object.

## GC::Profiler.result → String

Returns a profile data report such as:

GC 1 invo	okes.			
Index	Invoke	Time(sec)	Use	Size(byte)
1		0.012		159240
<b>.</b>				<u>ا</u>

## GC::Profiler.total\_time → float

The total time used for garbage collection in seconds

Generated by <u>RDoc</u> 3.12.2.

Generated with the Darkfish Rdoc Generator 3.

# class Hash

A <u>Hash</u> is a dictionary-like collection of unique keys and their values. Also called associative arrays, they are similar to Arrays, but where an <u>Array</u> uses integers as its index, a <u>Hash</u> allows you to use any object type.

Hashes enumerate their values in the order that the corresponding keys were inserted.

A <u>Hash</u> can be easily created by using its implicit form:

```
grades = { "Jane Doe" => 10, "Jim Doe" => 6 }
```

Hashes allow an alternate syntax form when your keys are always symbols. Instead of

You could write it as:

options = { font\_size: 10, font\_family: "Arial

Each named key is a symbol you can access in hash:

A <u>Hash</u> can also be created through its <u>::new</u> method:

```
grades = Hash.new
grades["Dorothy Doe"] = 9
```

Hashes have a *default value* that is returned when accessing keys that do not exist in the hash. If no default is set nil is used. You can set the default value by sending it as an argument to ::new:

grades = Hash.new(0)

Or by using the **default**= method:

```
grades = {"Timmy Doe" => 8}
grades.default = 0
```

Accessing a value in a <u>Hash</u> requires using its key:



### **Common Uses**

Hashes are an easy way to represent data structures, such as

```
books = {}
books[:matz] = "The Ruby Language"
books[:black] = "The Well-Grounded Rubyist"
```

Hashes are also commonly used as a way to have named parameters in functions. Note that no brackets are used below. If a hash is the last argument on a method call, no braces are needed, thus creating a really clean interface:

```
Person.create(name: "John Doe", age: 27)
def self.create(params)
  @name = params[:name]
  @age = params[:age]
end
```

## Hash Keys

Two objects refer to the same hash key when their hash value is identical and the two objects are eq1? to each other.

A user-defined class may be used as a hash key if the hash and eq1? methods are overridden to provide meaningful behavior. By default, separate instances refer to separate hash keys.

A typical implementation of hash is based on the object's data while eq1? is usually aliased to the overridden == method:

```
class Book
 attr_reader :author, :title
 def initialize(author, title)
 @author = author
 @title = title
```

```
end
```

```
def ==(other)
     self.class === other and
       other.author == @author and
       other.title == @title
  end
  alias eql? ==
  def hash
     @author.hash ^ @title.hash # XOR
  end
end
book1 = Book.new 'matz', 'Ruby in a Nutshell'
book2 = Book.new 'matz', 'Ruby in a Nutshell'
reviews = {}
reviews[book1] = 'Great reference!'
reviews[book2] = 'Nice and compact!'
reviews.length #=> 1
4
                                                    •
```

See also Object#hash and Object#eql?

## **In Files**

📄 hash.c

## Parent

Object

## **Included Modules**

## Enumerable

# **Public Class Methods**

```
Solution State State
```

```
Bash[ object ] → new_hash
```

Creates a new hash populated with the given objects.

Similar to the literal { *key* => *value*, ... }. In the first form, keys and values occur in pairs, so there must be an even number of arguments.

The second and third form take a single argument which is either an array of key-value pairs or an object convertible to a hash.



## onew → new\_hash

## onew(obj) → new\_hash

## $\otimes$ new {|hash, key| block } $\rightarrow$ new\_hash

Returns a new, empty hash. If this hash is subsequently accessed by a key that doesn't correspond to a hash entry, the value returned depends on the style of new used to create the hash. In the first form, the access returns nil. If *obj* is specified, this single object will be used for all *default values*. If a block is specified, it will be called with the hash object and the key, and should return the default value. It is the block's responsibility to store the value in the hash if required.

## onvert(obj) → hash or nil

Try to convert *obj* into a hash, using <u>#to\_hash</u> method. Returns converted hash or nil if *obj* cannot be converted for any reason.

```
Hash.try_convert({1=>2})  # => {1=>2}
Hash.try_convert("1=>2")  # => nil
```

# **Public Instance Methods**

## $_{\odot}$ hsh == other\_hash $\rightarrow$ true or false

Equality—Two hashes are equal if they each contain the same number of keys and if each key-value pair is equal to (according to <code>object#==</code>) the corresponding elements in the other hash.

```
h1 = { "a" => 1, "c" => 2 }
h2 = { 7 => 35, "c" => 2, "a" => 1 }
h3 = { "a" => 1, "c" => 2, 7 => 35 }
h4 = { "a" => 1, "d" => 2, "f" => 35 }
```

h1	==	h2	
h2	==	h3	
h3	==	h4	

## 

Element Reference—Retrieves the *value* object corresponding to the *key* object. If not found, returns the default value (see Hash::new for details).



```
rightarrow hsh[key] = value → value
rightarrow store(key, value) → value
```
## **Element Assignment**

Associates the value given by value with the key given by key.



key should not have its value changed while it is in use as a key (an unfrozen String passed as a key will be duplicated and frozen).

a = "a" b = "b".freeze h = { a => 100, b => 200 } h.key(100).equal? a #=> false h.key(200).equal? b #=> true

any? [{ |(key, value)| block }] → true or false See also Enumerable#any?

### assoc(obj) → an\_array or nil

Searches through the hash comparing *obj* with the key using ==. Returns the key-value pair (two elements array) or nil if no match is found. See Array#assoc.



## ⊚ clear → hsh

4

Removes all key-value pairs from hsh.



۲

## $\odot$ compare\_by\_identity $\rightarrow$ hsh

Makes *hsh* compare its keys by their identity, i.e. it will consider exact same objects as same keys.



## $\odot$ compare\_by\_identity? $\rightarrow$ true or false

Returns true if *hsh* will compare its keys by their identity. Also see Hash#compare\_by\_identity.

## ⊕ default(key=nil) → obj

Returns the default value, the value that would be returned by <u>hsh</u> if *key* did not exist in *hsh*. See also Hash::new and Hash#default=.

h = Hash.new h.default h.default(2)	
h = Hash.new("cat") h.default h.default(2)	



## ⊕ default = obj → obj

Sets the default value, the value returned for a key that does not exist in the hash. It is not possible to set the default to a Proc that will be executed on each key lookup.



## $_{\odot}$ default\_proc $\rightarrow$ anObject

If Hash::new was invoked with a block, return that block, otherwise return nil.



## odefault\_proc = proc\_obj or nil

Sets the default proc to be executed on each failed key lookup.

h.default\_proc = proc do |hash, key|



## $_{\odot}$ delete(key) → value $_{\odot}$ delete(key) {| key | block } → value

Deletes the key-value pair and returns the value from *hsh* whose key is equal to *key*. If the key is not found, returns the *default value*. If the optional code block is given and the key is not found, pass in the key and return the result of *block*.



## 

Deletes every key-value pair from *hsh* for which *block* evaluates to true.

If no block is given, an enumerator is returned instead.



 $_{\odot}$  each {| key, value | block } → hsh  $_{\odot}$  each\_pair {| key, value | block } → hsh

- a each  $\rightarrow$  an enumerator
- $\Rightarrow$  each  $\Rightarrow$  an\_enumerator
- $\otimes$  each\_pair  $\rightarrow$  an\_enumerator

Calls block once for each key in hsh, passing the

key-value pair as parameters.

If no block is given, an enumerator is returned instead.



produces:

a is 100 b is 200

## $_{\odot}$ each\_key {| key | block } → hsh $_{\odot}$ each\_key → an\_enumerator

Calls *block* once for each key in *hsh*, passing the key as a parameter.

If no block is given, an enumerator is returned instead.



produces:



```
_{\odot} each {| key, value | block } → hsh
```

- $_{\odot}$  each\_pair {| key, value | block }  $\rightarrow$  hsh
- each → an\_enumerator

## $\otimes$ each\_pair $\rightarrow$ an\_enumerator

Calls *block* once for each key in *hsh*, passing the key-value pair as parameters.

If no block is given, an enumerator is returned instead.



## $_{\odot}$ each\_value {| value | block } → hsh $_{\odot}$ each\_value → an\_enumerator

Calls *block* once for each key in *hsh*, passing the value as a parameter.

If no block is given, an enumerator is returned instead.

```
h = { "a" => 100, "b" => 200 }
h.each_value {|value| puts value }
```

produces:

100 200

## empty? → true or false

Returns true if hsh contains no key-value pairs.



## $_{\odot}$ eql?(other) $\rightarrow$ true or false

Returns true if *hash* and *other* are both hashes with the same content.

```
_{\odot} fetch(key [, default] ) → obj
_{\odot} fetch(key) {| key | block } → obj
```

Returns a value from the hash for the given key. If the key can't be found, there are several options: With no other arguments, it will raise an KeyError exception; if *default* is given, then that will be returned; if the optional code block is specified, then that will be run and its result returned.



The following example shows that an exception is raised if the key is not found and a default value is not supplied.

```
h = { "a" => 100, "b" => 200 }
h.fetch("z")
```

produces:

```
prog.rb:2:in `fetch': key not found (KeyError)
  from prog.rb:2
```

## I flatten → an\_array I flatten(level) → an\_array

Returns a new array that is a one-dimensional flattening of this hash. That is, for every key or value that is an array, extract its elements into the new array. Unlike <u>Array#flatten</u>, this method does not flatten recursively by default. The optional *level* argument determines the level of recursion to flatten.



```
_{\odot} has_key?(key) \rightarrow true or false
```

- $\odot$  include?(key)  $\rightarrow$  true or false
- $\otimes$  key?(key)  $\rightarrow$  true or false
- $_{\odot}$  member?(key)  $\rightarrow$  true or false

Returns true if the given key is present in hsh.



## $_{\odot}$ has\_value?(value) → true or false $_{\odot}$ value?(value) → true or false

Returns true if the given value is present for some key in *hsh*.

## 

Compute a hash-code for this hash. Two hashes with the same content will have the same hash code (and will compare using eq1?).

See also Object#hash.

has\_key?(key) → true or false
 include?(key) → true or false
 key?(key) → true or false
 member?(key) → true or false
 member?(key) → true or false
 Returns true if the given key is present in *hsh*.
 h = { "a" => 100, "b" => 200 }
 h.has\_key?("a") #=> true

#### 

## os → string

## $\odot$ inspect $\rightarrow$ string

Return the contents of this hash as a string.



Also aliased as: to\_s

## invert → new\_hash

Returns a new hash created by using *hsh*'s values as keys, and the keys as values.



## weep\_if {| key, value | block } → hsh keep\_if → an\_enumerator

Deletes every key-value pair from *hsh* for which *block* evaluates to false.

If no block is given, an enumerator is returned instead.

## Skey(value) → key

Returns the key of an occurrence of a given value. If the value is not found, returns nil.



- $_{\odot}$  has\_key?(key) → true or false
- $_{\odot}$  include?(key)  $\rightarrow$  true or false
- $\otimes$  key?(key)  $\rightarrow$  true or false
- member?(key) → true or false

Returns true if the given key is present in hsh.



## 

Returns a new array populated with the keys from this hash. See also Hash#values.



## length → fixnum

## isize → fixnum

Returns the number of key-value pairs in the hash.



- $\otimes$  has\_key?(key)  $\rightarrow$  true or false
- $_{\odot}$  include?(key)  $\rightarrow$  true or false
- $\otimes$  key?(key)  $\rightarrow$  true or false
- ∞ member?(key) → true or false Returns true if the given key is present in *hsh*.



# merge(other\_hash) → new\_hash merge(other\_hash){|key, oldval, newval| block} → new\_hash

Returns a new hash containing the contents of *other\_hash* and the contents of *hsh*. If no block is specified, the value for entries with duplicate keys will be that of *other\_hash*. Otherwise the value for each duplicate key is determined by calling the block with the key, its value in *hsh* and its value in *other\_hash*.



```
merge!(other_hash) → hsh
update(other_hash) → hsh
merge!(other_hash){|key, oldval, newval|
block} → hsh
update(other_hash){|key, oldval, newval|
block} → hsh
Adds the contents of other_hash to hsh. If no block is
specified, entries with duplicate keys are overwritten
with the values from other_hash, otherwise the value
of each duplicate key is determined by calling the
block with the key, its value in hsh and its value in
other_hash.
```



## $rassoc(obj) \rightarrow an_array or nil$

Searches through the hash comparing *obj* with the value using ==. Returns the first key-value pair (two-element array) that matches. See also Array#rassoc.



## $_{\odot}$ rehash $\rightarrow$ hsh

Rebuilds the hash based on the current hash values for each key. If values of key objects have changed since they were inserted, this method will reindex *hsh*. If Hash#rehash is called while an iterator is traversing the hash, an RuntimeError will be raised in the iterator.



 $_{\odot}$  reject {|key, value| block}  $\rightarrow$  a\_hash

## oreject → an\_enumerator

Returns a new hash consisting of entries for which the block returns false.

If no block is given, an enumerator is returned instead.



## $_{\odot}$ reject! {| key, value | block } → hsh or nil $_{\odot}$ reject! → an\_enumerator

Equivalent to Hash#delete\_if, but returns nil if no changes were made.

## $_{\odot}$ replace(other\_hash) $\rightarrow$ hsh

Replaces the contents of *hsh* with the contents of *other\_hash*.



## $_{\odot}$ select {|key, value| block} → a\_hash $_{\odot}$ select → an\_enumerator

Returns a new hash consisting of entries for which the block returns true.

If no block is given, an enumerator is returned instead.



## $_{\odot}$ select! {| key, value | block } → hsh or nil $_{\odot}$ select! → an\_enumerator

Equivalent to Hash#keep\_if, but returns nil if no changes were made.

## $_{\odot}$ shift $\rightarrow$ anArray or obj

Removes a key-value pair from *hsh* and returns it as the two-item array [*key, value*], or the hash's default value if the hash is empty.



## length → fixnum size → fixnum

Returns the number of key-value pairs in the hash.



rightarrow hsh[key] = value → value rightarrow store(key, value) → value

## **Element Assignment**

Associates the value given by value with the key given by key.



key should not have its value changed while it is in use as a key (an unfrozen String passed as a key will be duplicated and frozen).

a = "a" b = "b".freeze h = { a => 100, b => 200 } h.key(100).equal? a #=> false h.key(200).equal? b #=> true

#### to\_a → array

Converts *hsh* to a nested array of [*key, value*] arrays.



## $_{\odot}$ to\_h $\rightarrow$ hsh or new\_hash

Returns self. If called on a subclass of  $\underline{\text{Hash}}$ , converts the receiver to a  $\underline{\text{Hash}}$  object.

## $\odot$ to\_hash $\rightarrow$ hsh

Returns self.

to\_s()
 Alias for: inspect

```
\otimes merge!(other_hash) \rightarrow hsh
```

```
_{\odot} update(other_hash) \rightarrow hsh
```

merge!(other\_hash){|key, oldval, newval| block}  $\rightarrow$  hsh

Adds the contents of *other\_hash* to *hsh*. If no block is specified, entries with duplicate keys are overwritten with the values from *other\_hash*, otherwise the value of each duplicate key is determined by calling the block with the key, its value in *hsh* and its value in *other\_hash*.



## $_{\odot}$ has\_value?(value) → true or false $_{\odot}$ value?(value) → true or false

Returns true if the given value is present for some key in *hsh*.



### $\odot$ values $\rightarrow$ array

Returns a new array populated with the values from *hsh*. See also Hash#keys.



## os values\_at(key, ...) → array

Return an array containing the values associated with the given keys. Also see Hash.select.



Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class IO

The <u>IO</u> class is the basis for all input and output in Ruby. An I/O stream may be *duplexed* (that is, bidirectional), and so may use more than one native operating system stream.

Many of the examples in this section use the File class, the only standard subclass of IO. The two classes are closely associated. Like the File class, the Socket library subclasses from IO (such as TCPSocket or UDPSocket).

The <u>Kernel#open</u> method can create an <u>IO</u> (or <u>File</u>) object for these types of arguments:

- A plain string represents a filename suitable for the underlying operating system.
- A string starting with "|" indicates a subprocess. The remainder of the string following the "|" is invoked as a process with appropriate input/output channels connected to it.
- A string equal to "|-" will create another Ruby instance as a subprocess.

The IO may be opened with different file modes (read-only, write-only) and encodings for proper conversion. See <u>::new</u> for these options. See <u>Kernel#open</u> for details of the various command formats described above.

::popen, the Open3 library, or Process#spawn may also be used to communicate with subprocesses through an IO.

Ruby will convert pathnames between different operating system conventions if possible. For instance, on a Windows system the filename "/gumby/ruby/test.rb" will be opened as "\gumby\ruby\test.rb". When specifying a Windows-style filename in a Ruby string, remember to escape the backslashes:

### "c:\\gumby\\ruby\\test.rb"

Our examples here will use the Unix-style forward slashes; File::ALT\_SEPARATOR can be used to get the platform-specific separator character.

The global constant ARGF (also accessible as \$<) provides an IO-like stream which allows access to all files mentioned on the command line (or STDIN if no files are mentioned). ARGF#path and its alias ARGF#filename are provided to access the name of the file currently being read.

## io/console

The io/console extension provides methods for interacting with the console. The console can be accessed from IO.console or the standard input/output/error IO objects.

Requiring io/console adds the following methods:

- IO::console
- IO#raw
- IO#raw!
- IO#cooked
- IO#cooked!
- IO#getch
- IO#echo=
- IO#echo?
- IO#noecho
- IO#winsize
- IO#winsize=
- IO#iflush
- IO#ioflush
- IO#oflush

Example:

require 'io/console' rows, columns = \$stdin.winsize puts "Your screen is #{columns} wide and #{row 4

#### **[ ↓**]

## **In Files**

file.c

📄 io.c

## Parent

**Object** 

## **Included Modules**

File::Constants

Enumerable

## Constants

## **EWOULDBLOCKWaitReadable**

EAGAINWaitReadable

## **EWOULDBLOCKWaitWritable**

EAGAINWaitWritable

### SEEK\_CUR

Set I/O position from the current position

### SEEK\_DATA

Set I/O position to the next location containing data

▶

### SEEK\_END

Set I/O position from the end

## SEEK\_HOLE

Set I/O position to the next hole

## SEEK\_SET

Set I/O position from the beginning

## **Public Class Methods**

## $_{\odot}$ binread(name, [length [, offset]] ) $\rightarrow$ string

Opens the file, optionally seeks to the given *offset*, then returns *length* bytes (defaulting to the rest of the file). binread ensures the file is closed before returning. The open mode would be "rb:ASCII-8BIT".



binwrite(name, string, [offset]) → fixnum
 binwrite(name, string, [offset], open\_args)
 → fixnum

Same as IO.write except opening the file in binary mode and ASCII-8BIT encoding ("wb:ASCII-8BIT").

copy\_stream(src, dst)

# copy\_stream(src, dst, copy\_length) copy\_stream(src, dst, copy\_length, src\_offset)

::copy\_stream copies *src* to *dst*. *src* and *dst* is either

a filename or an IO.

This method returns the number of bytes copied.

If optional arguments are not given, the start position of the copy is the beginning of the filename or the current file offset of the <u>IO</u>. The end position of the copy is the end of file.

If *copy\_length* is given, No more than *copy\_length* bytes are copied.

If *src\_offset* is given, it specifies the start position of the copy.

When *src\_offset* is specified and *src* is an <u>IO</u>, ::copy\_stream doesn't move the current file offset.

## ⊚ for\_fd(fd, mode [, opt]) → io

Synonym for IO.new.

foreach(name, sep=\$/ [, open\_args]) {|line|  $\textcircled{block} \rightarrow nil$ 

foreach(name, limit [, open\_args]) {|line|

 $\bigotimes$  block }  $\rightarrow$  nil

foreach(name, sep, limit [, open\_args]) {|line|  $\otimes$  block }  $\rightarrow$  nil

oforeach(...) → an\_enumerator

Executes the block for every line in the named I/O port, where lines are separated by *sep*.

If no block is given, an enumerator is returned instead.

```
IO.foreach("testfile") {|x| print "GOT ", x }
```

produces:



```
GOT This is line two
GOT This is line three
GOT And so on...
```

If the last argument is a hash, it's the keyword argument to open. See IO.read for detail.

## $_{\odot}$ new(fd [, mode] [, opt]) $\rightarrow$ io

Returns a new <u>IO</u> object (a stream) for the given integer file descriptor fd and mode string. opt may be used to specify parts of mode in a more readable fashion. See also ::sysopen and ::for\_fd.

<u>::new</u> is called by various File and IO opening methods such as ::open, Kernel#open, and File.open.

## **Open Mode**

When mode is an integer it must be combination of the modes defined in File::Constants (+File::RDONLY+, +File::WRONLY | File::CREAT+). See the open(2) man page for more information.

When mode is a string it must be in one of the following forms:

```
fmode
fmode ":" ext_enc
fmode ":" ext_enc ":" int_enc
fmode ":" "BOM|UTF-*"
```

fmode is an <u>IO</u> open mode string, ext\_enc is the external encoding for the <u>IO</u> and int\_enc is the internal encoding.

#### IO Open Mode

Ruby allows the following open modes:



```
"r+" Read-write, starts at beginning of file.
"w" Write-only, truncates existing file
  to zero length or creates a new file for wri
"w+" Read-write, truncates existing file to zero
  or creates a new file for reading and writir
"a" Write-only, each write call appends data at
  Creates a new file for writing if file does
"a+" Read-write, each write call appends data at
  Creates a new file for reading and writing i
  not exist.
```

The following modes must be used separately, and along with one or more of the modes seen above.



When the open mode of original <u>IO</u> is read only, the mode cannot be changed to be writable. Similarly, the open mode cannot be changed from write only to readable.

When such a change is attempted the error is raised in different locations according to the platform.

## **IO** Encoding

When ext\_enc is specified, strings read will be tagged by the encoding when reading, and strings output will be converted to the specified encoding when writing.

When ext\_enc and int\_enc are specified read strings will be converted from ext\_enc to int\_enc upon input, and written strings will be converted from int\_enc to ext\_enc upon output. See <u>Encoding</u> for further details of transcoding on input and output.

If "BOM|UTF-8", "BOM|UTF-16LE" or "BOM|UTF16-BE" are used, ruby checks for a Unicode BOM in the input document to help determine the encoding. For UTF-16 encodings the file open mode must be binary. When present, the BOM is stripped and the external encoding from the BOM is used. When the BOM is missing the given Unicode encoding is used as ext\_enc. (The BOM-set encoding option is case insensitive, so "bom|utf-8" is also valid.)

## **Options**

opt can be used instead of mode for improved readability. The following keys are supported:

### :mode

Same as mode parameter

## :external\_encoding

External encoding for the  $\underline{IO}$ . "-" is a synonym for the default external encoding.

### :internal\_encoding

Internal encoding for the IO. "-" is a synonym for the default internal encoding.

If the value is nil no conversion occurs.

### :encoding

Specifies external and internal encodings as "extern:intern".

### :textmode

If the value is truth value, same as "t" in argument mode.

### :binmode

If the value is truth value, same as "b" in argument

mode.

#### :autoclose

If the value is false, the fd will be kept open after this IO instance gets finalized.

Also, opt can have same keys in <u>String#encode</u> for controlling conversion between the external encoding and the internal encoding.

## Example 1

```
fd = I0.sysopen("/dev/tty", "w")
a = I0.new(fd,"w")
$stderr.puts "Hello"
a.puts "World"
```

Produces:

Hello World

## Example 2

Both of above print "Hello, World!" in UTF-16LE to standard error output with converting EOL generated by puts to CR.

## open(fd, mode="r" [, opt]) → io open(fd, mode="r" [, opt]) { |io| block } → obj

With no associated block, IO.open is a synonym for ::new. If the optional code block is given, it will be passed io as an argument, and the IO object will automatically be closed when the block terminates. In this instance, ::open returns the value of the block.

See <u>::new</u> for a description of the fd, mode and opt parameters.

## ◎ pipe → [read\_io, write\_io]

pipe(ext\_enc) → [read\_io, write\_io] pipe("ext\_enc:int\_enc" [, opt]) → [read\_io, write\_io]

pipe(ext\_enc, int\_enc [, opt]) → [read\_io, write\_io]

## pipe(...) {|read\_io, write\_io| ... }

Creates a pair of pipe endpoints (connected to each other) and returns them as a two-element array of 10 objects: [ *read\_io*, *write\_io* ].

If a block is given, the block is called and returns the value of the block. *read\_io* and *write\_io* are sent to the block as arguments. If read\_io and write\_io are not closed when the block exits, they are closed. i.e. closing read\_io and/or write\_io doesn't cause an error.

Not available on all platforms.

If an encoding (encoding name or encoding object) is specified as an optional argument, read string from pipe is tagged with the encoding specified. If the argument is a colon separated two encoding names "A:B", the read string is converted from encoding A (external encoding) to encoding B (internal encoding), then tagged with B. If two optional arguments are specified, those must be encoding objects or encoding names, and the first one is the external encoding, and the second one is the internal encoding. If the external encoding and the internal encoding is specified, optional hash argument specify the conversion option.

In the example below, the two processes close the ends of the pipe that they are not using. This is not just a cosmetic nicety. The read end of a pipe will not generate an end of file condition if there are any writers with the pipe still open. In the case of the parent process, the rd.read will never return if it does not first issue a wr.close.

```
rd, wr = IO.pipe
if fork
  wr.close
  puts "Parent got: <#{rd.read}>"
  rd.close
  Process.wait
else
  rd.close
  puts "Sending message to parent"
  wr.write "Hi Dad"
  wr.close
end
```

produces:

```
Sending message to parent
Parent got: <Hi Dad>
```

popen([env,] cmd, mode="r" [, opt]) → io popen([env,] cmd, mode="r" [, opt]) {|io| block } → obj

Runs the specified command as a subprocess; the

subprocess's standard input and output will be connected to the returned 10 object.

The PID of the started process can be obtained by **#pid** method.

*cmd* is a string or an array as follows.

cmd: "_"	:	for
commandline [env, cmdname, arg1,, opts] [onv, [cmdnamo, argv0], arg1,, onts]	:	com
(env and opts are optional.)	•	

If *cmd* is a string "-", then a new instance of Ruby is started as the subprocess.

If *cmd* is an Array of string, then it will be used as the subprocess's argv bypassing a shell. The array can contains a hash at first for environments and a hash at last for options similar to spawn.

The default mode for the new file object is "r", but *mode* may be set to any of the modes listed in the description for class <u>IO</u>. The last argument *opt* qualifies *mode*.

```
# set IO encoding
IO.popen("nkf -e filename", :external_encoding=>'
euc_jp_string = nkf_io.read
}
# merge standard output and standard error using
# spawn option. See the document of Kernel.spawn
IO.popen(["ls", "/", :err=>[:child, :out]]) {|ls_
ls_result_with_error = ls_io.read
}
# spawn options can be mixed with IO options
IO.popen(["ls", "/"], :err=>[:child, :out]) {|ls_
ls_result_with_error = ls_io.read
}
```

. ◀ ]

Raises exceptions which IO.pipe and Kernel.spawn raise.

If a block is given, Ruby will run the command as a child connected to Ruby with a pipe. Ruby's end of the pipe will be passed as a parameter to the block. At the end of block, Ruby closes the pipe and sets \$?. In this case IO.popen returns the value of the block.

If a block is given with a *cmd* of "-", the block will be run in two separate processes: once in the parent, and once in a child. The parent process will be passed the pipe object as a parameter to the block, the child version of the block will be passed nil, and the child's standard in and standard out will be connected to the parent through the pipe. Not available on all platforms.

```
f = I0.popen("uname")
p f.readlines
f.close
puts "Parent is #{Process.pid}"
I0.popen("date") { |f| puts f.gets }
I0.popen("-") {|f| $stderr.puts "#{Process.pid} i
p $?
I0.popen(%wsed -e s|^|<foo>| -e s&$&;zot;&", "r+'
f.puts "bar"; f.close_write; puts f.gets
}
```

produces:

```
["Linux\n"]
Parent is 21346
Thu Jan 15 22:41:19 JST 2009
21346 is here, f is #<I0:fd 3>
21352 is here, f is nil
#<Process::Status: pid 21352 exit 0>
<foo>bar;zot;
```

## read(name, [length [, offset]] [, opt] ) $\rightarrow$ $_{\textcircled{}}$ string

Opens the file, optionally seeks to the given offset, then returns length bytes (defaulting to the rest of the file). read ensures the file is closed before returning.

## **Options**

The options hash accepts the following keys:

## encoding

string or encoding

Specifies the encoding of the read string. encoding: will be ignored if length is specified. See Encoding.aliases for possible encodings.

## mode

string

Specifies the mode argument for open(). It must start with an "r" otherwise it will cause an error. See <u>::new</u> for the list of possible modes.

## open\_args

array of strings

Specifies arguments for open() as an array. This key can not be used in combination with either encoding: Or mode:.

Examples:



readlines(name, sep=\$/ [, open\_args]) →

🏐 array

# readlines(name, limit [, open\_args]) → array readlines(name, sep, limit [, open\_args]) → array

Reads the entire file specified by *name* as individual lines, and returns those lines in an array. Lines are separated by *sep*.

If the last argument is a hash, it's the keyword argument to open. See IO.read for detail.

## select(read\_array

- [, write\_array
- 🚳 [, error\_array

## $_{\odot}$ [, timeout]]]) $\rightarrow$ array or nil

Calls select(2) system call. It monitors given arrays of 10 objects, waits one or more of 10 objects ready for reading, are ready for writing, and have pending exceptions respectively, and returns an array that contains arrays of those <u>IO</u> objects. It will return nil if optional *timeout* value is given and no 10 object is ready in *timeout* seconds.

IO.select peeks the buffer of IO objects for testing readability. If the IO buffer is not empty, IO.select immediately notify readability. This "peek" is only happen for IO objects. It is not happen for IO-like objects such as OpenSSL::SSLS::SSLSocket.

The best way to use IO.select is invoking it after nonblocking methods such as read\_nonblock, write\_nonblock, etc. The methods raises an exception which is extended by IO:::WaitReadable Or IO::WaitWritable. The modules notify how the caller should wait with IO.select. If IO::WaitReadable is raised, the caller should wait for reading. If IO::WaitWritable is raised, the caller should wait for writing.

So, blocking read (readpartial) can be emulated using read\_nonblock and IO.select as follows:

```
begin
  result = io_like.read_nonblock(maxlen)
rescue I0::WaitReadable
  I0.select([io_like])
  retry
rescue I0::WaitWritable
  I0.select(nil, [io_like])
  retry
end
```

Especially, the combination of nonblocking methods and IO.select is preferred for IO like objects such as OpenSSL::SSL::SSLSocket. It has to\_io method to return underlying IO object. IO.select calls to\_io to obtain the file descriptor to wait.

This means that readability notified by IO.select doesn't mean readability from OpenSSL::SSL::SSLSocket Object.

Most possible situation is <code>OpenSSL::SSLSocket</code> buffers some data. <code>IO.select</code> doesn't see the buffer. So <code>IO.select</code> can block when

OpenSSL::SSL::SSLSocket#readpartial doesn't block.

However several more complicated situation exists.

SSL is a protocol which is sequence of records. The record consists multiple bytes. So, the remote side of SSL sends a partial record, IO.select notifies readability but OpenSSL::SSLSocket cannot

decrypt a byte and
openSSL::SSL::SSLSocket#readpartial will blocks.

Also, the remote side can request SSL renegotiation which forces the local SSL engine writes some data. This means <code>OpenSSL::SSLSocket#readpartial</code> may invoke write system call and it can block. In such situation,

OpenSSL::SSL::SSLSocket#read\_nonblock raises IO::WaitWritable instead of blocking. So, the caller should wait for ready for writability as above example.

The combination of nonblocking methods and IO.select is also useful for streams such as tty, pipe socket socket when multiple process read form a stream.

Finally, Linux kernel developers doesn't guarantee that readability of select(2) means readability of following read(2) even for single process. See select(2) manual on GNU/Linux system.

Invoking IO.select before IO#readpartial works well in usual. However it is not the best way to use IO.select.

The writability notified by select(2) doesn't show how many bytes writable. IO#write method blocks until given whole string is written. So, IO#write(two or more bytes) can block after writability is notified by IO.select. IO#write\_nonblock is required to avoid the blocking.

Blocking write (write) can be emulated using write\_nonblock and IO.select as follows: IO::WaitReadable should also be rescued for SSL renegotiation in OpenSSL::SSL::SSLSocket.

#### while 0 < string.bytesize

```
begin
  written = io_like.write_nonblock(string)
rescue IO::WaitReadable
  IO.select([io_like])
  retry
rescue IO::WaitWritable
  IO.select(nil, [io_like])
  retry
end
string = string.byteslice(written..-1)
end
```

#### **Parameters**

#### read\_array

an array of 10 objects that wait until ready for read

#### write\_array

an array of 10 objects that wait until ready for write

#### error\_array

an array of 10 objects that wait for exceptions

#### timeout

a numeric value in second

### Example

```
rp, wp = I0.pipe
mesg = "ping "
100.times {
    # I0.select follows IO#read. Not the best way
    rs, ws, = I0.select([rp], [wp])
    if r = rs[0]
       ret = r.read(5)
       print ret
       case ret
       when /ping/
        mesg = "pong\n"
       when /pong/
        mesg = "ping "
       end
       end
```


produces:

ping pong ping pong ping pong (snipped) ping		
ping		

# $_{\odot}$ sysopen(path, [mode, [perm]]) → fixnum

Opens the given path, returning the underlying file descriptor as a Fixnum.



# onvert(obj) → io or nil

Try to convert *obj* into an <u>IO</u>, using <u>#to\_io</u> method. Returns converted <u>IO</u> or nil if *obj* cannot be converted for any reason.



# write(name, string, [offset] ) → fixnum write(name, string, [offset], open\_args ) → fixnum

Opens the file, optionally seeks to the given offset,

writes *string*, then returns the length written. write ensures the file is closed before returning. If *offset* is not given, the file is truncated. Otherwise, it is not truncated.

If the last argument is a hash, it specifies option for internal open(). The key would be the following. open\_args: is exclusive to others.

```
encoding: string or encoding
specifies encoding of the read string. encoding
if length is specified.
mode: string
specifies mode argument for open(). it should s
otherwise it would cause error.
perm: fixnum
specifies perm argument for open().
open_args: array
specifies arguments for open() as an array.
IO.write("testfile", "0123456789", 20) # => 10
# File could contain: "This is line one\nThi0:
IO.write("testfile", "0123456789") #=> 10
# File would now read: "0123456789"
```

# **Public Instance Methods**

```
rightarrow ios << obj → ios

<u>String</u> Output—Writes obj to ios. obj will be converted

to a string using to_s.
```

```
$stdout << "Hello " << "world!\n"</pre>
```

produces:

Hello world!

# $_{\odot}$ advise(advice, offset=0, len=0) $\rightarrow$ nil

Announce an intention to access data from the current file in a specific pattern. On platforms that do not support the *posix\_fadvise(2)* system call, this method is a no-op.

advice is one of the following symbols:

#### :normal

No advice to give; the default assumption for an open file.

#### :sequential

The data will be accessed sequentially with lower offsets read before higher ones.

#### :random

The data will be accessed in random order.

#### :willneed

The data will be accessed in the near future.

#### :dontneed

The data will not be accessed in the near future.

#### :noreuse

The data will only be accessed once.

The semantics of a piece of advice are platformdependent. See *man 2 posix\_fadvise* for details.

"data" means the region of the current file that begins at *offset* and extends for *len* bytes. If *len* is 0, the region ends at the last byte of the file. By default, both *offset* and *len* are 0, meaning that the advice applies to the entire file. If an error occurs, one of the following exceptions will be raised:

#### **IOError**

The 10 stream is closed.

#### Errno::EBADF

The file descriptor of the current file is invalid.

#### Errno::EINVAL

An invalid value for *advice* was given.

#### Errno::ESPIPE

The file descriptor of the current file refers to a FIFO or pipe. (Linux raises Errno::EINVAL in this case).

#### TypeError

Either *advice* was not a <u>Symbol</u>, or one of the other arguments was not an Integer.

#### RangeError

One of the arguments given was too big/small.

#### This list is not exhaustive; other Errno

exceptions are also possible.

#### autoclose = bool → true or false

Sets auto-close flag.

```
f = open("/dev/null")
IO.for_fd(f.fileno)
# ...
f.gets # may cause IOError
f = open("/dev/null")
IO.for_fd(f.fileno).autoclose = true
# ...
f.gets # won't cause IOError
```

#### autoclose? → true or false

Returns true if the underlying file descriptor of *ios* will be closed automatically at its finalization, otherwise false.

#### $\odot$ binmode $\rightarrow$ ios

Puts *ios* into binary mode. Once a stream is in binary mode, it cannot be reset to nonbinary mode.

- newline conversion disabled
- encoding conversion disabled
- content is treated as ASCII-8BIT

#### binmode? → true or false

Returns true if ios is binmode.

#### bytes()

This is a deprecated alias for each\_byte.

#### 🚳 chars()

This is a deprecated alias for each\_char.

#### $\odot$ close $\rightarrow$ nil

Closes *ios* and flushes any pending writes to the operating system. The stream is unavailable for any further data operations; an IOError is raised if such an attempt is made. I/O streams are automatically closed when they are claimed by the garbage collector.

If ios is opened by IO.popen, close sets \$?.

### close\_on\_exec = bool → true or false

Sets a close-on-exec flag.



Ruby sets close-on-exec flags of all file descriptors by default since Ruby 2.0.0. So you don't need to set by yourself. Also, unsetting a close-on-exec flag can cause file descriptor leak if another thread use fork() and exec() (via system() method for example). If you really needs file descriptor inheritance to child process, use spawn()'s argument such as fd=>fd.

#### close\_on\_exec? → true or false

Returns true if ios will be closed on exec.



# ose\_read → nil

Closes the read end of a duplex I/O stream (i.e., one that contains both a read and a write stream, such as a pipe). Will raise an IDError if the stream is not duplexed.

```
f = I0.popen("/bin/sh","r+")
f.close_read
f.readlines
```

produces:



#### or close\_write → nil

Closes the write end of a duplex I/O stream (i.e., one that contains both a read and a write stream, such as a pipe). Will raise an IDError if the stream is not duplexed.

```
f = I0.popen("/bin/sh","r+")
f.close_write
f.print "nowhere"
```

produces:

```
prog.rb:3:in `write': not opened for writing (IOE
from prog.rb:3:in `print'
from prog.rb:3
```

#### $\odot$ closed? $\rightarrow$ true or false

Returns true if *ios* is completely closed (for duplex streams, both reader and writer), false otherwise.



# codepoints()

This is a deprecated alias for each\_codepoint.

- $_{\odot}$  each(sep=\$/) {|line| block }  $\rightarrow$  ios
- rightarrow each(limit) {|line| block }  $\rightarrow$  ios
- $\odot$  each(sep,limit) {|line| block }  $\rightarrow$  ios
- $\otimes$  each(...)  $\rightarrow$  an\_enumerator
- each\_line(sep=\$/) {|line| block } → ios
- $\otimes$  each\_line(limit) {|line| block }  $\rightarrow$  ios
- $\otimes$  each\_line(sep,limit) {|line| block }  $\rightarrow$  ios

```
each_line(...) \rightarrow an_enumerator
```

Executes the block for every line in *ios*, where lines are separated by *sep*. *ios* must be opened for reading or an IDError will be raised.

If no block is given, an enumerator is returned instead.

```
f = File.new("testfile")
f.each {|line| puts "#{f.lineno}: #{line}" }
```

produces:

```
1: This is line one
2: This is line two
3: This is line three
4: And so on...
```

### $ⓐ each_byte {|byte| block } → ios$ $<math> ⓐ each_byte → an_enumerator$

Calls the given block once for each byte (0..255) in *ios*, passing the byte as an argument. The stream must be opened for reading or an IOError will be raised.

If no block is given, an enumerator is returned instead.

```
f = File.new("testfile")
checksum = 0
```



# $● each_char {|c| block } → ios$ $<math> ● each_char → an_enumerator$

Calls the given block once for each character in *ios*, passing the character as an argument. The stream must be opened for reading or an IOError will be raised.

If no block is given, an enumerator is returned instead.



# $\otimes$ each\_codepoint {|c| block } $\rightarrow$ ios

- $\odot$  codepoints {|c| block }  $\rightarrow$  ios
- $\otimes$  each\_codepoint  $\rightarrow$  an\_enumerator

### codepoints → an\_enumerator

Passes the Integer ordinal of each character in *ios*, passing the codepoint as an argument. The stream must be opened for reading or an IOError will be raised.

If no block is given, an enumerator is returned instead.

# $\otimes$ each(sep=\$/) {|line| block } $\rightarrow$ ios

- each(limit) {|line| block } → ios
- $\otimes$  each(sep,limit) {|line| block }  $\rightarrow$  ios
- $\otimes$  each(...)  $\rightarrow$  an\_enumerator
- $\otimes$  each\_line(sep=\$/) {|line| block }  $\rightarrow$  ios

# $omegach_line(limit) {|line| block } → ios$ $<math>omegach_line(sep,limit) {|line| block } → ios$ $<math>omegach_line(...) → an_enumerator$

Executes the block for every line in *ios*, where lines are separated by *sep*. *ios* must be opened for reading or an IDError will be raised.

If no block is given, an enumerator is returned instead.

f = File.new("testfile")
f.each {|line| puts "#{f.lineno}: \_#{line}" }

produces:

```
1: This is line one
2: This is line two
3: This is line three
4: And so on...
```

### $_{\odot}$ eof → true or false $_{\odot}$ eof? → true or false

Returns true if *ios* is at end of file that means there are no more data to read. The stream must be opened for reading or an IOError will be raised.

```
f = File.new("testfile")
dummy = f.readlines
f.eof #=> true
```

If *ios* is a stream such as pipe or socket, 10#eof? blocks until the other end sends some data or closes it.

```
r, w = I0.pipe
Thread.new { sleep 1; w.close }
r.eof? #=> true after 1 second blocking
r, w = I0.pipe
Thread.new { sleep 1; w.puts "a" }
```



Note that IO#eof? reads data to the input byte buffer. So IO#sysread may not behave as you intend with IO#eof?, unless you call IO#rewind first (which is not available for some streams).

#### rightarrow eof → true or false rightarrow eof? → true or false

Returns true if *ios* is at end of file that means there are no more data to read. The stream must be opened for reading or an IDError will be raised.

```
f = File.new("testfile")
dummy = f.readlines
f.eof #=> true
```

If *ios* is a stream such as pipe or socket, 10#eof? blocks until the other end sends some data or closes it.

```
r, w = I0.pipe
Thread.new { sleep 1; w.close }
r.eof? #=> true after 1 second blocking
r, w = I0.pipe
Thread.new { sleep 1; w.puts "a" }
r.eof? #=> false after 1 second blocking
r, w = I0.pipe
r.eof? # blocks forever
```

Note that IO#eof? reads data to the input byte buffer. So IO#sysread may not behave as you intend with IO#eof?, unless you call IO#rewind first (which is not available for some streams).

#### ⊚ external\_encoding → encoding

Returns the Encoding object that represents the encoding of the file. If io is write mode and no encoding is specified, returns nil.

### $_{\odot}$ fcntl(integer\_cmd, arg) $\rightarrow$ integer

Provides a mechanism for issuing low-level commands to control or query file-oriented I/O streams. Arguments and results are platform dependent. If *arg* is a number, its value is passed directly. If it is a string, it is interpreted as a binary sequence of bytes (Array#pack might be a useful way to build this string). On Unix platforms, see fcnt1(2) for details. Not implemented on all platforms.

#### $_{\odot}$ fdatasync → 0 or nil

Immediately writes all buffered data in ios to disk.

If the underlying operating system does not support *fdatasync(2)*, IO#fsync is called instead (which might raise a NotImplementedError).

# illeno → fixnum

#### oto\_i → fixnum

Returns an integer representing the numeric file descriptor for *ios*.

\$stdin.fileno #=>
\$stdout.fileno #=>

Also aliased as: to\_i

# Is flush → ios

Flushes any buffered data within ios to the underlying

operating system (note that this is Ruby internal buffering only; the OS may buffer the data as well).

```
$stdout.print "no newline"
$stdout.flush
```

produces:

no newline

#### $_{\odot}$ fsync $\rightarrow$ 0 or nil

Immediately writes all buffered data in *ios* to disk. Note that fsync differs from using IO#sync=. The latter ensures that data is flushed from Ruby's buffers, but does not guarantee that the underlying operating system actually writes it to disk.

NotImplementedError is raised if the underlying operating system does not support *fsync(2)*.

#### getbyte → fixnum or nil

Gets the next 8-bit byte (0..255) from *ios*. Returns nil if called at end of file.

```
f = File.new("testfile")
f.getbyte #=> 84
f.getbyte #=> 104
```

### getc → string or nil

Reads a one-character string from *ios*. Returns nil if called at end of file.

```
f = File.new("testfile")
f.getc #=> "h"
f.getc #=> "e"
```

#### gets(sep=\$/) → string or nil

#### gets(limit) → string or nil

#### gets(sep, limit) → string or nil

Reads the next "line" from the I/O stream; lines are separated by *sep*. A separator of nil reads the entire contents, and a zero-length separator reads the input a paragraph at a time (two successive newlines in the input separate paragraphs). The stream must be opened for reading or an IDError will be raised. The line read in will be returned and also assigned to \$\_. Returns nil if called at end of file. If the first argument is an integer, or optional second argument is given, the returning string would not be longer than the given value in bytes.



### inspect → string

Return a string describing this **IO** object.

#### internal\_encoding → encoding

Returns the <u>Encoding</u> of the internal string if conversion is specified. Otherwise returns nil.

# $_{\odot}$ ioctl(integer\_cmd, arg) $\rightarrow$ integer

Provides a mechanism for issuing low-level commands to control or query I/O devices. Arguments and results are platform dependent. If *arg* is a number, its value is passed directly. If it is a string, it is interpreted as a binary sequence of bytes. On Unix platforms, see ioct1(2) for details. Not implemented on all platforms.

# rightarrow isatty → true or false rightarrow tty? → true or false

Returns true if *ios* is associated with a terminal device (tty), false otherwise.

```
File.new("testfile").isatty #=> false
File.new("/dev/tty").isatty #=> true
```

### o lineno → integer

Returns the current line number in *ios*. The stream must be opened for reading. lineno counts the number of times <u>gets</u> is called rather than the number of newlines encountered. The two values will differ if <u>gets</u> is called with a separator other than newline.

Methods that use \$/ like <u>each</u>, <u>lines</u> and <u>readline</u> will also increment lineno.

See also the \$. variable.

```
f = File.new("testfile")
f.lineno #=> 0
f.gets #=> "This is line one\n"
f.lineno #=> 1
f.gets #=> "This is line two\n"
f.lineno #=> 2
```

# lineno = integer → integer

Manually sets the current line number to the given value. \$. is updated only on the next read.



f.lineno	#=> 1000	
\$.		
f.gets		
\$.		
<b>_</b> ▲		Þ

# lines(\*args)

This is a deprecated alias for each\_line.

# is pid → fixnum

Returns the process ID of a child process associated with *ios*. This will be set by IO.popen.

```
pipe = IO.popen("-")
if pipe
  $stderr.puts "In parent, child pid is #{pipe.pi
else
  $stderr.puts "In child, pid is #{$$}"
end
```

produces:

```
In child, pid is 26209
In parent, child pid is 26209
```

#### 

#### tell → integer

Returns the current offset (in bytes) of ios.



### opos = integer → integer

Seeks to the given position (in bytes) in ios. It is not

guaranteed that seeking to the right position when *ios* is textmode.

```
f = File.new("testfile")
f.pos = 17
f.gets #=> "This is line two\n"
```

# 

Writes the given object(s) to *ios*. The stream must be opened for writing. If the output field separator (\$,) is not nil, it will be inserted between each object. If the output record separator (\$\</code>) is not <code>nil, it will be appended to the output. If no arguments are given, prints \$\_. Objects that aren't strings will be converted by calling their to\_s method. With no argument, prints the contents of the variable \$\_. Returns nil.

```
$stdout.print("This is ", 100, " percent.\n")
```

produces:

```
This is 100 percent.
```

# $_{\odot}$ printf(format\_string [, obj, ...]) → nil

Formats and writes to *ios*, converting parameters under control of the format string. See Kernel#sprintf for details.

# obj → obj → obj

If *obj* is Numeric, write the character whose code is the least-significant byte of *obj*, otherwise write the first byte of the string representation of *obj* to *ios*.

Note: This method is not safe for use with multi-byte characters as it will truncate them.



# oputs(obj, ...) → nil

Writes the given objects to *ios* as with IO#print. Writes a record separator (typically a newline) after any that do not already end with a newline sequence. If called with an array argument, writes each element on a new line. If called without arguments, outputs a single record separator.

```
$stdout.puts("this", "is", "a", "test")
```

produces:



# read([length [, outbuf]]) $\rightarrow$ string, outbuf, or $\otimes$ nil

Reads *length* bytes from the I/O stream.

*length* must be a non-negative integer or nil.

If *length* is a positive integer, it tries to read *length* bytes without any conversion (binary mode). It returns nil or a string whose length is 1 to *length* bytes. nil means it met EOF at beginning. The 1 to *length*-1 bytes string means it met EOF after reading

the result. The *length* bytes string means it doesn't meet EOF. The resulted string is always ASCII-8BIT encoding.

If *length* is omitted or is nil, it reads until EOF and the encoding conversion is applied. It returns a string even if EOF is met at beginning.

If length is zero, it returns "".

If the optional *outbuf* argument is present, it must reference a <u>String</u>, which will receive the data. The *outbuf* will contain only the received data after the method call even if it is not empty at the beginning.

At end of file, it returns nil or "" depend on *length*. *ios*.read() and *ios*.read(nil) returns "". *ios*.read(*positive-integer*) returns nil.

```
f = File.new("testfile")
f.read(16) #=> "This is line one"
# reads whole file
open("file") {|f|
  data = f.read # This returns a string even if 1
....
}
# iterate over fixed length records.
open("fixed-record-file") {|f|
  while record = f.read(256)
    ....
end
}
# iterate over variable length records.
# record is prefixed by 32-bit length.
open("variable-record-file") {|f|
  while len = f.read(4)
    len = len.unpack("N")[0] # 32-bit length
    record = f.read(len) # This returns a string
end
}
```

Note that this method behaves like fread() function in C. This means it retry to invoke read(2) system call to read data with the specified length (or until EOF). This behavior is preserved even if *ios* is non-blocking mode. (This method is non-blocking flag insensitive as other methods.) If you need the behavior like single read(2) system call, consider readpartial, **#read\_nonblock** and sysread.

# read\_nonblock(maxlen) → string read\_nonblock(maxlen, outbuf) → outbuf Reads at most maxlen bytes from ios using the

read(2) system call after O\_NONBLOCK is set for the underlying file descriptor.

If the optional *outbuf* argument is present, it must reference a <u>String</u>, which will receive the data. The *outbuf* will contain only the received data after the method call even if it is not empty at the beginning.

#read\_nonblock just calls the read(2) system call. It causes all errors the read(2) system call causes: Errno::EWOULDBLOCK, Errno::EINTR, etc. The caller should care such errors.

If the exception is Errno::EWOULDBLOCK or Errno::AGAIN, it is extended by <u>IO::WaitReadable</u>. So <u>IO::WaitReadable</u> can be used to rescue the exceptions for retrying read\_nonblock.

#read\_nonblock causes EOFError on EOF.

If the read byte buffer is not empty, <u>#read\_nonblock</u> reads from the buffer like readpartial. In this case, the read(2) system call is not called.

When <u>#read\_nonblock</u> raises an exception kind of <u>IO::WaitReadable</u>, <u>#read\_nonblock</u> should not be called until io is readable for avoiding busy loop. This can be done as follows.

```
# emulates blocking read (readpartial).
begin
   result = io.read_nonblock(maxlen)
rescue I0::WaitReadable
   I0.select([io])
   retry
end
```

Although <u>#read\_nonblock</u> doesn't raise IO::WaitWritable. OpenSSL::Buffering#read\_nonblock can raise IO::WaitWritable. If IO and SSL should be used polymorphically, IO::WaitWritable should be rescued too. See the document of OpenSSL::Buffering#read\_nonblock for sample code.

Note that this method is identical to readpartial except the non-blocking flag is set.

# oreadbyte → fixnum

Reads a byte as with IO#getbyte, but raises an EOFError on end of file.

### $\odot$ readchar $\rightarrow$ string

Reads a one-character string from *ios*. Raises an EOFError on end of file.



 $_{\odot}$  readline(sep=\$/)  $\rightarrow$  string

 $_{\odot}$  readline(limit)  $\rightarrow$  string

# $\odot$ readline(sep, limit) $\rightarrow$ string

Reads a line as with IO#gets, but raises an EOFError on end of file.

- $_{\odot}$  readlines(sep=\$/)  $\rightarrow$  array
- $\odot$  readlines(limit)  $\rightarrow$  array
- $\otimes$  readlines(sep, limit)  $\rightarrow$  array

Reads all of the lines in *ios*, and returns them in *an<u>Array</u>*. Lines are separated by the optional *sep*. If *sep* is nil, the rest of the stream is returned as a single record. If the first argument is an integer, or optional second argument is given, the returning string would not be longer than the given value in bytes. The stream must be opened for reading or an IOError will be raised.

f = File.new("testfile")
f.readlines[0] #=> "This is line one\n"

# $_{\odot}$ readpartial(maxlen) $\rightarrow$ string

### ◎ readpartial(maxlen, outbuf) → outbuf

Reads at most *maxlen* bytes from the I/O stream. It blocks only if *ios* has no data immediately available. It doesn't block if some data available. If the optional *outbuf* argument is present, it must reference a <u>String</u>, which will receive the data. The *outbuf* will contain only the received data after the method call even if it is not empty at the beginning. It raises EOFError on end of file.

readpartial is designed for streams such as pipe, socket, tty, etc. It blocks only when no data immediately available. This means that it blocks only when following all conditions hold.

- byte buffer in the IO object is empty.
- left the content of the stream is empty.
- be the stream is not reached to EOF.

When readpartial blocks, it waits data or EOF on the stream. If some data is reached, readpartial returns with the data. If EOF is reached, readpartial raises EOFError.

When readpartial doesn't blocks, it returns or raises immediately. If the byte buffer is not empty, it returns the data in the buffer. Otherwise if the stream has some content, it returns the data in the stream. Otherwise if the stream is reached to EOF, it raises EOFError.



Note that readpartial behaves similar to sysread. The differences are:

- If the byte buffer is not empty, read from the byte buffer instead of "sysread for buffered <u>IO</u> (<u>IOError</u>)".
- It doesn't cause Errno::EWOULDBLOCK and Errno::EINTR. When readpartial meets EWOULDBLOCK and EINTR by read system call, readpartial retry the system call.

The later means that readpartial is nonblocking-flag

insensitive. It blocks on the situation <u>#sysread</u> causes Errno::EWOULDBLOCK as if the fd is blocking mode.

# $reopen(other_IO) → ios$ $reopen(path, mode_str) → ios$

Reassociates *ios* with the I/O stream given in *other\_IO* or to a new stream opened on *path*. This may dynamically change the actual class of this stream.



#### $\odot$ rewind $\rightarrow$ 0

Positions *ios* to the beginning of input, resetting lineno to zero.



Note that it cannot be used with streams such as pipes, ttys, and sockets.

### $_{\odot}$ seek(amount, whence=IO::SEEK\_SET) → 0

Seeks to a given offset *an<u>Integer</u>* in the stream according to the value of *whence*:

:CUR or IO::SEEK\_CUR | Seeks to \_amount\_ plus cu :END or IO::SEEK\_END | Seeks to \_amount\_ plus er | probably want a negative



- $\otimes$  set\_encoding("ext\_enc:int\_enc")  $\rightarrow$  io
- $\otimes$  set\_encoding(ext\_enc, int\_enc)  $\rightarrow$  io
- $\otimes$  set\_encoding("ext\_enc:int\_enc", opt)  $\rightarrow$  io
- Set\_encoding(ext\_enc, int\_enc, opt) → io If single argument is specified, read string from io is tagged with the encoding specified. If encoding is a colon separated two encoding names "A:B", the read string is converted from encoding A (external encoding) to encoding B (internal encoding), then tagged with B. If two arguments are specified, those must be encoding objects or encoding names, and the first one is the external encoding, and the second one is the internal encoding. If the external encoding and the internal encoding is specified, optional hash argument specify the conversion option.

#### ⊚ stat → stat

Returns status information for *ios* as an object of type File::Stat.



#### sync → true or false

Returns the current "sync mode" of *ios*. When sync mode is true, all output is immediately flushed to the underlying operating system and is not buffered by Ruby internally. See also IO#fsync.

```
f = File.new("testfile")
f.sync #=> false
```

### sync = boolean → boolean

Sets the "sync mode" to true or false. When sync mode is true, all output is immediately flushed to the underlying operating system and is not buffered internally. Returns the new state. See also IO#fsync.

```
f = File.new("testfile")
f.sync = true
```

(produces no output)

# $_{\odot}$ sysread(maxlen[, outbuf]) $\rightarrow$ string

Reads *maxlen* bytes from *ios* using a low-level read and returns them as a string. Do not mix with other methods that read from *ios* or you may get unpredictable results. If the optional *outbuf* argument is present, it must reference a <u>String</u>, which will receive the data. The *outbuf* will contain only the received data after the method call even if it is not empty at the beginning. Raises SystemCallError On error and EOFError at end of file.

```
f = File.new("testfile")
f.sysread(16) #=> "This is line one"
```

# sysseek(offset, whence=IO::SEEK\_SET) $\rightarrow$ $_{\textcircled{}}$ integer

Seeks to a given *offset* in the stream according to the value of *whence* (see IO#seek for values of *whence*). Returns the new offset into the file.



# syswrite(string) → integer

Writes the given string to *ios* using a low-level write. Returns the number of bytes written. Do not mix with other methods that write to *ios* or you may get unpredictable results. Raises systemCallError on error.



to\_io → ios Returns ios.

#### rightarrow isatty → true or false rightarrow true or false

Returns true if *ios* is associated with a terminal device (tty), false otherwise.



# omega ungetbyte(string) → nil omega ungetbyte(integer) → nil

Pushes back bytes (passed as a parameter) onto *ios*, such that a subsequent buffered read will return it. Only one byte may be pushed back before a subsequent read operation (that is, you will be able to read only the last of several bytes that have been pushed back). Has no effect with unbuffered reads (such as IO#sysread).



# ongetc(string) → nil

Pushes back one character (passed as a parameter) onto *ios*, such that a subsequent buffered character read will return it. Only one character may be pushed back before a subsequent read operation (that is, you will be able to read only the last of several characters that have been pushed back). Has no effect with unbuffered reads (such as IO#sysread).



```
f.getc
```

#### owrite(string) → integer

Writes the given string to *ios*. The stream must be opened for writing. If the argument is not a string, it will be converted to a string using to\_s. Returns the number of bytes written.

```
count = $stdout.write("This is a test\n")
puts "That was #{count} bytes of data"
```

produces:

```
This is a test
That was 15 bytes of data
```

# write\_nonblock(string) → integer write\_nonblock(string [, options]) → integer

Writes the given string to *ios* using the write(2) system call after O\_NONBLOCK is set for the underlying file descriptor.

It returns the number of bytes written.

<u>#write\_nonblock</u> just calls the write(2) system call. It causes all errors the write(2) system call causes: Errno::EWOULDBLOCK, Errno::EINTR, etc. The result may also be smaller than string.length (partial write). The caller should care such errors and partial write.

If the exception is Errno::EWOULDBLOCK or Errno::AGAIN, it is extended by <u>IO::WaitWritable</u>. So <u>IO::WaitWritable</u> can be used to rescue the exceptions for retrying write\_nonblock.

```
# Creates a pipe.
r, w = I0.pipe
```



If the write buffer is not empty, it is flushed at first.

When <u>#write\_nonblock</u> raises an exception kind of <u>IO::WaitWritable</u>, <u>#write\_nonblock</u> should not be called until io is writable for avoiding busy loop. This can be done as follows.

```
begin
  result = io.write_nonblock(string)
rescue I0::WaitWritable, Errno::EINTR
  I0.select(nil, [io])
  retry
end
```

Note that this doesn't guarantee to write all data in string. The length written is reported as result and it should be checked later.

On some platforms such as Windows, <u>#write\_nonblock</u> is not supported according to the kind of the <u>IO</u> object. In such cases, <u>#write\_nonblock</u> raises Errno::EBADF.

By specifying `exception: false`, the options hash allows you to indicate that <u>#write\_nonblock</u> should not raise an <u>IO::WaitWritable</u> exception, but return the symbol :wait\_writable instead.

# class IO::EAGAINWaitReadable

**In Files** 

lile.c

Parent

rb\_eEAGAIN

# **Included Modules**

IO::WaitReadable

# class IO::EAGAINWaitWritable

**In Files** 

lile.c

Parent

rb\_eEAGAIN

# **Included Modules**

IO::WaitWritable

# class IO::EINPROGRESSWaitRead

**In Files** 

📄 file.c

Parent

rb\_eEINPROGRESS

# **Included Modules**

IO::WaitReadable

# class IO::EINPROGRESSWaitWrita

**In Files** 

📄 file.c

Parent

rb\_eEINPROGRESS

# **Included Modules**

IO::WaitWritable

# class IO::EWOULDBLOCKWaitRea

**In Files** 

📄 file.c

Parent

rb\_eEWOULDBLOCK

# **Included Modules**

IO::WaitReadable

# class IO::EWOULDBLOCKWaitWri

**In Files** 

📄 file.c

Parent

rb\_eEWOULDBLOCK

# **Included Modules**

IO::WaitWritable
# module IO::WaitReadable

# **In Files**

📄 file.c

# module IO::WaitWritable

# **In Files**

📄 file.c

# class IOError

Raised when an <u>IO</u> operation fails.



Note that some <u>IO</u> failures raise +SystemCallError+s and these are not subclasses of IOError:



In Files

📄 io.c

# Parent

StandardError

# class IndexError

Raised when the given index is invalid.

a = [:foo, a.fetch(0) a[4] a.fetch(4)	:bar] #=> #=> #=>		
_ <b>_</b>			•

# In Files

error.c

# Parent

StandardError

# class Integer

This class is the basis for the two concrete classes that hold whole numbers, <u>Bignum</u> and <u>Fixnum</u>.

In Files	
numeric.c	

rational.c

# Parent

Numeric

# **Public Instance Methods**

# $\odot$ to\_i $\rightarrow$ integer

As int is already an <u>Integer</u>, all these methods simply return the receiver.

Synonyms are to\_int, floor, ceil, truncate.

# ohr([encoding]) → string

Returns a string containing the character represented by the int's value according to encoding.



#### $\odot$ denominator $\rightarrow$ 1

Returns 1.

# rightarrow downto(limit) {|i| block } $\rightarrow$ self rightarrow downto(limit) $\rightarrow$ an\_enumerator

Iterates the given block, passing decreasing values from int down to and including limit.

If no block is given, an Enumerator is returned instead.



#### rightarrow even? $\rightarrow$ true or false

Returns true if int is an even number.

#### osto\_i → integer

As int is already an <u>Integer</u>, all these methods simply return the receiver.

Synonyms are to\_int, floor, ceil, truncate.

## Gcd(int2) → integer integer

Returns the greatest common divisor (always positive). 0.gcd(x) and x.gcd(0) return abs(x).

#### ogcdlcm(int2) → array

Returns an array; [int.gcd(int2), int.lcm(int2)].



# integer? → true

Since int is already an <u>Integer</u>, this always returns true.

# o lcm(int2) → integer

Returns the least common multiple (always positive). 0.lcm(x) and x.lcm(0) return zero.



# onext → integer

#### Succ → integer

Returns the Integer equal to int + 1, same as #next.

# $\odot$ numerator $\rightarrow$ self

Returns self.

## odd? → true or false

Returns true if int is an odd number.

 $\odot$  ord  $\rightarrow$  self

Returns the int itself.

This method is intended for compatibility to character constant in Ruby 1.9.

For example, ?a.ord returns 97 both in 1.8 and 1.9.

# opred → integer

Returns the Integer equal to int - 1.

1.pred #=> 0 (-1).pred #=> -2

# $_{\odot}$ rationalize([eps]) $\rightarrow$ rational

Returns the value as a rational. The optional argument eps is always ignored.

# $rightarrow round([ndigits]) \rightarrow integer or float$

Rounds int to a given precision in decimal digits (default 0 digits).

Precision may be negative. Returns a floating point number when ndigits is positive, self for zero, and round down for negative.



times {|i| block } → self
 times → an\_enumerator
 Iterates the given block int times, passing in values from zero to int - 1.
 If no block is given, an Enumerator is returned

instead.

(-1).next



# oto\_i → integer

As int is already an <u>Integer</u>, all these methods simply return the receiver.

Synonyms are to\_int, floor, ceil, truncate.

## oto\_i → integer

As int is already an <u>Integer</u>, all these methods simply return the receiver.

Synonyms are to\_int, floor, ceil, truncate.

# oto\_r → rational

Returns the value as a rational.



# oto\_i → integer

As int is already an Integer, all these methods simply

return the receiver.

Synonyms are to\_int, floor, ceil, truncate.

# rightarrow upto(limit) {|i| block } → self rightarrow upto(limit) → an\_enumerator

Iterates the given block, passing in integer values from int up to and including limit.

If no block is given, an Enumerator is returned instead.

For example:



# **class Interrupt**

Raised with the interrupt signal is received, typically because the user pressed on Control-C (on most posix platforms). As such, it is a subclass of SignalException.



produces:



then waits until it is interrupted with Control-C and then prints:





Parent

SignalException

# **module Kernel**

The <u>Kernel</u> module is included by class <u>Object</u>, so its methods are available in every Ruby object.

The Kernel instance methods are documented in class Object while the module methods are documented here. These methods are called without a receiver and thus can be called in functional form:

```
sprintf "%.1f", 1.234 #
In Files
complex.c
cont.c
error.c
eval.c
📄 eval jump.c
📄 file.c
io.c
load.c
object.c
proc.c
process.c
random.c
rational.c
l ruby.c
📄 signal.c
vm backtrace.c
```

vm\_eval.c vm\_trace.c

# **Public Instance Methods**

### 

Returns arg as an Array.

First tries to call to\_ary on arg, then to\_a.

# omega om

Returns x+i\*y;

Complex(1, 2)	
<pre>Complex('1+2i')</pre>	
Complex(nil)	
Complex(1, nil)	

Syntax of string form:

See <u>String#to\_c</u>.

# Solution → Float → Float

Returns *arg* converted to a float. <u>Numeric</u> types are converted directly, the rest are converted using *arg*.to\_f. Converting nil generates a TypeError.



# ⊚Hash(arg) → hash

Converts arg to a Hash by calling arg.to\_hash. Returns an empty Hash when arg is nil or [].



# $_{\odot}$ Integer(arg, base=0) $\rightarrow$ integer

Converts *arg* to a Fixnum Or Bignum. <u>Numeric</u> types are converted directly (with floating point numbers being truncated). *base* (0, or between 2 and 36) is a base for integer string representation. If *arg* is a String, when *base* is omitted or equals zero, radix indicators (0, 0b, and 0x) are honored. In any case, strings should be strictly conformed to numeric representation. This behavior is different from that of String#to\_i. Non string values will be converted by first trying to\_int, then to\_i. Passing nil raises a **TypeError**.

```
Integer(123.999) #=> 123
Integer("0x1a") #=> 26
Integer(Time.new) #=> 1204973019
Integer("0930", 10) #=> 930
```

Integer("111", 2) #=> 7 Integer(nil) #=> TypeError

# $_{\odot}$ Rational(x[, y]) → numeric

Returns x/y;

Rational(1, 2)	
Rational('1/2')	
Rational(nil)	
Rational(1, nil)	

Syntax of string form:

```
string form = extra spaces , rational , extra spa
rational = [ sign ] , unsigned rational ;
unsigned rational = numerator | numerator , "/" ,
numerator = integer part | fractional part | inte
denominator = digits ;
integer part = digits ;
fractional part = "." , digits , [ ( "e" | "E" )
sign = "-" | "+" ;
digits = digit , { digit | "_" , digit } ;
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6"
extra spaces = ? \s* ? ;
```

See <u>String#to\_r</u>.

# String(arg) → string

Returns arg as a string.

First tries to call its to\_str method, then its to\_s method.

```
      String(self)
      #=> "main"

      String(self.class)
      #=> "Object"

      String(123456)
      #=> "123456"
```

## omega\_callee\_\_ → symbol

Returns the called name of the current method as a

Symbol. If called outside of a method, it returns nil.

# omega\_dir\_ → string

Returns the canonicalized absolute path of the directory of the file from which this method is called. It means symlinks in the path is resolved. If \_\_FILE\_\_ is nil, it returns nil. The return value equals to File.dirname(File.realpath(\_\_FILE\_\_)).

# method → symbol

Returns the name at the definition of the current method as a <u>Symbol</u>. If called outside of a method, it returns nil.

# (a) `cmd` → string

Returns the standard output of running *cmd* in a subshell. The built-in syntax %x{...} uses this method. Sets \$? to the process status.



# 🚳 abort

- Kernel::abort([msg])
- Process::abort([msg])

Terminate execution immediately, effectively by calling Kernel.exit(false). If *msg* is given, it is written to STDERR prior to terminating.

# at\_exit { block } → proc

Converts *block* to a proc object (and therefore binds it at the point of call) and registers it for execution when the program exits. If multiple handlers are registered, they are executed in reverse order of registration.

```
def do_at_exit(str1)
   at_exit { print str1 }
end
at_exit { puts "cruel world" }
do_at_exit("goodbye ")
exit
```

produces:

goodbye cruel world

# $_{\odot}$ autoload(module, filename) $\rightarrow$ nil

Registers *filename* to be loaded (using Kernel::require) the first time that *module* (which may be a string or a symbol) is accessed.



## $_{\odot}$ autoload?(name) $\rightarrow$ String or nil

Returns *filename* to be loaded if *name* is registered as autoload.



## $_{\odot}$ binding $\rightarrow$ a\_binding

Returns a Binding object, describing the variable and method bindings at the point of call. This object can be used when calling eval to execute the evaluated command in this environment. See also the description of class Binding.



# rightarrow block\_given? → true or false rightarrow iterator? → true or false

Returns true if yield would execute a block in the current context. The iterator? form is mildly deprecated.

<pre>def try    if block_given?     vield</pre>	
"no block"	
end	
end	
try	
<pre>try { "hello" }</pre>	
try do "hello" end	

# $\odot$ callcc {|cont| block } $\rightarrow$ obj

Generates a <u>Continuation</u> object, which it passes to the associated block. You need to require 'continuation' before using this method. Performing a *cont*.call will cause the <u>callcc</u> to return (as will falling through the end of the block). The value returned by the <u>callcc</u> is the value of the block, or the value passed to *cont*.call. See class <u>Continuation</u> for more details. Also see <u>#throw</u> for an alternative mechanism for unwinding a call stack.

# $_{\odot}$ caller(start=1, length=nil) → array or nil $_{\odot}$ caller(range) → array or nil

Returns the current execution stack—an array containing strings in the form file:line Or file:line: in `method'.

The optional *start* parameter determines the number of initial stack entries to omit from the top of the stack.

A second optional length parameter can be used to limit how many entries are returned from the stack.

Returns nil if *start* is greater than the size of current execution stack.

Optionally you can pass a range, which will return an array containing the entries within the specified range.

```
def a(skip)
  caller(skip)
end
def b(skip)
  a(skip)
end
def c(skip)
  b(skip)
end
C(0)
C(1)
c(2)
C(3)
c(4)
C(5)
4
```

caller\_locations(start=1, length=nil)  $\rightarrow$  array  $_{\textcircled{}}$  or nil

 $_{\odot}$  caller\_locations(range)  $\rightarrow$  array or nil

Returns the current execution stack—an array

containing backtrace location objects.

See Thread::Backtrace::Location for more information.

The optional *start* parameter determines the number of initial stack entries to omit from the top of the stack.

A second optional length parameter can be used to limit how many entries are returned from the stack.

Returns nil if *start* is greater than the size of current execution stack.

Optionally you can pass a range, which will return an array containing the entries within the specified range.

# $_{\odot}$ catch([tag]) {|tag| block } $\rightarrow$ obj

catch executes its block. If throw is not called, the block executes normally, and catch returns the value of the last expression evaluated.

```
catch(1) { 123 } # => 123
```

If +throw(tag2, val)+ is called, Ruby searches up its stack for a catch block whose tag has the same object\_id as *tag2*. When found, the block stops executing and returns *val* (or nil if no second argument was given to throw).

```
catch(1) { throw(1, 456) } # => 456
catch(1) { throw(1) } # => nil
```

When tag is passed as the first argument, catch yields it as the parameter of the block.



When no tag is given, catch yields a new unique object (as from object.new) as the block parameter. This object can then be used as the argument to throw, and will match the correct catch block.

```
catch do |obj_A|
  catch do |obj_B|
    throw(obj_B, 123)
    puts "This puts is not reached"
  end
  puts "This puts is displayed"
  456
end
catch do obj_A
  catch do |obj_B|
    throw(obj_A, 123)
    puts "This puts is still not reached"
  end
  puts "Now this puts is also not reached"
  456
end
```

### rightarrow chomp → \$\_ rightarrow chomp(string) → \$\_

Equivalent to  $= \$ .chomp(*string*). See String#chomp. Available only when -p/-n command line option specified.

# $\odot$ chop $\rightarrow$ \$\_

Equivalent to (\$\_.dup).chop!, except nil is never returned. See string#chop!. Available only when -p/n command line option specified.

# eval(string [, binding [, filename [,lineno]]]) $\circledast \rightarrow obj$

Evaluates the Ruby expression(s) in *string*. If *binding* is given, which must be a Binding object, the evaluation is performed in its context. If the optional *filename* and *lineno* parameters are present, they will be used when reporting syntax errors.



# exec([env,] command... [,options])

Replaces the current process by running the given external *command*, which can take one of the following forms:

```
exec(commandline)
```

command line string which is passed to the standard shell

```
exec(cmdname, arg1, ...)
```

command name and one or more arguments (no shell)

```
exec([cmdname, argv0], arg1, ...)
```

command name, <u>argv</u> and zero or more arguments (no shell)

In the first form, the string is taken as a command line that is subject to shell expansion before being executed. The standard shell always means "/bin/sh" on Unixlike systems, same as ENV["RUBYSHELL"] (Or ENV["COMSPEC"] on Windows NT series), and similar.

If the string from the first form (exec("command")) follows these simple rules:

- no meta characters
- local no shell reserved word and no special built-in
- Ruby invokes the command directly without shell

You can force shell invocation by adding ";" to the string (because ";" is a meta character).

Note that this behavior is observable by pid obtained (return value of spawn() and <u>IO#pid</u> for <u>IO.popen</u>) is the pid of the invoked command, not shell.

In the second form (exec("command1", "arg1", ...)), the first is taken as a command name and the rest are passed as parameters to command with no shell expansion.

In the third form (exec(["command", "argv0"], "arg1", ...)), starting a two-element array at the beginning of the command, the first element is the command to be executed, and the second argument is used as the argv[0] value, which may show up in process listings.

In order to execute the command, one of the exec(2) system calls are used, so the running command may inherit some of the environment of the original program (including open file descriptors).

This behavior is modified by the given env and options parameters. See ::spawn for details.

If the command fails to execute (typically

Errno::ENOENT when it was not found) a SystemCallError exception is raised.

This method modifies process attributes according to given options before exec(2) system call. See ::spawn for more details about the given options.

The modified attributes may be retained when exec(2) system call fails.

For example, hard resource limits are not restorable.

Consider to create a child process using ::spawn or <u>#system</u> if this is not acceptable.



# exit(status=true)

## Kernel::exit(status=true)

## Process::exit(status=true)

Initiates the termination of the Ruby script by raising the systemExit exception. This exception may be caught. The optional parameter is used to return a status code to the invoking environment. true and FALSE of *status* means success and failure respectively. The interpretation of other integer values are system dependent.

```
begin
  exit
  puts "never get here"
rescue SystemExit
  puts "rescued a SystemExit exception"
end
puts "after begin block"
```

produces:

```
rescued a SystemExit exception after begin block
```

Just prior to termination, Ruby executes any at\_exit functions (see Kernel::at\_exit) and runs any object finalizers (see ObjectSpace.define\_finalizer).

```
at_exit { puts "at_exit function" }
ObjectSpace.define_finalizer("string", proc { pu
exit
```

produces:

at\_exit function in finalizer

# exit!(status=false)

Exits the process immediately. No exit handlers are run. *status* is returned to the underlying system as the exit status.



#### 🚳 raise

- raise(string)
- raise(exception [, string [, array]])
- 🚳 fail
- fail(string)
- fail(exception [, string [, array]])

With no arguments, raises the exception in \$! or raises a RuntimeError if \$! is nil. With a single string argument, raises a RuntimeError with the string as a message. Otherwise, the first parameter should be the name of an Exception class (or an object that returns an Exception object when sent an exception message). The optional second parameter sets the message associated with the exception, and the third parameter is an array of callback information. Exceptions are caught by the rescue clause of begin...end blocks.

raise "Failed to create socket" raise ArgumentError, "No parameters", caller

# o fork [{ block }] → fixnum or nil o fork [{ block }] → fixnum or nil

Creates a subprocess. If a block is specified, that block is run in the subprocess, and the subprocess terminates with a status of zero. Otherwise, the fork call returns twice, once in the parent, returning the process ID of the child, and once in the child, returning *nil*. The child process can exit using Kernel.exit! to avoid running any at\_exit functions. The parent process should use Process.wait to collect the termination statuses of its children or use Process.detach to register disinterest in their status; otherwise, the operating system may accumulate zombie processes.

The thread calling fork is the only thread in the created child process. fork doesn't copy other threads.

If fork is not usable, Process.respond\_to?(:fork) returns false.

Note that fork(2) is not available on some platforms like Windows and NetBSD 4. Therefore you should use spawn() instead of fork().

# 

# sprintf(format\_string [, arguments...]) $\rightarrow$ $_{\odot}$ string

Returns the string resulting from applying *format\_string* to any additional arguments. Within the format string, any characters other than format sequences are copied to the result.

The syntax of a format sequence is follows.

#### %[flags][width][.precision]type

A format sequence consists of a percent sign, followed by optional flags, width, and precision indicators, then terminated with a field type character. The field type controls how the corresponding sprintf argument is to be interpreted, while the flags modify that interpretation.

The field type characters are:

Field	Integer Format
b     	Convert argument as a binary number. Negative numbers will be displayed as a t prefixed with `1'.
B	Equivalent to `b', but uses an uppercase in the alternative format by #.
d	Convert argument as a decimal number.
i	Identical to `d'.
0     	Convert argument as an octal number. Negative numbers will be displayed as a t prefixed with `7'.
u	Identical to `d'.
X     	Convert argument as a hexadecimal number. Negative numbers will be displayed as a t prefixed with `f' (representing an infi leading 'ff's).
X	Equivalent to `x', but uses uppercase let
Field	Float Format

е	Convert floating point argument into expo
	with one digit before the decimal point a
	The precision specifies the number of dig
	point (defaulting to six).
E	Equivalent to `e', but uses an uppercase
	the exponent.
f	Convert floating point argument as [-]ddd
	where the precision specifies the number
	the decimal point.
g	Convert a floating point number using exp
	if the exponent is less than -4 or greate
	equal to the precision, or in dd.dddd for
	The precision specifies the number of sid
G	Equivalent to `g', but use an uppercase
a	Convert floating point argument as [-]0x
a	which is consisted from optional sign "
	l as hexadecimal "n" and exponential nart
Δ	Equivalent to `a' but use unnercase `X'
Field	l Other Format
	+
с —	Argument is the numeric code for a single
	1 a single character string itself
	The valuing of argument inspect
- р 	Argument is a string to be substituted
	Argument is a string to be substituted.
	sequence contains a precision, at most the
0/	WIII De Copieu.
%	A percent sign itself will be displayed.
4	► International

The flags modifies the behavior of the formats. The flag characters are:

Flag	Applies to	Meaning
space	bBdiouxX   aAeEfgG   (numeric fmt)   	Leave a space at the s non-negative numbers. For `o', `x', `X', `b' a minus sign with abso negative values.
(digit)\$	all	Specifies the absolute for this field. Absol argument numbers canno sprintf string.
#	bBoxX	Use an alternative for

	aAeEfgG	For the conversions `c until the first digit it is not formatted as For the conversions `> on non-zero, prefix th ``0X'', ``0b'' and ``C For `a', `A', `e', `E' force a decimal point even if no digits foll For `g' and 'G', do no
+	bBdiouxX aAeEfgG (numeric fmt)	Add a leading plus sig   numbers.   For `o', `x', `X', `b'   a minus sign with abso   negative values.
-	all	Left-justify the resu
0 (zero)	bBdiouxX aAeEfgG (numeric fmt)	Pad with zeros, not sp For `o', `x', `X', `b' is used for negative r complements.
*	all	Use the next argument If negative, left-just asterisk is followed k sign, use the indicate
•		► I

Examples of flags:

```
# `+' and space flag specifies the sign of non-ne
sprintf("%d", 123) #=> "123"
sprintf("%+d", 123) #=> "+123"
# `#' flag for `o' increases number of digits to
# `+' and space flag changes format of negative r
sprintf("%o", 123) #=> "173"
sprintf("%#o", 123) #=> "0173"
sprintf("%+o", -123) #=> "0173"
sprintf("%o", -123) #=> ".7605"
sprintf("%#o", -123) #=> ".7605"
# `#' flag for `x' add a prefix `0x' for non-zero
# `#' flag for `x' add a prefix `0x' for non-zero
# `+' and space flag disables complements for negative r
```

```
sprintf("%x", 123)
sprintf("%#x", 123)
sprintf("%+x", -123)
sprintf("%x", -123)
sprintf("%#x", -123)
sprintf("%#x", 0)
sprintf("%X", 123)
sprintf("%#X", 123)
sprintf("%b", 123)
sprintf("%#b", 123)
sprintf("%+b", -123)
sprintf("%b", -123)
sprintf("%#b", -123)
sprintf("%#b", 0)
sprintf("%B", 123)
sprintf("%#B", 123)
sprintf("%.0e", 1)
sprintf("%#.0e", 1)
sprintf("%.0f", 1234)
sprintf("%#.0f", 1234)
sprintf("%g", 123.4)
sprintf("%#g", 123.4)
sprintf("%g", 123456)
sprintf("%#g", 123456)
4
```

The field width is an optional integer, followed optionally by a period and a precision. The width specifies the minimum number of characters that will be written to the result for this field.

Examples of width:

<pre># padding is done by s</pre>	
# 0 or radix-1.	
sprintf("%20d", 123)	
sprintf("%+20d", 123)	
sprintf("%020d", 123)	
sprintf("%+020d", 123)	
sprintf("% 020d", 123)	
sprintf("%-20d", 123)	
sprintf("%-+20d", 123)	
sprintf("%- 20d", 123)	
sprintf("%020x", -123)	

For numeric fields, the precision controls the number of decimal places displayed. For string fields, the precision determines the maximum number of characters to be copied from the string. (Thus, the format sequence %10.10s will always contribute exactly ten characters to the result.)

Examples of precisions:

<pre>sprintf("%20.8d",</pre>	123)		
<pre>sprintf("%20.80",</pre>	123)		
<pre>sprintf("%20.8x",</pre>	123)		
<pre>sprintf("%20.8b",</pre>	123)		
<pre>sprintf("%20.8d",</pre>	-123)		
<pre>sprintf("%20.80",</pre>	-123)		
<pre>sprintf("%20.8x",</pre>	-123)		
<pre>sprintf("%20.8b",</pre>	-11)		
<pre>sprintf("%#20.8d",</pre>	123)		
<pre>sprintf("%#20.80",</pre>	123)		
<pre>sprintf("%#20.8x",</pre>	123)		
<pre>sprintf("%#20.8b",</pre>	123)		
<pre>sprintf("%#20.8d",</pre>	-123)		
<pre>sprintf("%#20.80",</pre>	-123)		
<pre>sprintf("%#20.8x",</pre>	-123)		
<pre>sprintf("%#20.8b",</pre>	-11)		



Examples:



For more complex formatting, Ruby supports a reference by name. %<name>s style uses format style, but %{name} style doesn't.

Examples:



gets(sep=\$/) → string or nil
 gets(limit) → string or nil
 gets(sep,limit) → string or nil

Returns (and assigns to \$\_) the next line from the list of files in ARGV (or \$\*), or from standard input if no files are present on the command line. Returns nil at end of file. The optional argument specifies the record separator. The separator is included with the contents of each record. A separator of nil reads the entire contents, and a zero-length separator reads the input one paragraph at a time, where paragraphs are divided by two consecutive newlines. If the first argument is an integer, or optional second argument is given, the returning string would not be longer than the given value in bytes. If multiple filenames are present in ARGV, +gets(nil)+ will read the contents one file at a time.

```
ARGV << "testfile"
print while gets</pre>
```

produces:

```
This is line one
This is line two
This is line three
And so on...
```

The style of programming using \$\_ as an implicit parameter is gradually losing favor in the Ruby community.

#### $\odot$ global\_variables $\rightarrow$ array

Returns an array of the names of global variables.



 $_{\odot}$  gsub(pattern, replacement) → \$\_  $_{\odot}$  gsub(pattern) {|...| block } → \$\_ Equivalent to \$\_.gsub..., except that \$\_ will be updated if substitution occurs. Available only when p/-n command line option specified.

# rightarrow block\_given? → true or false rightarrow iterator? → true or false

Returns true if yield would execute a block in the current context. The iterator? form is mildly deprecated.



# $_{\odot}$ lambda { |...| block } → a\_proc

Equivalent to Proc.new, except the resulting Proc objects check the number of parameters passed when called.

#### $_{\odot}$ load(filename, wrap=false) $\rightarrow$ true

Loads and executes the Ruby program in the file *filename*. If the filename does not resolve to an absolute path, the file is searched for in the library directories listed in \$:. If the optional *wrap* parameter is true, the loaded script will be executed under an anonymous module, protecting the calling program's global namespace. In no circumstance will any local variables in the loaded file be propagated to the loading environment.

# $\odot$ local\_variables $\rightarrow$ array

Returns the names of the current local variables.



# $rightarrow loop { block }$ $rightarrow loop → an_enumerator$

Repeatedly executes the block.

If no block is given, an enumerator is returned instead.

```
loop do
  print "Input: "
  line = gets
  break if !line or line =~ /^qQ/
  # ...
end
```

StopIteration raised in the block breaks the loop.

open(path [, mode [, perm]] [, opt])  $\rightarrow$  io or  $_{\textcircled{}}$  nil

# open(path [, mode [, perm]] [, opt]) {|io| block $\Rightarrow$ obj

Creates an <u>IO</u> object connected to the given stream, file, or subprocess.

If path does not start with a pipe character (|), treat it as the name of a file to open using the specified mode (defaulting to "r").
The mode is either a string or an integer. If it is an integer, it must be bitwise-or of open(2) flags, such as File::RDWR or File::EXCL. If it is a string, it is either "fmode", "fmode:ext\_enc", or "fmode:ext\_enc:int\_enc".

See the documentation of <u>IO.new</u> for full documentation of the mode string directives.

If a file is being created, its initial permissions may be set using the perm parameter. See <u>File.new</u> and the open(2) and chmod(2) man pages for a description of permissions.

If a block is specified, it will be invoked with the <u>IO</u> object as a parameter, and the <u>IO</u> will be automatically closed when the block terminates. The call returns the value of the block.

If path starts with a pipe character ("|"), a subprocess is created, connected to the caller by a pair of pipes. The returned <u>IO</u> object may be used to write to the standard input and read from the standard output of this subprocess.

If the command following the pipe is a single minus sign ("|-"), Ruby forks, and this subprocess is connected to the parent. If the command is not "-", the subprocess runs the command.

When the subprocess is ruby (opened via "|-"), the open call returns nil. If a block is associated with the open call, that block will run twice — once in the parent and once in the child.

The block parameter will be an <u>IO</u> object in the parent and nil in the child. The parent's IO object will be connected to the child's \$stdin and \$stdout. The subprocess will be terminated at the end of the block.

### **Examples**

Reading from "testfile":

```
open("testfile") do |f|
   print f.gets
end
```

Produces:

```
This is line one
```

Open a subprocess and read its output:

```
cmd = open("|date")
print cmd.gets
cmd.close
```

Produces:

Wed Apr 9 08:56:31 CDT 2003

Open a subprocess running the same Ruby program:

```
f = open("|-", "w+")
if f == nil
   puts "in Child"
   exit
else
   puts "Got: #{f.gets}"
end
```

Produces:

Got: in Child

Open a subprocess using a block to receive the <u>IO</u> object:

```
open "|-" do |f|
if f then
    # parent process
    puts "Got: #{f.gets}"
else
    # child process
    puts "in Child"
```



```
    ⇒ p(obj) → obj
    ⇒ p(obj1, obj2, ...) → [obj, ...]
    ⇒ p() → nil
    For each object, directly writes obj inspect
```

For each object, directly writes *obj.inspect* followed by a newline to the program's standard output.

```
S = Struct.new(:name, :state)
s = S['dave', 'TX']
p s
```

produces:

```
#<S name="dave", state="TX">
```

# ◎ print(obj, ...) → nil

Prints each object in turn to \$stdout. If the output field separator (\$,) is not nil, its contents will appear between each field. If the output record separator (\$\ </code>) is not nil, it will be appended to the output. If no arguments are given, prints <code>\$\_. Objects that aren't strings will be converted by calling their to\_s method.

```
print "cat", [1,2,3], 99, "\n"
$, = ", "
$\ = "\n"
print "cat", [1,2,3], 99
```

produces:

cat12399

cat, 1, 2, 3, 99

```
    printf(io, string [, obj ... ]) → nil
    printf(string [, obj ... ]) → nil
```

Equivalent to:

io.write(sprintf(string, obj, ...))

or

\$stdout.write(sprintf(string, obj, ...))

### proc { |...| block } → a\_proc Equivalent to Proc.new.

# ◎ putc(int) → int

Equivalent to:

\$stdout.putc(int)

Refer to the documentation for <u>IO#putc</u> for important information regarding multi-byte characters.

# puts(obj, ...) → nil

Equivalent to

\$stdout.puts(obj, ...)

### 🚳 raise

- raise(string)
- raise(exception [, string [, array]])
- 🚳 fail
- 🚳 fail(string)

# fail(exception [, string [, array]])

With no arguments, raises the exception in \$! or raises a RuntimeError if \$! is nil. With a single string argument, raises a RuntimeError with the string as a message. Otherwise, the first parameter should be the name of an Exception class (or an object that returns an Exception object when sent an exception message). The optional second parameter sets the message associated with the exception, and the third parameter is an array of callback information. Exceptions are caught by the rescue clause of begin...end blocks.

raise "Failed to create socket"
raise ArgumentError, "No parameters", caller

## $_{\odot}$ rand(max=0) $\rightarrow$ number

If called without an argument, or if max.to\_i.abs == 0, rand returns a pseudo-random floating point number between 0.0 and 1.0, including 0.0 and excluding 1.0.

rand #=> 0.2725926052826416

When max.abs is greater than or equal to 1, rand returns a pseudo-random integer greater than or equal to 0 and less than max.to\_i.abs.

### rand(100) #=> 1

When max is a <u>Range</u>, rand returns a random number where range.member?(number) == true.

Negative or floating point values for max are allowed, but may give surprising results.



•

<u>#srand</u> may be used to ensure that sequences of random numbers are reproducible between different runs of a program.

See also Random#rand.

## $_{\odot}$ readline(sep=\$/) $\rightarrow$ string

 $\odot$  readline(limit)  $\rightarrow$  string

### $_{\odot}$ readline(sep, limit) $\rightarrow$ string

Equivalent to Kernel::gets, except readline raises EOFError at end of file.

## $_{\odot}$ readlines(sep=\$/) $\rightarrow$ array

### $_{\odot}$ readlines(limit) $\rightarrow$ array

## oreadlines(sep,limit) → array

Returns an array containing the lines returned by calling Kernel.gets(*sep*) until the end of file.

## or require(name) → true or false or false

Loads the given name, returning true if successful and false if the feature is already loaded.

If the filename does not resolve to an absolute path, it will be searched for in the directories listed in \$LOAD\_PATH (\$:).

If the filename has the extension ".rb", it is loaded as a source file; if the extension is ".so", ".o", or ".dll", or the default shared library extension on the current platform, Ruby loads the shared library as a Ruby extension. Otherwise, Ruby tries adding ".rb", ".so", and so on to the name until found. If the file named cannot be found, a LoadError will be raised.

Þ

For Ruby extensions the filename given may use any shared library extension. For example, on Linux the socket extension is "socket.so" and require 'socket.dll' will load the socket extension.

The absolute path of the loaded file is added to \$LOADED\_FEATURES (\$"). A file will not be loaded again if its path already appears in \$". For example, require 'a'; require './a' will not load a.rb again.

```
require "my-library.rb"
require "db-driver"
```

Any constants or globals within the loaded source file will be available in the calling program's global namespace. However, local variables will not be propagated to the loading environment.

### $\odot$ require\_relative(string) $\rightarrow$ true or false

Ruby tries to load the library named *string* relative to the requiring file's path. If the file's path cannot be determined a LoadError is raised. If a file is loaded true is returned and false otherwise.

### select(read\_array

- [, write\_array
- [, error\_array
  ]

### (, timeout]]]) → array or nil

Calls select(2) system call. It monitors given arrays of 10 objects, waits one or more of 10 objects ready for reading, are ready for writing, and have pending exceptions respectively, and returns an array that contains arrays of those <u>IO</u> objects. It will return nil if optional *timeout* value is given and no 10 object is ready in *timeout* seconds.

IO.select peeks the buffer of IO objects for testing readability. If the IO buffer is not empty, IO.select immediately notify readability. This "peek" is only happen for IO objects. It is not happen for IO-like objects such as OpenSSL::SSL::SSLSocket.

The best way to use IO.select is invoking it after nonblocking methods such as read\_nonblock, write\_nonblock, etc. The methods raises an exception which is extended by IO::WaitReadable Or IO::WaitWritable. The modules notify how the caller should wait with IO.select. If IO::WaitReadable is raised, the caller should wait for reading. If IO::WaitWritable is raised, the caller should wait for writing.

So, blocking read (readpartial) can be emulated using read\_nonblock and IO.select as follows:

```
begin
  result = io_like.read_nonblock(maxlen)
rescue IO::WaitReadable
  IO.select([io_like])
  retry
rescue IO::WaitWritable
  IO.select(nil, [io_like])
  retry
end
```

Especially, the combination of nonblocking methods and IO.select is preferred for IO like objects such as OpenSSL::SSL::SSLSocket. It has to\_io method to return underlying IO object. IO.select calls to\_io to obtain the file descriptor to wait.

This means that readability notified by IO.select doesn't mean readability from OpenSSL::SSL::SSLSocket Object. Most possible situation is <code>OpenSSL::SSL::SSLSocket</code> buffers some data. <code>IO.select</code> doesn't see the buffer. So <code>IO.select</code> can block when

OpenSSL::SSL::SSLSocket#readpartial doesn't block.

However several more complicated situation exists.

SSL is a protocol which is sequence of records. The record consists multiple bytes. So, the remote side of SSL sends a partial record, IO.select notifies readability but OpenSSL::SSLSocket cannot decrypt a byte and

OpenSSL::SSL::SSLSocket#readpartial will blocks.

Also, the remote side can request SSL renegotiation which forces the local SSL engine writes some data. This means <code>OpenSSL::SSLSocket#readpartial</code> may invoke write system call and it can block. In such situation,

OpenSSL::SSL::SSLSocket#read\_nonblock raises IO::WaitWritable instead of blocking. So, the caller should wait for ready for writability as above example.

The combination of nonblocking methods and IO.select is also useful for streams such as tty, pipe socket socket when multiple process read form a stream.

Finally, Linux kernel developers doesn't guarantee that readability of select(2) means readability of following read(2) even for single process. See select(2) manual on GNU/Linux system.

Invoking IO.select before IO#readpartial works well in usual. However it is not the best way to use IO.select.

The writability notified by select(2) doesn't show how many bytes writable. IO#write method blocks until

given whole string is written. So, IO#write(two or more bytes) can block after writability is notified by IO.select. IO#write\_nonblock is required to avoid the blocking.

Blocking write (write) can be emulated using write\_nonblock and IO.select as follows: IO::WaitReadable should also be rescued for SSL renegotiation in OpenSSL::SSLSocket.

```
while 0 < string.bytesize
    begin
    written = io_like.write_nonblock(string)
    rescue IO::WaitReadable
    IO.select([io_like])
    retry
    rescue IO::WaitWritable
    IO.select(nil, [io_like])
        retry
    end
    string = string.byteslice(written..-1)
end</pre>
```

### Parameters

read\_array an array of 10 objects that wait until ready for read

### write\_array

an array of 10 objects that wait until ready for write

error\_array

an array of 10 objects that wait for exceptions

#### timeout

a numeric value in second

### Example



```
mesg = "ping "
<u>100.times {</u>
  # IO.select follows IO#read. Not the best way
  rs, ws, = IO.select([rp], [wp])
  if r = rs[0]
    ret = r.read(5)
    print ret
    case ret
    when /ping/
      mesg = "pong n"
    when /pong/
      mesg = "ping "
    end
  end
  if w = ws[0]
    w.write(mesg)
  end
}
4
```

produces:

ping pong	
ping pong	
ping pong	
(snipped)	
ning	

## set\_trace\_func(proc) → proc set\_trace\_func(nil) → nil

Establishes *proc* as the handler for tracing, or disables tracing if the parameter is nil.

**Note:** this method is obsolete, please use <u>TracePoint</u> instead.

proc takes up to six parameters:

- 📄 an event name
- 📄 a filename
- 📄 a line number
- an object id

a binding

the name of a class

proc is invoked whenever an event occurs.

Events are:

```
c-call
```

call a C-language routine

c-return return from a C-language routine

call

call a Ruby method

### class

start a class or module definition),

end

finish a class or module definition),

### line

execute code on a new line

### raise

raise an exception

### return

return from a Ruby method

Tracing is disabled within the context of proc.

```
class Test
def test
  a = 1
  b = 2
end
end
set_trace_func proc { |event, file, line, id, k
  printf "%8s %s:%-2d %10s %8s\n", event, file
}
t = Test.new
t.test
line prog.rb:11 false
```

c-call	prog.rb:11	new	Class	
c-call	prog.rb:11	initialize	0bject	
c-return	prog.rb:11	initialize	0bject	
c-return	prog.rb:11	new	Class	
line	prog.rb:12		false	
call	prog.rb:2	test	Test	
line	prog.rb:3	test	Test	
line	prog.rb:4	test	Test	
return	prog.rb:4	test	Test	
<b>↓</b>			ĺ	Þ

## Sleep([duration]) → fixnum

Suspends the current thread for *duration* seconds (which may be any number, including a Float with fractional seconds). Returns the actual number of seconds slept (rounded), which may be less than that asked for if another thread calls Thread#run. Called without an argument, sleep() will sleep forever.



 $_{\odot}$  spawn([env,] command... [,options]) → pid  $_{\odot}$  spawn([env,] command... [,options]) → pid

spawn executes specified command and return its pid.



This method is similar to <u>#system</u> but it doesn't wait for the command to finish.

The parent process should use Process.wait to collect the termination status of its child or use Process.detach to register disinterest in their status; otherwise, the operating system may accumulate zombie processes.

spawn has bunch of options to specify process attributes:

```
env: hash
  name => val : set the environment variable
  name => nil : unset the environment variable
command...:
  commandline: command line stricmdname, arg1, ...: command name and[cmdname, argv0], arg1, ...: command name, arg
options: hash
  clearing environment variables:
    :unsetenv_others => true : clear environmer
    :unsetenv_others => false : don't clear (def
  process group:
    :pgroup => true or 0 : make a new process gro
    :pgroup => pgid : join to specified prod
    :pgroup => nil : don't change the proce
  create new process group: Windows only
    :new_pgroup => true : the new process is the
    :new_pgroup => false : don't create a new pro
  resource limit: resourcename is core, cpu, data
    :rlimit_resourcename => limit
    :rlimit_resourcename => [cur_limit, max_limit
  umask:
    :umask => int
  redirection:
    key:
      FD : single file descriptor ir
[FD, FD, ...] : multiple file descriptor
    value:
      FD
                                 : redirect to the
                                 : redirect to fil
      string
                                 : redirect to fil
      [string]
      [string, open_mode] : redirect to fi
      [string, open_mode, perm] : redirect to fil
                           : redirect to the
      [:child, FD]
      :close
                                 : close the file
    FD is one of follows
```



If a hash is given as env, the environment is updated by env before exec(2) in the child process. If a pair in env has nil as the value, the variable is deleted.



If a hash is given as options, it specifies process group, create new process group, resource limit, current directory, umask and redirects for the child process. Also, it can be specified to clear environment variables.

The :unsetenv\_others key in options specifies to clear environment variables, other than specified by env.



The :pgroup key in options specifies a process group. The corresponding value should be true, zero or positive integer. true and zero means the process should be a process leader of a new process group. Other values specifies a process group to be belongs.



The :new\_pgroup key in options specifies to pass CREATE\_NEW\_PROCESS\_GROUP flag to CreateProcessW() that is Windows API. This option is only for Windows. true means the new process is the root process of the new process group. The new process has CTRL+C disabled. This flag is necessary for Process.kill(:SIGINT, pid) on the subprocess. :new\_pgroup is false by default.



The :rlimit\_foo key specifies a resource limit. foo should be one of resource types such as core. The corresponding value should be an integer or an array which have one or two integers: same as cur\_limit and max\_limit arguments for Process.setrlimit.



The :umask key in options specifies the umask.

```
pid = spawn(command, :umask=>077)
```

The :in, :out, :err, a fixnum, an <u>IO</u> and an array key specifies a redirection. The redirection maps a file descriptor in the child process.

For example, stderr can be merged into stdout as follows:

pid =	spawn(command,	:err=>:out)
pid =	spawn(command,	2=>1)
pid =	spawn(command,	<pre>STDERR=&gt;:out)</pre>
pid =	spawn(command,	STDERR=>STDOUT)

The hash keys specifies a file descriptor in the child process started by spawn. :err, 2 and STDERR specifies the standard error stream (stderr).

The hash values specifies a file descriptor in the parent process which invokes spawn. :out, 1 and STDOUT specifies the standard output stream (stdout).

In the above example, the standard output in the child process is not specified. So it is inherited from the parent process.

The standard input stream (stdin) can be specified by :in, 0 and STDIN.

A filename can be specified as a hash value.



For stdout and stderr (and combination of them), it is opened in write mode. Otherwise read mode is used.

For specifying flags and permission of file creation explicitly, an array is used instead.

<pre>pid = pid = pid = pid = pid = pid =</pre>	<pre>spawn(command, spawn(command, spawn(command, spawn(command, spawn(command,</pre>	<pre>:in=&gt;["file"]] :in=&gt;["file", :out=&gt;["log", :out=&gt;["log", :out=&gt;["log",</pre>	)
. <b>▲</b>			D

. ◀ ]

The array specifies a filename, flags and permission. The flags can be a string or an integer. If the flags is omitted or nil, File::RDONLY is assumed. The permission should be an integer. If the permission is omitted or nil, 0644 is assumed.

If an array of IOs and integers are specified as a hash key, all the elements are redirected.



Another way to merge multiple file descriptors is [:child, fd]. [:child, fd] means the file descriptor in the child process. This is different from fd. For example, :err=>:out means redirecting child stderr to parent stdout. But :err=>[:child, :out] means redirecting child stderr to child stdout. They differ if stdout is redirected in the child process as follows.



[:child, :out] can be used to merge stderr into stdout in <u>IO.popen</u>. In this case, <u>IO.popen</u> redirects stdout to a pipe in the child process and [:child, :out] refers the redirected stdout.



The :chdir key in options specifies the current directory.



spawn closes all non-standard unspecified

descriptors by default. The "standard" descriptors are 0, 1 and 2. This behavior is specified by :close\_others option. :close\_others doesn't affect the standard descriptors which are closed only if :close is specified explicitly.



:close\_others is true by default for spawn and <u>IO.popen</u>.

Note that fds which close-on-exec flag is already set are closed regardless of :close\_others option.

So IO.pipe and spawn can be used as IO.popen.



close is specified as a hash value to close a fd individually.



If a file descriptor need to be inherited, io=>io can be used.

```
# valgrind has --log-fd option for log destination
# log_w=>log_w indicates log_w.fileno inherits to
log_r, log_w = IO.pipe
pid = spawn("valgrind", "--log-fd=#{log_w.fileno]
log_w.close
p log_r.read
```

It is also possible to exchange file descriptors.

pid = spawn(command, :out=>:err, :err=>:out)

The hash keys specify file descriptors in the child process. The hash values specifies file descriptors in the parent process. So the above specifies exchanging stdout and stderr. Internally, spawn uses an extra file descriptor to resolve such cyclic file descriptor mapping.

See Kernel.exec for the standard shell.

# format(format\_string [, arguments...]) $\rightarrow$ $_{\textcircled{}}$ string

# sprintf(format\_string [, arguments...]) $\rightarrow$ $\otimes$ string

Returns the string resulting from applying *format\_string* to any additional arguments. Within the format string, any characters other than format sequences are copied to the result.

The syntax of a format sequence is follows.

### %[flags][width][.precision]type

A format sequence consists of a percent sign, followed by optional flags, width, and precision indicators, then terminated with a field type character. The field type controls how the corresponding sprintf argument is to be interpreted, while the flags modify that interpretation.

The field type characters are:

Field	Integer Format
b	<pre>  Convert argument as a binary number.   Negative numbers will be displayed as a t   prefixed with `1'.</pre>
В	<pre>  Equivalent to `b', but uses an uppercase   in the alternative format by #.</pre>
d	Convert argument as a decimal number.

i	Identical to `d'.
0	Convert argument as an octal number.
	Negative numbers will be displayed as a t
	prefixed with `7'
u	Identical to `d'.
X	Convert argument as a hexadecimal number
	Negative numbers will be displayed as a t
	prefixed witht' (representing an infi
V	
Field	Float Format
+	Convert floating point argument into ever
e	with one digit before the decimal point a
	The precision specifies the number of die
	point (defaulting to six).
E	Equivalent to `e', but uses an uppercase
	the exponent.
f _	Convert floating point argument as [-]ddd
	where the precision specifies the number
	the decimal point.
g	Convert a floating point number using exp
	if the exponent is less than -4 or greate
	equal to the precision, or in dd.dddd for
	The precision specifies the number of sig
G	Equivalent to g', but use an uppercase
a	which is consisted from optional sign -
	as hevadecimal "n" and exponential part
A	Further to $a'$ but use unnercase $x'$
Field	Other Format
_C	Argument is the numeric code for a single
	a single character string itself.
р	The valuing of argument.inspect.
S	Argument is a string to be substituted.
	sequence contains a precision, at most th
	will be copied.
%	A percent sign itself will be displayed.
4	

The flags modifies the behavior of the formats. The flag characters are:

Flag	Applies to	Meaning	
------	------------	---------	--

space	bBdiouxX aAeEfgG (numeric fmt)	Leave a space at the s non-negative numbers. For `o', `x', `X', `b' a minus sign with abso negative values.
(digit)\$	all	Specifies the absolute for this field. Absol argument numbers canno sprintf string.
#	bBoxX aAeEfgG	Use an alternative for For the conversions `d until the first digit it is not formatted as For the conversions `> on non-zero, prefix th `OX'', `Ob'' and ``G For `a', `A', `e', `E' force a decimal point even if no digits fol For `g' and 'G', do no
+	bBdiouxX aAeEfgG (numeric fmt)	Add a leading plus sig numbers. For `o', `x', `X', `b' a minus sign with abso negative values.
-	all	Left-justify the resul
0 (zero)	bBdiouxX aAeEfgG (numeric fmt)	Pad with zeros, not sp For `o', `x', `X', `b' is used for negative r complements.
*	all	Use the next argument If negative, left-just asterisk is followed k sign, use the indicate
4		•

Examples of flags:

# `+' and space flag specifies the sign of non-ne sprintf("%d", 123) #=> "123"

sprintf("%+d", 123)
sprintf("% d", 123) sprintf("%0", 123) sprintf("%#0", 123) sprintf("%+0", -123)
sprintf("%0", -123)
sprintf("%#0", -123) sprintf("%x", 123) sprint('%x', 123)
sprintf('%#x", 123)
sprintf('%+x", -123)
sprintf('%x", -123)
sprintf('%#x", -123)
sprintf('%#x", 0) sprintf("%X", 123) sprintf("%#X", 123) sprintf("%b", 123) sprintf("%#b", 123)
sprintf("%+b", -123) sprintf("%b", -123)
sprintf("%#b", -123) sprintf("%#b", 0) sprintf("%B", 123) sprintf("%#B", 123) sprintf("%.0e", 1) sprintf("%#.0e", 1) sprintf("%.0f", 1234) sprintf("%#.0f", 1234)



The field width is an optional integer, followed optionally by a period and a precision. The width specifies the minimum number of characters that will be written to the result for this field.

Examples of width:

	, width=20
	<>
sprintf("%20d", 123)	" 123'
sprintf("%+20d", 123)	"+123'
sprintf("%020d", 123)	"0000000000000000000123'
<pre>sprintf("%+020d", 123)</pre>	"+00000000000000000123'
<pre>sprintf("% 020d", 123)</pre>	" 00000000000000000123'
sprintf("%-20d", 123)	"123 '
<pre>sprintf("%-+20d", 123)</pre>	"+123 '
<pre>sprintf("%- 20d", 123)</pre>	" 123 '
<pre>sprintf("%020x", -123)</pre>	"ffffffffffffffffff
4	

For numeric fields, the precision controls the number of decimal places displayed. For string fields, the precision determines the maximum number of characters to be copied from the string. (Thus, the format sequence %10.10s will always contribute exactly ten characters to the result.)

Examples of precisions:

<pre># precision for `</pre>			
# minimum number (			
<pre>sprintf("%20.8d",</pre>	123)		
<pre>sprintf("%20.80",</pre>	123)		
<pre>sprintf("%20.8x",</pre>	123)		
<pre>sprintf("%20.8b",</pre>	123)		
<pre>sprintf("%20.8d",</pre>	-123)		
<pre>sprintf("%20.80",</pre>	-123)		
<pre>sprintf("%20.8x",</pre>	-123)		
<pre>sprintf("%20.8b",</pre>	-11)		

<pre># "0x" and "0b" for `#x' and `#b # precision but "0" for `#o' is a sprintf("%#20.8d", 123) #=&gt; " sprintf("%#20.8o", 123) #=&gt; " sprintf("%#20.8x", 123) #=&gt; " sprintf("%#20.8b", 123) #=&gt; " sprintf("%#20.8d", -123) #=&gt; " sprintf("%#20.8o", -123) #=&gt; " sprintf("%#20.8x", -123) #=&gt; " sprintf("%#20.8b", -11) #=&gt; "</pre>	
<pre># precision for `e' is number of # digits after the decimal point sprintf("%20.8e", 1234.56789) #=</pre>	
<pre># precision for `f' is number of # digits after the decimal point sprintf("%20.8f", 1234.56789) #=</pre>	
<pre># precision for `g' is number of # significant digits sprintf("%20.8g", 1234.56789) #=</pre>	
# sprintf("%20.8g", 123456789) #=:	
<pre># precision for `s' is # maximum number of characters sprintf("%20.8s", "string test") </pre>	#=> "

Examples:



For more complex formatting, Ruby supports a reference by name. %<name>s style uses format style, but %{name} style doesn't.

Examples:



# srand(number = Random.new\_seed) → old\_seed

Seeds the system pseudo-random number generator, Random::DEFAULT, with number. The previous seed value is returned.

If number is omitted, seeds the generator using a source of entropy provided by the operating system, if available (/dev/urandom on Unix systems or the RSA cryptographic provider on Windows), which is then combined with the time, the process id, and a sequence number.

srand may be used to ensure repeatable sequences of pseudo-random numbers between different runs of the program. By setting the seed to a known value, programs can be made deterministic during testing.



# $_{\odot}$ sub(pattern, replacement) → \$\_ $_{\odot}$ sub(pattern) {|...| block } → \$\_

Equivalent to \$\_.sub(args), except that \$\_ will be updated if substitution occurs. Available only when p/-n command line option specified.

### $_{\odot}$ syscall(num [, args...]) $\rightarrow$ integer

Calls the operating system function identified by returns the result of the function or raises Syst it failed.

Arguments for the function can follow \_num\_. They +String+ objects or +Integer+ objects. A +Stringas a pointer to the byte sequence. An +Integer+ of as an integer whose bit size is same as a pointer Up to nine parameters may be passed (14 on the At

The function identified by \_num\_ is system dependent. On some Unix systems, the numbers may header file called <code>syscall.h</code>.

<em>produces:</em>

hello

Calling +syscall+ on a platform which does not ha an arbitrary system function just fails with Not!

#### Note

syscall is essentially unsafe and unportable. Feel free to shoot your foot. DL (Fiddle) library is preferred for safer and a bit more portable programming.

### system([env,] command... [,options]) $\rightarrow$ true, $_{\odot}$ false or nil

Executes *command...* in a subshell. *command...* is one of following forms.

commandline	:	command	line	string
cmdname, arg1,	:	command	name	and or
[cmdname, argv0], arg1,	:	command	name,	argv[
<u>الا</u>				Þ

system returns true if the command gives zero exit status, false for non zero exit status. Returns nil if command execution fails. An error status is available in \$?. The arguments are processed in the same way as for Kernel.spawn.

The hash arguments, env and options, are same as exec and spawn. See Kernel.spawn for details.

```
system("echo *")
system("echo", "*")
```

produces:

```
config.h main.rb
```

See Kernel.exec for the standard shell.

### $_{\odot}$ test(cmd, file1 [, file2] ) → obj

Uses the character cmd to perform various tests on file1 (first table below) or on file1 and file2 (second table).

File tests on a single file:

Cmd	Returns	Meaning
"A"	Time	Last access time for file1
"b"	boolean	True if file1 is a block device
"c"	boolean	True if file1 is a character dev
"C"	Time	Last change time for file1
"d"	boolean	True if file1 exists and is a di
"e"	boolean	True if file1 exists
"f"	boolean	True if file1 exists and is a re
"g"	boolean	True if file1 has the setgid
		set (false under NT)
"G"	boolean	True if file1 exists and has a 🤉
		ownership equal to the caller's
"k"	boolean	True if file1 exists and has the
"1"	boolean	True if file1 exists and is a sy
"M"	Time	Last modification time for file
"0"	boolean	True if file1 exists and is owne

		the caller's effective uid
"0"	boolean	True if file1 exists and is owne
		the caller's real uid
"p"	boolean	True if file1 exists and is a fi
"r"	boolean	True if file1 is readable by the
		uid/gid of the caller
"R"	boolean	True if file is readable by the
		uid/gid of the caller
"S"	int/nil	If file1 has nonzero size, retur
		otherwise return nil
"S"	boolean	True if file1 exists and is a so
"u"	boolean	True if file1 has the setuid bit
"w"	boolean	True if file1 exists and is writ
		the effective uid/gid
"W"	boolean	True if file1 exists and is writ
		the real uid/gid
"x"	boolean	True if file1 exists and is exec
		the effective uid/gid
"X"	boolean	True if file1 exists and is exec
		the real uid/gid
"z"	boolean	True if file1 exists and has a z
4		×

Tests that take two files:

"_" "="	boolean     boolean	True if file1 and file2 are ider True if the modification times of
"<"	boolean	and file2 are equal True if the modification time of
">"	boolean	True if the modification time of
4		

# throw(tag [, obj])

Transfers control to the end of the active catch block waiting for *tag*. Raises UncaughtThrowError if there is no catch block for the *tag*. The optional second parameter supplies a return value for the catch block, which otherwise defaults to nil. For examples, see Kernel::catch.

# $_{\odot}$ trace\_var(symbol, cmd ) → nil $_{\odot}$ trace\_var(symbol) {|val| block } → nil

Controls tracing of assignments to global variables. The parameter symbol identifies the variable (as either a string name or a symbol identifier). *cmd* (which may be a string or a Proc object) or block is executed whenever the variable is assigned. The block or Proc object receives the variable's new value as a parameter. Also see Kernel::untrace\_var.



produces:

```
$_ is now 'hello'
$_ is now ' there'
```

# Trap( signal, command ) → obj Trap( signal ) {| | block } → obj

Specifies the handling of signals. The first parameter is a signal name (a string such as "SIGALRM", "SIGUSR1", and so on) or a signal number. The characters "SIG" may be omitted from the signal name. The command or block specifies code to be run when the signal is raised. If the command is the string "IGNORE" or "SIG\_IGN", the signal will be ignored. If the command is "DEFAULT" or "SIG\_DFL", the Ruby's default handler will be invoked. If the command is "EXIT", the script will be terminated by the signal. If the command is "SYSTEM\_DEFAULT", the operating system's default handler will be invoked. Otherwise, the given command or block will be run. The special signal name "EXIT" or signal number zero will be invoked just prior to program termination. trap returns the previous handler for the given signal.



produces:

```
Terminating: 27461
Child died
Terminating: 27460
```

### $_{\odot}$ untrace\_var(symbol [, cmd]) $\rightarrow$ array or nil

Removes tracing for the specified command on the given global variable and returns nil. If no command is specified, removes all tracing for that variable and returns an array containing the commands actually removed.

### $_{\odot}$ warn(msg, ...) → nil

Displays each of the given messages followed by a record separator on STDERR unless warnings have been disabled (for example with the -wo flag).

```
warn("warning 1", "warning 2")
<em>produces:</em>
warning 1
warning 2
```

# class KeyError

Raised when the specified key is not found. It is a subclass of IndexError.



# In Files

error.c

# Parent

IndexError

# class LoadError

Raised when a file required (a Ruby script, extension library, ...) fails to load.

require 'this/file/does/not/exist'

raises the exception:

LoadError: no such file to load -- this/file/o

In Files

Parent

ScriptError

# **Attributes**

rb\_intern\_const("path")<sup>[R]</sup> the path failed to load

# class LocalJumpError

Raised when Ruby can't yield as requested.

A typical scenario is attempting to yield when no block is given:

```
def call_block
yield 42
end
call_block
```

raises the exception:

LocalJumpError: no block given (yield)

A more subtle example:

```
def get_me_a_return
    Proc.new { return 42 }
end
get_me_a_return.call
```

raises the exception:

LocalJumpError: unexpected return



Parent

StandardError

# **Public Instance Methods**

### ⊚ exit\_value → obj

Returns the exit value associated with this LocalJumpError.

## reason → symbol

The reason this block was terminated: :break, :redo, :retry, :next, :return, or :noreason.

# module Marshal

The marshaling library converts collections of Ruby objects into a byte stream, allowing them to be stored outside the currently active script. This data may subsequently be read and the original objects reconstituted.

Marshaled data has major and minor version numbers stored along with the object information. In normal use, marshaling can only load data written with the same major version number and an equal or lower minor version number. If Ruby's "verbose" flag is set (normally using -d, -v, -w, or –verbose) the major and minor numbers must match exactly. <u>Marshal</u> versioning is independent of Ruby's version numbers. You can extract the version by reading the first two bytes of marshaled data.

```
str = Marshal.dump("thing")
RUBY_VERSION  #=> "1.9.0"
str[0].ord  #=> 4
str[1].ord  #=> 8
```

Some objects cannot be dumped: if the objects to be dumped include bindings, procedure or method objects, instances of class <u>IO</u>, or singleton objects, a TypeError will be raised.

If your class has special serialization needs (for
example, if you want to serialize in some specific format), or if it contains objects that would otherwise not be serializable, you can implement your own serialization strategy.

There are two methods of doing this, your object can define either marshal\_dump and marshal\_load or \_dump and \_load. marshal\_dump will take precedence over \_dump if both are defined. marshal\_dump may result in smaller <u>Marshal</u> strings.

# **Security considerations**

By design, <u>::load</u> can deserialize almost any class loaded into the Ruby process. In many cases this can lead to remote code execution if the <u>Marshal</u> data is loaded from an untrusted source.

As a result, <u>::load</u> is not suitable as a general purpose serialization format and you should never unmarshal user supplied input or other untrusted data.

If you need to deserialize untrusted data, use JSON or another serialization format that is only able to load simple, 'primitive' types such as <u>String</u>, <u>Array</u>, <u>Hash</u>, etc. Never allow user input to specify arbitrary types to deserialize into.

# marshal\_dump and marshal\_load

When dumping an object the method marshal\_dump will be called. marshal\_dump must return a result containing the information necessary for marshal\_load to reconstitute the object. The result can be any object.

When loading an object dumped using marshal\_dump the object is first allocated then marshal\_load is called with the result from marshal\_dump. marshal\_load must recreate the object from the information in the result.

Example:

```
class MyObj
def initialize name, version, data
  @name = name
  @version = version
  @data = data
end
def marshal_dump
  [@name, @version]
end
def marshal_load array
  @name, @version = array
end
end
```

# \_dump and \_load

Use \_dump and \_load when you need to allocate the object you're restoring yourself.

When dumping an object the instance method \_dump is called with an <u>Integer</u> which indicates the maximum depth of objects to dump (a value of -1 implies that you should disable depth checking). \_dump must return a <u>String</u> containing the information necessary to reconstitute the object.

The class method \_load should take a <u>String</u> and use it to return an object of the same class.

Example:

```
class MyObj
def initialize name, version, data
  @name = name
  @version = version
  @data = data
end
def _dump level
  [@name, @version].join ':'
end
def self._load args
  new(*args.split(':'))
end
end
```

Since ::dump outputs a string you can have

\_dump return a <u>Marshal</u> string which is Marshal.loaded in \_load for complex objects.

**In Files** 

📄 marshal.c

# Constants

#### MAJOR\_VERSION

major version

#### MINOR\_VERSION

minor version

# **Public Class Methods**

#### $_{\odot}$ dump( obj [, anIO] , limit=-1 ) $\rightarrow$ anIO

Serializes obj and all descendant objects. If an<u>IO</u> is specified, the serialized data will be written to it, otherwise the data will be returned as a <u>String</u>. If limit is specified, the traversal of subobjects will be limited to that depth. If limit is negative, no checking of depth will be performed.

```
class Klass
  def initialize(str)
    @str = str
  end
  def say_hello
    @str
  end
end
```

(produces no output)

```
o = Klass.new("hello\n")
data = Marshal.dump(o)
obj = Marshal.load(data)
obj.say_hello #=> "hello\n"
```

Marshal can't dump following objects:

- anonymous Class/Module.
- objects which are related to system (ex: <u>Dir</u>, File::Stat, IO, File, Socket and so on)
- an instance of <u>MatchData</u>, <u>Data</u>, <u>Method</u>, <u>UnboundMethod</u>, <u>Proc</u>, <u>Thread</u>, <u>ThreadGroup</u>, <u>Continuation</u>
- objects which define singleton methods

# $_{\odot}$ load( source [, proc] ) → obj $_{\odot}$ restore( source [, proc] ) → obj

Returns the result of converting the serialized data in source into a Ruby object (possibly with associated subordinate objects). source may be either an instance of <u>IO</u> or an object that responds to to\_str. If proc is specified, each object will be passed to the proc, as the object is being deserialized.

Never pass untrusted data (including user supplied input) to this method. Please see the overview for further details.

# rightarrow load( source [, proc] ) → obj rightarrow restore( source [, proc] ) → obj

Returns the result of converting the serialized data in source into a Ruby object (possibly with associated subordinate objects). source may be either an instance of <u>IO</u> or an object that responds to to\_str. If

proc is specified, each object will be passed to the proc, as the object is being deserialized.

Never pass untrusted data (including user supplied input) to this method. Please see the overview for further details.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class MatchData

MatchData is the type of the special variable \$~, and is the type of the object returned by Regexp#match and Regexp.last\_match. It encapsulates all the results of a pattern match, results normally accessed through the special variables \$&, \$', \$`, \$1, \$2, and so on.

# **In Files**

📄 re.c

# Parent

**Object** 

# **Public Instance Methods**

# $_{\odot}$ mtch == mtch2 $\rightarrow$ true or false

#### $\odot$ eql?(mtch2) $\rightarrow$ true or false

Equality—Two matchdata are equal if their target strings, patterns, and matched positions are identical.

mtch[i] → str or nilmtch[start, length] → arraymtch[range] → array

mtch[name] → str or nil

Match Reference – MatchData acts as an array, and may be accessed using the normal array indexing techniques. mtch[0] is equivalent to the special variable \$&, and returns the entire matched string. mtch[1], mtch[2], and so on return the values of the matched backreferences (portions of the pattern between parentheses).



#### begin(n) → integer

Returns the offset of the start of the *n*th element of the match array in the string. *n* can be a string or symbol to reference a named capture.



#### captures → array

Returns the array of captures; equivalent to mtch.to\_a[1..-1].

 $f_1, f_2, f_3, f_4 = /(.)(.)(\d+)(\d)/.match("THX1138.")$ 



# $\odot$ end(n) $\rightarrow$ integer

Returns the offset of the character immediately following the end of the *n*th element of the match array in the string. *n* can be a string or symbol to reference a named capture.



# rightarrow mtch == mtch2 → true or false rightarrow eql?(mtch2) → true or false

Equality—Two matchdata are equal if their target strings, patterns, and matched positions are identical.

#### $_{\odot}$ hash $\rightarrow$ integer

Produce a hash based on the target string, regexp and matched positions of this matchdata.

See also Object#hash.

#### $\odot$ inspect $\rightarrow$ str

Returns a printable version of *mtch*.

puts /.\$/.match("foo").inspect
#=> #<MatchData "o">



#### $_{\odot}$ length $\rightarrow$ integer

#### $\odot$ size $\rightarrow$ integer

Returns the number of elements in the match array.



# $_{\odot}$ names $\rightarrow$ [name1, name2, ...]

Returns a list of names of captures as an array of strings. It is same as mtch.regexp.names.



# $\odot$ offset(n) $\rightarrow$ array

Returns a two-element array containing the beginning and ending offsets of the *n*th match. *n* can be a string or symbol to reference a named capture.





#### String → str Returns a frozen copy of the string passed in to

match.



#### oto\_a → anArray

Returns the array of matches.



Because to\_a is called when expanding \**variable*, there's a useful assignment shortcut for extracting matched fields. This is slightly slower than accessing the fields directly (as an intermediate array is generated).



#### ⊚ to\_s → str

Returns the entire matched string.



# $\odot$ values\_at([index]\*) $\rightarrow$ array

Uses each *index* to access the matching values, returning an array of the corresponding matches.



Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# module Math

The <u>Math</u> module contains module functions for basic trigonometric and transcendental functions. See class <u>Float</u> for a list of constants that define Ruby's floating point accuracy.

Domains and codomains are given only for real (not complex) numbers.

# **In Files**

📄 math.c

# Constants

# Ε

Definition of the mathematical constant  $\underline{E}$  (e) as a  $\underline{Float}$  number.

#### PI

Definition of the mathematical constant  $\underline{\mathsf{PI}}$  as a  $\underline{\mathsf{Float}}$  number.

# **Public Class Methods**

 $\otimes$  acos(x)  $\rightarrow$  Float

Computes the arc cosine of x. Returns 0..PI.

Domain: [-1, 1]

Codomain: [0, PI]

Math.acos(0) == Math::PI/2 #=> tr

# acosh(x) → Float Computes the inverse hyperbolic cosine of x. Domain: [1, INFINITY) Codomain: [0, INFINITY) Math.acosh(1) #=> 0.0

asin(x) → Float
 Computes the arc sine of ×. Returns -PI/2..PI/2.
 Domain: [-1, -1]
 Codomain: [-PI/2, PI/2]

Math.asin(1) == Math::PI/2 #=> true

# $_{\odot}$ asinh(x) $\rightarrow$ Float

Computes the inverse hyperbolic sine of x.

Domain: (-INFINITY, INFINITY)

Codomain: (-INFINITY, INFINITY)

Math.asinh(1) #=> 0.881373587019543

# $_{\odot}$ atan(x) $\rightarrow$ Float

Computes the arc tangent of x. Returns -PI/2..PI/2.

Domain: (-INFINITY, INFINITY)

Codomain: (-PI/2, PI/2)

Math.atan(0) #=> 0.0

# $_{\odot}$ atan2(y, x) $\rightarrow$ Float

Computes the arc tangent given y and x. Returns a Float in the range -PI..PI.

Domain: (-INFINITY, INFINITY)

Codomain: [-<u>PI</u>, PI]

Math.atan2(-0.0, -1.0)				
Math.atan2(-1.0, -1.0)				
Math.atan2(-1.0, 0.0)				
Math.atan2(-1.0, 1.0)				
Math.atan2(-0.0, 1.0)				
Math.atan2(0.0, 1.0)				
Math.atan2(1.0, 1.0)				
Math.atan2(1.0, 0.0)				
Math.atan2(1.0, -1.0)				
Math.atan2(0.0, -1.0)				
Math.atan2(INFINITY, IN	FINITY)			
Math.atan2(INFINITY, -I	NFINITY) #=> 2.356194490:			
<pre>Math.atan2(-INFINITY, INFINITY) #=&gt; -0.785398163</pre>				
Math.atan2(-INFINITY, -	INFINITY) #=> -2.356194490			
4	Þ			

#### $\otimes$ atanh(x) $\rightarrow$ Float

Computes the inverse hyperbolic tangent of x.

Domain: (-1, 1)

Codomain: (-INFINITY, INFINITY)

Math.atanh(1) #=> Infinity

#### $\odot$ cbrt(x) $\rightarrow$ Float

Returns the cube root of x.

Domain: [0, INFINITY)

Codomain:[0, INFINITY)



# $\odot \cos(x) \rightarrow Float$

Computes the cosine of  $\times$  (expressed in radians). Returns a Float in the range -1.0..1.0.

Domain: (-INFINITY, INFINITY)

Codomain: [-1, 1]

Math.cos(Math::PI) #=> -1.0

# $\odot$ cosh(x) $\rightarrow$ Float

Computes the hyperbolic cosine of  $\times$  (expressed in radians).

Domain: (-INFINITY, INFINITY)

Codomain: [1, INFINITY)

Math.cosh(0) #=> 1.0

# ● erf(x) → Float Calculates the error function of x. Domain: (-INFINITY, INFINITY) Codomain: (-1, 1)

Math.erf(0) #=> 0.6

# $\odot$ erfc(x) $\rightarrow$ Float

Calculates the complementary error function of x.

Domain: (-INFINITY, INFINITY)

Codomain: (0, 2)

Math.erfc(0) #=> 1.0

# osep(x) → Float

Returns e\*\*x.

Domain: (-INFINITY, INFINITY)

Codomain: (0, INFINITY)

Math.exp(0)	
Math.exp(1)	
Math.exp(1.5)	

# ↔ frexp(x) → [fraction, exponent]

Returns a two-element array containing the normalized fraction (a  $\underline{Float}$ ) and exponent (a  $\underline{Fixnum}$ ) of x.



# gamma(x) → Float

Calculates the gamma function of x.

Note that gamma(n) is same as fact(n-1) for integer n > 0. However gamma(n) returns float and can be an approximation.

<pre>def fact(n) (1n).inject(1) { r,i  r*i } end 1.upto(26) { i  p [i, Math.gamma(i), fact(i-1)] ]</pre>
#=> [1, 1.0, 1]
# [2, 1.0, 1]
# [3, 2.0, 2]
# [4, 6.0, 6]
# [5, 24.0, 24]
# [6, 120.0, 120]
# [7, 720.0, 720]
# [8, 5040.0, 5040]
# [9, 40320.0, 40320]
# [10, 362880.0, 362880]
<b># [11, 3628800.0, 3628800]</b>
# [12, 39916800.0, 39916800]
# [13, 479001600.0, 479001600]
<b># [14, 6227020800.0, 6227020800]</b>
# [15, 87178291200.0, 87178291200]
<b>#</b> [16, 1307674368000.0, 1307674368000]
# [17, 20922789888000.0, 20922789888000]
<b>#</b> [18, 355687428096000.0, 355687428096000]
# [19, 6.402373705728e+15, 6402373705728000]
# [20, 1.21645100408832e+17, 121645100408832000
# [21, 2.43290200817664e+18, 243290200817664000
# [22, 5.109094217170944e+19, 5109094217170944
# [23, 1.1240007277776077e+21, 112400072777760
# [24, 2.5852016738885062e+22, 258520167388849
# [25, 6.204484017332391e+23, 6204484017332394
# [26, 1.5511210043330954e+25, 155112100433309

 $_{\odot}$  hypot(x, y)  $\rightarrow$  Float

Returns  $sqrt(x^{**2} + y^{**2})$ , the hypotenuse of a rightangled triangle with sides  $\times$  and y.



but avoid overflow by ::gamma for large x.

Math.lgamma(0) #=> [Infinity, 1]

# log(x) → Float

#### $\odot \log(x, base) \rightarrow Float$

Returns the logarithm of x. If additional second argument is given, it will be the base of logarithm. Otherwise it is e (for the natural logarithm).

Domain: (0, INFINITY)

Codomain: (-INFINITY, INFINITY)

```
Math.log(0) #=> -Infinity
Math.log(1) #=> 0.0
Math.log(Math::E) #=> 1.0
Math.log(Math::E**3) #=> 3.0
```

Math.log(12, 3) #=> 2.26185950714291

# $\odot$ log10(x) $\rightarrow$ Float

Returns the base 10 logarithm of x.

Domain: (0, INFINITY)

Codomain: (-INFINITY, INFINITY)

# $\odot \log 2(x) \rightarrow Float$

Returns the base 2 logarithm of x.

Domain: (0, INFINITY)

Codomain: (-INFINITY, INFINITY)

Math.log2(1)	
Math.log2(2)	
Math.log2(32768)	
Math.log2(65536)	

#### $\otimes sin(x) \rightarrow Float$

Computes the sine of × (expressed in radians). Returns a <u>Float</u> in the range -1.0..1.0. Domain: (-INFINITY, INFINITY)

Codomain: [-1, 1]

Math.sin(Math::PI/2) #=> 1.0

# $\otimes$ sinh(x) $\rightarrow$ Float

Computes the hyperbolic sine of  $\times$  (expressed in radians).

#### Domain: (-INFINITY, INFINITY)

Codomain: (-INFINITY, INFINITY)

Math.sinh(0) #=> 0.0

# $\odot$ sqrt(x) $\rightarrow$ Float

Returns the non-negative square root of x.

Domain: [0, INFINITY)

Codomain:[0, INFINITY)



# $rightarrow tan(x) \rightarrow Float$

Computes the tangent of x (expressed in radians).

Domain: (-INFINITY, INFINITY)

Codomain: (-INFINITY, INFINITY)

# $_{\odot}$ tanh(x) $\rightarrow$ Float

Computes the hyperbolic tangent of  $\times$  (expressed in radians).

Domain: (-INFINITY, INFINITY)

Codomain: (-1, 1)

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class Math::DomainError

Raised when a mathematical function is evaluated outside of its domain of definition.

For example, since cos returns values in the range -1..1, its inverse function acos is only defined on that interval:

```
Math.acos(42)
```

produces:

Ν	Math::DomainError:	Numerical	argument	is	out
4					

In Files

📄 math.c

# Parent

StandardError

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class Method

Proc

# **In Files**

proc.c

# Parent

Object

# **Public Instance Methods**

# 

Two method objects are equal if they are bound to the same object and refer to the same method definition and their owners are the same class or module.

# rightarrow call(args, ...) → obj rightarrow meth[args, ...] → obj

Invokes the *meth* with the specified arguments, returning the method's return value.



## arity → fixnum

Returns an indication of the number of arguments accepted by a method. Returns a nonnegative integer for methods that take a fixed number of arguments. For Ruby methods that take a variable number of arguments, returns -n-1, where n is the number of required arguments. For methods written in C, returns -1 if the call takes a variable number of arguments.



#### $_{\odot}$ call(args, ...) → obj $_{\odot}$ meth[args, ...] → obj

Invokes the *meth* with the specified arguments, returning the method's return value.

m = 12.method	d("+")	
m.call(3)		
m.call(20)		

#### osignment of the second se

Returns a clone of this method.

```
class A
  def foo
    return "bar"
  end
end
m = A.new.method(:foo)
m.call # => "bar"
n = m.clone.call # => "bar"
```

# rightarrow curry → proc rightarrow curry(arity) → proc

Returns a curried proc based on the method. When the proc is called with a number of arguments that is lower than the method's arity, then another curried proc is returned. Only when enough arguments have been supplied to satisfy the method signature, will the method actually be called.

The optional *arity* argument should be supplied when currying methods with variable arguments to determine how many arguments are needed before the method is called.

```
def foo(a,b,c)
  [a, b, c]
end
proc = self.method(:foo).curry
proc2 = proc.call(1, 2)  #=> #<Proc>
proc2.call(3)  #=> [1,2,3]
def vararg(*args)
  args
end
proc = self.method(:vararg).curry(4)
proc2 = proc.call(:x)  #=> #<Proc>
```

# $_{\odot}$ eql?(other\_meth) → true or false $_{\odot}$ meth == other\_meth → true or false

Two method objects are equal if they are bound to the same object and refer to the same method definition and their owners are the same class or module.

# 

Returns a hash value corresponding to the method object.

See also Object#hash.

# output ou

#### $\odot$ inspect $\rightarrow$ string

Returns the name of the underlying method.



#### mame → symbol

Returns the name of the method.

#### $_{\odot}$ original\_name $\rightarrow$ symbol

Returns the original name of the method.

#### owner → class\_or\_module

Returns the class or module that defines the method.

#### ◎ parameters → array

Returns the parameter information of this method.

#### or receiver → object

Returns the bound receiver of the method object.

#### source\_location → [String, Fixnum]

Returns the Ruby source filename and line number containing this method or nil if this method was not defined in Ruby (i.e. native)

#### super\_method()

Returns a <u>Method</u> of superclass, which would be called when super is used.

#### $\odot$ to\_proc $\rightarrow$ proc

Returns a Proc object corresponding to this method.

#### oto\_s → string

#### inspect → string

Returns the name of the underlying method.



# on unbind → unbound\_method

Dissociates *meth* from its current receiver. The resulting UnboundMethod can subsequently be bound to a new object of the same class (see UnboundMethod).

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class Module

A Module is a collection of methods and constants. The methods in a module may be instance methods or module methods. Instance methods appear as methods in a class when the module is included, module methods do not. Conversely, module methods may be called without creating an encapsulating object, while instance methods may not. (See Module#module\_function.)

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



# In Files

- 📄 class.c
- 📄 eval.c
- 📄 load.c

object.c
 proc.c
 vm\_eval.c

vm\_method.c

# Parent

Object

# **Public Class Methods**

#### $\odot$ constants $\rightarrow$ array

#### $\odot$ constants(inherited) $\rightarrow$ array

In the first form, returns an array of the names of all constants accessible from the point of call. This list includes the names of all modules and classes defined in the global scope.



The second form calls the instance method constants.

#### $\odot$ nesting $\rightarrow$ array

Returns the list of Modules nested at the point of call.

```
module M1
module M2
$a = Module.nesting
```



# onew → mod onew {|mod| block } → mod

Creates a new anonymous module. If a block is given, it is passed the module object, and the block is evaluated in the context of this module using module\_eval.



Assign the module to a constant (name starting uppercase) if you want to treat it like a regular module.

# **Public Instance Methods**

# $_{\odot}$ mod < other $\rightarrow$ true, false, or nil

Returns true if *mod* is a subclass of *other*. Returns nil if there's no relationship between the two. (Think of the relationship in terms of the class definition: "class A<B" implies "A<B".)

#### $_{\odot}$ mod <= other $\rightarrow$ true, false, or nil

Returns true if *mod* is a subclass of *other* or is the same as *other*. Returns nil if there's no relationship between the two. (Think of the relationship in terms of the class definition: "class A<B" implies "A<B".)

#### module <=> other\_module → -1, 0, +1, or nil Comparison—Returns -1, 0, +1 or nil depending on whether module includes other\_module, they are the same, or if module is included by other\_module. This is the basis for the tests in Comparable.

Returns nil if module has no relationship with other\_module, if other\_module is not a module, or if the two values are incomparable.

#### $\bigcirc$ obj == other → true or false $\bigcirc$ equal?(other) → true or false $\bigcirc$ eql?(other) → true or false

Equality — At the object level, == returns true only if obj and other are the same object. Typically, this method is overridden in descendant classes to provide class-specific meaning.

Unlike ==, the equal? method should never be overridden by subclasses as it is used to determine object identity (that is, a.equal?(b) if and only if a is the same object as b):



The eq1? method returns true if obj and other refer to the same hash key. This is used by <u>Hash</u> to test members for equality. For objects of class object, eq1? is synonymous with ==. Subclasses normally continue this tradition by aliasing eq1? to their overridden == method, but there are exceptions. Numeric types, for example, perform type conversion across ==, but not across eq1?, so:

#### mod === obj → true or false

Case Equality—Returns true if *obj* is an instance of *mod* or one of *mod*'s descendants. Of limited use for modules, but can be used in case statements to classify objects by class.

#### $_{\odot}$ mod > other $\rightarrow$ true, false, or nil

Returns true if *mod* is an ancestor of *other*. Returns nil if there's no relationship between the two. (Think of the relationship in terms of the class definition: "class A<B" implies "B>A".)

#### $_{\odot}$ mod >= other $\rightarrow$ true, false, or nil

Returns true if *mod* is an ancestor of *other*, or the two modules are the same. Returns nil if there's no relationship between the two. (Think of the relationship in terms of the class definition: "class A<B" implies "B>A".)

#### ancestors → array
Returns a list of modules included/prepended in *mod* (including *mod* itself).



#### $_{\odot}$ autoload(module, filename) $\rightarrow$ nil

Registers *filename* to be loaded (using Kernel::require) the first time that *module* (which may be a string or a symbol) is accessed in the namespace of *mod*.



#### autoload?(name) → String or nil

Returns *filename* to be loaded if *name* is registered as autoload in the namespace of *mod*.



Evaluates the string or block in the context of *mod*, except that when a block is given, constant/class variable lookup is not affected. This can be used to add methods to a class. module\_eval returns the result of evaluating its argument. The optional *filename* and *lineno* parameters set the text for error messages.

```
class Thing
end
a = %q{def hello() "Hello there!" end}
Thing.module_eval(a)
puts Thing.new.hello()
Thing.module_eval("invalid code", "dummy", 123)
```

produces:



#### rightarrow module\_exec(arg...) {|var...| block } → obj rightarrow class\_exec(arg...) {|var...| block } → obj

Evaluates the given block in the context of the class/module. The method defined in the block will belong to the receiver. Any arguments passed to the method will be passed to the block. This can be used if the block needs to access instance variables.

```
class Thing
end
Thing.class_exec{
   def hello() "Hello there!" end
}
puts Thing.new.hello()
```

produces:

Hello there!

```
class_variable_defined?(symbol) \rightarrow true or _{\textcircled{}} false
```

class\_variable\_defined?(string)  $\rightarrow$  true or  $_{\textcircled{}}$  false

Returns true if the given class variable is defined in *obj.* String arguments are converted to symbols.



#### $_{\odot}$ class\_variable\_get(symbol) → obj $_{\odot}$ class\_variable\_get(string) → obj

Returns the value of the given class variable (or throws a NameError exception). The @@ part of the variable name should be included for regular class variables. <u>String</u> arguments are converted to symbols.



#### rightarrow class\_variable\_set(symbol, obj) → obj rightarrow class variable set(string, obj) → obj

Sets the class variable named by *symbol* to the given object. If the class variable name is passed as a string, that string is converted to a symbol.

```
class Fred
@@foo = 99
def foo
```



#### $\odot$ class\_variables(inherit=true) $\rightarrow$ array

Returns an array of the names of class variables in *mod*. This includes the names of class variables in any included modules, unless the *inherit* parameter is set to false.



# const\_defined?(sym, inherit=true) $\rightarrow$ true or $_{\textcircled{}}$ false

# const\_defined?(str, inherit=true) $\rightarrow$ true or $\otimes$ false

Says whether *mod* or its ancestors have a constant with the given name:



If *mod* is a Module, additionally Object and its ancestors are checked:

Math.const\_defined?(:String) #=> true, found i

•

In each of the checked classes or modules, if the constant is not present but there is an autoload for it, true is returned directly without autoloading:



If the constant is not found the callback const\_missing is **not** called and the method returns false.

If inherit is false, the lookup only checks the constants in the receiver:



In this case, the same logic for autoloading applies.

If the argument is not a valid constant name a NameError is raised with the message "wrong constant name *name*":



#### const\_get(sym, inherit=true) → obj const\_get(str, inherit=true) → obj

Checks for a constant with the given name in *mod*. If inherit is set, the lookup will also search the ancestors (and Object if *mod* is a Module).

The value of the constant is returned if a definition is found, otherwise a NameError is raised.



Þ

This method will recursively look up constant names if a namespaced class name is provided. For example:

```
module Foo; class Bar; end end
Object.const_get 'Foo::Bar'
```

The inherit flag is respected on each lookup. For example:



If the argument is not a valid constant name a NameError will be raised with a warning "wrong constant name".



#### const\_missing(sym) → obj

Invoked when a reference is made to an undefined constant in *mod*. It is passed a symbol for the undefined constant, and returns a value to be used for that constant. The following code is an example of the same:



In the next example when a reference is made to an undefined constant, it attempts to load a file whose name is the lowercase version of the constant (thus class Fred is assumed to be in file fred.rb). If found, it returns the loaded class. It therefore implements an autoload feature similar to Kernel#autoload and #autoload.



#### onst\_set(sym, obj) → obj const\_set(str, obj) → obj

. ◀1

Sets the named constant to the given object, returning that object. Creates a new constant if no constant with the given name previously existed.



If sym or str is not a valid constant name a NameError will be raised with a warning "wrong constant name".

С	bject.const_	_set('foobar',	42)		
•					<b>_</b> )

constants(inherit=true) → array

Returns an array of the names of the constants accessible in *mod*. This includes the names of constants in any included modules (example at start of section), unless the *inherit* parameter is set to false.



Also see Module::const\_defined?.

#### $_{\odot}$ freeze $\rightarrow$ mod

Prevents further modifications to mod.

This method returns self.

#### include(module, ...) → self

Invokes Module.append\_features on each parameter in reverse order.

#### $_{\odot}$ include?(module) $\rightarrow$ true or false

Returns true if *module* is included in *mod* or one of *mod*'s ancestors.

module A end	
class B	
include A	
end	
class C < B	
end	
<pre>B.include?(A)</pre>	
C.include?(A)	
A.include?(A)	

#### $_{\odot}$ included\_modules $\rightarrow$ array

Returns the list of modules included in mod.



(a) inspect()
Alias for: to\_s

# instance\_method(symbol) $\rightarrow$ $_{\odot}$ unbound\_method

Returns an UnboundMethod representing the given instance method in *mod*.

```
class Interpreter
  def do_a() print "there, "; end
  def do_d() print "Hello ";
                              end
  def do_e() print "!\n";
                              end
  def do_v() print "Dave"; end
  Dispatcher = {
    "a" => instance_method(:do_a),
    "d" => instance_method(:do_d),
    "e" => instance_method(:do_e),
    "v" => instance_method(:do_v)
  }
  def interpret(string)
    string.each_char {|b| Dispatcher[b].bind(self
  end
end
interpreter = Interpreter.new
interpreter.interpret('dave')
•
```

produces:



#### instance\_methods(include\_super=true) → ⊚ array

Returns an array containing the names of the public and protected instance methods in the receiver. For a module, these are the public and protected methods; for a class, they are the instance (not singleton) methods. If the optional parameter is false, the methods of any ancestors are not included.

```
module A
  def method1() end
end
class B
  include A
  def method2() end
end
class C < B
  def method3() end
end
A.instance_methods(false)
B.instance_methods(false)
B.instance_methods(true).include?(:method1)
C.instance_methods(false)
C.instance_methods.include?(:method2)
4
```

#### rightarrow method\_defined?(symbol) → true or false rightarrow method\_defined?(string) → true or false

Returns true if the named method is defined by *mod* (or its included modules and, if *mod* is a class, its ancestors). Public and protected methods are matched. <u>String</u> arguments are converted to symbols.

```
module A
  def method1() end
end
class B
  def method2() end
```

```
end
class C < B
    include A
    def method3() end
end
A.method_defined? :method1    #=> true
C.method_defined? "method1"    #=> true
C.method_defined? "method2"    #=> true
C.method_defined? "method3"    #=> true
C.method_defined? "method4"    #=> false
```

# class\_eval(string [, filename [, lineno]]) $\rightarrow$ $_{\textcircled{}}$ obj

#### $_{\odot}$ module\_eval {|| block } $\rightarrow$ obj

Evaluates the string or block in the context of *mod*, except that when a block is given, constant/class variable lookup is not affected. This can be used to add methods to a class. module\_eval returns the result of evaluating its argument. The optional *filename* and *lineno* parameters set the text for error messages.

```
class Thing
end
a = %q{def hello() "Hello there!" end}
Thing.module_eval(a)
puts Thing.new.hello()
Thing.module_eval("invalid code", "dummy", 123)
```

produces:



 $_{\odot}$  module\_exec(arg...) {|var...| block }  $_{\rightarrow}$  obj  $_{\odot}$  class\_exec(arg...) {|var...| block }  $_{\rightarrow}$  obj Evaluates the given block in the context of the class/module. The method defined in the block will belong to the receiver. Any arguments passed to the method will be passed to the block. This can be used if the block needs to access instance variables.

```
class Thing
end
Thing.class_exec{
   def hello() "Hello there!" end
}
puts Thing.new.hello()
```

produces:

Hello there!

#### $\otimes$ name $\rightarrow$ string

Returns the name of the module *mod*. Returns nil for anonymous modules.

#### $_{\odot}$ prepend(module, ...) → self

Invokes Module.prepend\_features on each parameter in reverse order.

#### $_{\odot}$ private\_class\_method(symbol, ...) $\rightarrow$ mod

#### $_{\odot}$ private\_class\_method(string, ...) $\rightarrow$ mod

Makes existing class methods private. Often used to hide the default constructor new.

String arguments are converted to symbols.

```
class SimpleSingleton # Not thread safe
private_class_method :new
def SimpleSingleton.create(*args, &block)
    @me = new(*args, &block) if ! @me
    @me
end
```

```
end
```

#### $_{\odot}$ private\_constant(symbol, ...) $\rightarrow$ mod

Makes a list of existing constants private.

# private\_instance\_methods(include\_super=true)

#### li → array

Returns a list of the private instance methods defined in *mod*. If the optional parameter is false, the methods of any ancestors are not included.



# private\_method\_defined?(symbol) $\rightarrow$ true or $_{\textcircled{}}$ false

#### private\_method\_defined?(string) $\rightarrow$ true or $\otimes$ false

Returns true if the named private method is defined by \_ mod\_ (or its included modules and, if *mod* is a class, its ancestors). <u>String</u> arguments are converted to symbols.

```
module A
  def method1() end
end
class B
  private
  def method2() end
end
class C < B
  include A</pre>
```



#### protected\_instance\_methods(include\_super=tri ⊚ → array

Returns a list of the protected instance methods defined in *mod*. If the optional parameter is false, the methods of any ancestors are not included.

# protected\_method\_defined?(symbol) $\rightarrow$ true $_{\textcircled{}}$ or false

# protected\_method\_defined?(string) $\rightarrow$ true $\odot$ or false

Returns true if the named protected method is defined by *mod* (or its included modules and, if *mod* is a class, its ancestors). <u>String</u> arguments are converted to symbols.

# $_{\odot}$ public\_class\_method(symbol, ...) → mod $_{\odot}$ public\_class\_method(string, ...) → mod

Makes a list of existing class methods public.

String arguments are converted to symbols.

#### $_{\odot}$ public\_constant(symbol, ...) $\rightarrow$ mod

Makes a list of existing constants public.

# public\_instance\_method(symbol) $\rightarrow$ $_{\odot}$ unbound\_method

Similar to *#instance\_method*, searches public method only.

#### public\_instance\_methods(include\_super=true) ⊚ → array

Returns a list of the public instance methods defined in *mod*. If the optional parameter is false, the methods of any ancestors are not included.

# public\_method\_defined?(symbol) $\rightarrow$ true or $\otimes$ false

#### public\_method\_defined?(string) $\rightarrow$ true or $\otimes$ false

Returns true if the named public method is defined by *mod* (or its included modules and, if *mod* is a class, its ancestors). <u>String</u> arguments are converted to symbols.

```
module A
   def method1() end
end
```

```
class B
  protected
  def method2() end
end
class C < B
  include A
  def method3() end
end
A.method_defined? :method1  #=> true
C.public_method_defined? "method1" #=> true
C.public_method_defined? "method2" #=> true
C.method_defined? "method2" #=> true
```

#### $\odot$ remove\_class\_variable(sym) $\rightarrow$ obj

Removes the definition of the *sym*, returning that constant's value.

```
class Dummy
  @@var = 99
  puts @@var
  remove_class_variable(:@@var)
  p(defined? @@var)
end
```

produces:

99 nil

#### singleton\_class? → true or false

Returns true if *mod* is a singleton class or false if it is an ordinary class or module.



#### os to\_s → string

Returns a string representing this module or class. For basic classes and modules, this is the name. For singletons, we show information on the thing we're attached to as well.

Also aliased as: inspect

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# **class Mutex**

Mutex implements a simple semaphore that can be used to coordinate access to shared data from multiple concurrent threads.

Example:



## In Files

thread.c

### Parent

Object

## **Public Class Methods**

#### mew → mutex

Creates a new Mutex

#### **Public Instance Methods**

#### lock → self

Attempts to grab the lock and waits if it isn't available. Raises ThreadError if mutex was locked by the current thread.

#### $\odot$ locked? $\rightarrow$ true or false

Returns true if this lock is currently held by some thread.

#### $\odot$ owned? $\rightarrow$ true or false

Returns true if this lock is currently held by current thread. *This API is experimental, and subject to change.* 

#### $_{\odot}$ sleep(timeout = nil) $\rightarrow$ number

Releases the lock and sleeps timeout seconds if it is given and non-nil or forever. Raises ThreadError if mutex wasn't locked by the current thread.

When the thread is next woken up, it will attempt to reacquire the lock.

Note that this method can wakeup without explicit <u>Thread#wakeup</u> call. For example, receiving signal and so on.

#### 𝔅 synchronize { ... } → result of the block

Obtains a lock, runs the block, and releases the lock when the block completes. See the example under Mutex.

#### $_{\odot}$ try\_lock $\rightarrow$ true or false

Attempts to obtain the lock and returns immediately. Returns true if the lock was granted.

#### $\odot$ unlock $\rightarrow$ self

Releases the lock. Raises ThreadError if mutex wasn't locked by the current thread.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class NameError

Raised when a given name is invalid or undefined.

puts foo
raises the exception:
NameError: undefined local variable or method

Since constant names must start with a capital:

Fixnum.const\_set :answer, 42

raises the exception:

NameError: wrong constant name answer

In Files

error.c

Parent

StandardError

**Public Class Methods** 

#### $_{\odot}$ new(msg [, name]) $\rightarrow$ name\_error

Construct a new <u>NameError</u> exception. If given the *name* parameter may subsequently be examined using the NameError.name method.

### **Public Instance Methods**

#### $_{\odot}$ name $\rightarrow$ string or nil

Return the name associated with this <u>NameError</u> exception.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class NilClass

The class of the singleton object nil.

# In Files

- complex.c
- object.c
- 📄 rational.c

### Parent

Object

### **Public Instance Methods**

#### false & obj → false

#### obj → false

And—Returns false. *obj* is always evaluated as it is the argument to a method call—there is no short-circuit evaluation in this case.

#### $_{\odot}$ false ^ obj → true or false $_{\odot}$ nil ^ obj → true or false

Exclusive Or—If *obj* is nil or false, returns false; otherwise, returns true.

#### inspect → "nil"

Always returns the string "nil".

#### inil? → true

Only the object nil responds true to nil?.

#### $_{\odot}$ rationalize([eps]) $\rightarrow$ (0/1)

Returns zero as a rational. The optional argument eps is always ignored.

#### ⊚ to\_a → []

Always returns an empty array.

### ⊚ to\_c → (0+0i)

Returns zero as a complex.

#### $\odot$ to\_f $\rightarrow$ 0.0

Always returns zero.

#### $\odot$ to\_h $\rightarrow$ {}

Always returns an empty hash.

nil.to\_h #=> {}

#### rightarrow to\_i $\rightarrow$ 0

Always returns zero.



#### $oldsymbol{black}$ to\_r $\rightarrow$ (0/1) Returns zero as a rational.

● **to\_s** → **""** Always returns the empty string.

#### $_{\odot}$ false | obj → true or false $_{\odot}$ nil | obj → true or false

Or—Returns false if *obj* is nil or false; true otherwise.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class NoMemoryError

Raised when memory allocation fails.

**In Files** 

error.c

### Parent

Exception

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class NoMethodError

Raised when a method is called on a receiver which doesn't have it defined and also fails to respond with method\_missing.

```
"hello".to_ary
```

raises the exception:



### **In Files**

error.c

Parent

NameError

## **Public Class Methods**

Some provide the second struct a NoMethodError exception for a method of the given name called with the given arguments. The name may be accessed using the #name method on the resulting object, and the arguments using the #args method.

## **Public Instance Methods**

#### obj args → obj

Return the arguments passed in as the third parameter to the constructor.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class NotImplementedError

Raised when a feature is not implemented on the current platform. For example, methods depending on the fsync or fork system calls may raise this exception if the underlying operating system or Ruby runtime does not support them.

Note that if fork raises a NotImplementedError, then respond\_to?(:fork) returns false.

### **In Files**

error.c

#### Parent

ScriptError

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class Numeric

The top-level number class.

# In Files

- complex.c
- 📄 numeric.c
- 📄 rational.c

# Parent

**Object** 

## **Included Modules**

Comparable

## **Public Instance Methods**

modulo(numeric) → real

x.modulo(y) means  $x-y^*(x/y)$ .floor

Equivalent to num.divmod(numeric)[1].

See #divmod.

#### 

Unary Plus-Returns the receiver's value.

#### onum → numeric

Unary Minus—Returns the receiver's value, negated.

#### $_{\odot}$ number <=> other $\rightarrow$ 0 or nil

Returns zero if number equals other, otherwise nil is returned if the two values are incomparable.

#### abs → numeric

#### magnitude → numeric

Returns the absolute value of num.

12.abs	
(-34.56).abs	
-34.56.abs	

#magnitude is an alias of #abs.

#### $\odot$ abs2 $\rightarrow$ real

Returns square of self.

- rightarrow arg  $\rightarrow$  0 or float
- $_{\odot}$  angle  $\rightarrow$  0 or float

#### $\odot$ phase $\rightarrow$ 0 or float

Returns 0 if the value is positive, pi otherwise.

#### $\odot$ arg $\rightarrow$ 0 or float

 $_{\odot}$  angle  $\rightarrow$  0 or float

#### $rightarrow phase \rightarrow 0 \text{ or float}$

Returns 0 if the value is positive, pi otherwise.

 $\odot$  ceil  $\rightarrow$  integer

Returns the smallest possible Integer that is greater than or equal to num.

Numeric achieves this by converting itself to a Float then invoking Float#ceil.



#### $\odot$ coerce(numeric) $\rightarrow$ array

If a +numeric is the same type as num, returns an array containing numeric and num. Otherwise, returns an array with both a numeric and num represented as Float objects.

This coercion mechanism is used by Ruby to handle mixed-type numeric operations: it is intended to find a compatible common type between the two operands of the operator.

1.coerce(2.5) 1.2.coerce(3) 1.coerce(2)	#=> [2.5, #=> [3.0, #=> [2, 1]	1.0] 1.2]	
<ul> <li>conj → self</li> <li>conjugate →</li> <li>Returns self.</li> </ul>	self		
<ul> <li>conj → self</li> <li>conjugate →</li> <li>Returns self.</li> </ul>	self		
🚳 denominator	→ integer		

Returns the denominator (always positive).

#### oiv(numeric) → integer

Uses / to perform division, then converts the result to an integer. numeric does not define the / operator; this is left to subclasses.

Equivalent to num.divmod(numeric)[0].

See #divmod.

#### o divmod(numeric) → array

Returns an array containing the quotient and modulus obtained by dividing num by numeric.

If q, r = \* x.divmod(y), then

q = floor(x/y)
x = q\*y+r

The quotient is rounded toward -infinity, as shown in the following table:

a	b	a.di\	/mod(b)	a/b	a.modulo(
13	+   4	3, +	1	3	1
13	+   -4	-4,	-3	-4	-3
-13	+   4	-4,	3	-4	3
-13	+   -4	3, +	-1	3	-1
11.5	+   4	2,	3.5	2.8	75   3.5
11.5	+   -4		-0.5	-2.8	75   -0.5
-11.5	+   4		0.5	-2.8	75   0.5
-11.5	-4	2,	-3.5	2.8	75   -3.5
[▲]					

Examples

11.divmod(3)	
11.divmod(-3)	
11.divmod(3.5)	
(-11).divmod(3.5)	
(11.5).divmod(3.5)	

#### $_{\odot}$ eql?(numeric) $\rightarrow$ true or false

Returns true if num and numeric are the same type and have equal values.



### fdiv(numeric) → float

Returns float division.

#### floor → integer

Returns the largest integer less than or equal to num.

Numeric implements this by converting an Integer to a Float and invoking Float#floor.



### i → Complex(0,num)

Returns the corresponding imaginary number. Not available for complex numbers.

imag → 0
 imaginary → 0
 Returns zero.

rightarrow imag → 0 rightarrow imaginary → 0Returns zero.

#### initialize\_copy(p1)

Numerics are immutable values, which should not be copied.

Any attempt to use this method on a <u>Numeric</u> will raise a <u>TypeError</u>.

#### $_{\odot}$ integer? $\rightarrow$ true or false

Returns true if num is an <u>Integer</u> (including <u>Fixnum</u> and Bignum).

(1.0).integer? #=> false
(1).integer? #=> true

#### $_{\odot}$ abs $\rightarrow$ numeric

#### magnitude → numeric

Returns the absolute value of num.



#magnitude is an alias of #abs.

#### modulo(numeric) → real

x.modulo(y) means x-y\*(x/y).floor

Equivalent to num.divmod(numeric)[1].

See #divmod.

#### on nonzero? → self or nil

Returns self if num is not zero, nil otherwise.

This behavior is useful when chaining comparisons:



mumerator → integer

Returns the numerator.

#### rightarrow arg $\rightarrow$ 0 or float

#### $_{\odot}$ angle $\rightarrow$ 0 or float

#### $\odot$ phase $\rightarrow$ 0 or float

Returns 0 if the value is positive, pi otherwise.

#### polar → array

Returns an array; [num.abs, num.arg].

#### rightarrow quo(int\_or\_rat) → rat rightarrow quo(flo) → flo

Returns most exact division (rational for integers, float for floats).

real → self Returns self.

#### $real? \rightarrow$ true or false
Returns true if num is a Real number. (i.e. not Complex).

import → array
 import → array
 import → array
 Returns an array; [num, 0].

rect → array rectangular → arrayReturns an array; [num, 0].

```
oremainder(numeric) → real
```

x.remainder(y) means x-y\*(x/y).truncate

See #divmod.

#### $\odot$ round([ndigits]) $\rightarrow$ integer or float

Rounds num to a given precision in decimal digits (default 0 digits).

Precision may be negative. Returns a floating point number when ndigits is more than zero.

Numeric implements this by converting itself to a Float and invoking Float#round.

#### singleton\_method\_added(p1)

Trap attempts to add methods to <u>Numeric</u> objects. Always raises a <u>TypeError</u>.

Numerics should be values; singleton\_methods should not be added to them.

- $step(by: step, to: limit) {|i| block } → self$
- $\otimes$  step(by: step, to: limit)  $\rightarrow$  an\_enumerator
- $_{\odot}$  step(limit=nil, step=1) {|i| block } → self
- $\otimes$  step(limit=nil, step=1)  $\rightarrow$  an\_enumerator

Invokes the given block with the sequence of numbers starting at num, incremented by step (defaulted to 1) on each call.

The loop finishes when the value to be passed to the block is greater than limit (if step is positive) or less than limit (if step is negative), where *limit* is defaulted to infinity.

In the recommended keyword argument style, either or both of step and limit (default infinity) can be omitted. In the fixed position argument style, zero as a step (i.e. num.step(limit, 0)) is not allowed for historical compatibility reasons.

If all the arguments are integers, the loop operates using an integer counter.

If any of the arguments are floating point numbers, all are converted to floats, and the loop is executed the following expression:

```
floor(n + n*epsilon)+ 1
```

Where the n is the following:

```
n = (limit - num)/step
```

Otherwise, the loop starts at num, uses either the lessthan (<) or greater-than (>) operator to compare the counter against limit, and increments itself using the + operator.

If no block is given, an Enumerator is returned instead.

For example:

3.step(to: 5) {  i  print i, " 1.step(10, 2) {  i  print i, " Math::E.step(to: Math::PI, by:	"} "} 0.2){	f  print ·
<pre>p 1.step.take(4) p 10.step(by: -1).take(4)</pre>		

Will produce:



#### $\odot$ to\_c $\rightarrow$ complex

Returns the value as a complex.

#### oto\_int → integer

Invokes the child class's to\_i method to convert num to an integer.

1.0.class => Float
1.0.to\_int.class => Fixnum
1.0.to\_i.class => Fixnum

#### truncate → integer integer

Returns num truncated to an Integer.

Numeric implements this by converting its value to a Float and invoking Float#truncate.

#### sero? → true or false

Returns true if num has a zero value.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class Object

Object is the default root of all Ruby objects. Object inherits from BasicObject which allows creating alternate object hierarchies. Methods on Object are available to all classes unless explicitly overridden.

Object mixes in the Kernel module, making the built-in kernel functions globally accessible. Although the instance methods of Object are defined by the Kernel module, we have chosen to document them here for clarity.

When referencing constants in classes inheriting from <u>Object</u> you do not need to use the full namespace. For example, referencing File inside YourClass will find the top-level <u>File</u> class.

In the descriptions of Object's methods, the parameter *symbol* refers to a symbol, which is either a quoted string or a <u>Symbol</u> (such as :name).

# **In Files**

- 📄 class.c
- enumerator.c
- 📄 eval.c
- 📄 gc.c
- 📄 hash.c

- 📄 io.c
- object.c
- parse.c
- proc.c
- l ruby.c
- version.c
- 📄 vm.c
- vm\_eval.c
- vm\_method.c

# Parent

**BasicObject** 

# **Included Modules**

Kernel

# Constants

#### ARGF

<u>ARGF</u> is a stream designed for use in scripts that process files given as command-line arguments or passed in via STDIN.

See <u>ARGF</u> (the class) for more details.

#### ARGV

<u>ARGV</u> contains the command line arguments used to run ruby with the first value containing the name of the executable.

A library like OptionParser can be used to process command-line arguments.

#### DATA

DATA is a File that contains the data section of the executed file. To create a data section use \_\_END\_\_:

```
$ cat t.rb
puts DATA.gets
__END__
hello world!
$ ruby t.rb
```

hello world!

#### ENV

ENV is a Hash-like accessor for environment variables.

See ENV (the class) for more details.

#### FALSE

An alias of false

#### NIL

An alias of nil

#### **RUBY\_COPYRIGHT**

The copyright string for ruby

#### **RUBY\_DESCRIPTION**

The full ruby version string, like ruby -v prints'

#### **RUBY\_ENGINE**

The engine or interpreter this ruby uses.

#### **RUBY\_PATCHLEVEL**

The patchlevel for this ruby. If this is a development build of ruby the patchlevel will be -1

#### **RUBY\_PLATFORM**

The platform for this ruby

#### **RUBY\_RELEASE\_DATE**

The date this ruby was released

#### **RUBY\_REVISION**

The SVN revision for this ruby.

#### **RUBY\_VERSION**

The running version of ruby

#### SCRIPT\_LINES\_\_

When a <u>Hash</u> is assigned to SCRIPT\_LINES\_\_ the contents of files loaded after the assignment will be added as an <u>Array</u> of lines with the file name as the key.

#### STDERR

Holds the original stderr

#### STDIN

Holds the original stdin

#### STDOUT

Holds the original stdout

#### **TOPLEVEL\_BINDING**

The Binding of the top level scope

#### TRUE

An alias of true

# **Public Instance Methods**

#### $\odot$ obj !~ other $\rightarrow$ true or false

Returns true if two objects do not match (using the  $=\sim$  method), otherwise false.

#### $_{\odot}$ obj <=> other $\rightarrow$ 0 or nil

Returns 0 if obj and other are the same object or obj == other, otherwise nil.

The <=> is used by various methods to compare objects, for example <u>Enumerable#sort</u>, Enumerable#max etc.

Your implementation of <=> should return one of the following values: -1, 0, 1 or nil. -1 means self is smaller than other. 0 means self is equal to other. 1 means self is bigger than other. Nil means the two values could not be compared.

When you define <=>, you can include <u>Comparable</u> to gain the methods <=, <, ==, >=, > and between?.

#### $_{\odot}$ obj === other $\rightarrow$ true or false

Case Equality – For class <u>Object</u>, effectively the same as calling #==, but typically overridden by descendants to provide meaningful semantics in case statements.

obj =~ other → nil

Pattern Match—Overridden by descendants (notably Regexp and String) to provide meaningful patternmatch semantics.

#### on class → class

Returns the class of *obj*. This method must always be called with an explicit receiver, as class is also a reserved word in Ruby.

1.class #=> Fixnum self.class #=> Object

#### os clone → an\_object

Produces a shallow copy of *obj*—the instance variables of *obj* are copied, but not the objects they reference. clone copies the frozen and tainted state of *obj*. See also the discussion under <code>object#dup</code>.



This method may have class-specific behavior. If so, that behavior will be documented under the #initialize\_copy method of the class.

#### → proc

Defines a singleton method in the receiver. The *method* parameter can be a Proc, a Method or an UnboundMethod object. If a block is specified, it is used as the method body.



## $_{\odot}$ display(port=\$>) $\rightarrow$ nil

Prints *obj* on the given port (default \$>). Equivalent to:

```
def display(port=$>)
   port.write self
end
```

For example:

```
1.display
"cat".display
[ 4, 5, 6 ].display
puts
```

produces:

1cat456

#### o dup → an\_object

Produces a shallow copy of *obj*—the instance variables of *obj* are copied, but not the objects they reference. dup copies the tainted state of *obj*.

This method may have class-specific behavior. If so, that behavior will be documented under the #initialize\_copy method of the class.

#### on dup vs clone

In general, clone and dup may have different semantics in descendant classes. While clone is used to duplicate an object, including its internal state, dup typically uses the class of the descendant object to create the new instance.

When using <u>dup</u>, any modules that the object has been extended with will not be copied.

```
class Klass
  attr_accessor :str
end
module Foo
  def foo; 'foo'; end
end
s1 = Klass.new #=> #<Klass:0x401b3a38>
s1.extend(Foo) #=> #<Klass:0x401b3a38>
s1.foo #=> "foo"
s2 = s1.clone #=> #<Klass:0x401b3a38>
s2.foo #=> "foo"
s3 = s1.dup #=> #<Klass:0x401b3a38>
s3.foo #=> NoMethodError: undefined method `foo'
```

 $\odot$  to\_enum(method = :each, \*args)  $\rightarrow$  enum

- enum\_for(method = :each, \*args) → enum to\_enum(method = :each, \*args) {|\*args|
   block} → enum
   enum\_for(method = :each\_\*args){|\*args|

Creates a new Enumerator which will enumerate by calling method on obj, passing args if any.

If a block is given, it will be used to calculate the size of the enumerator without the need to iterate it (see Enumerator#size).

#### Examples



It is typical to call <u>#to\_enum</u> when defining methods for a generic Enumerable, in case no block is passed.

Here is such an example, with parameter passing and a sizing block:

```
module Enumerable
# a generic method to repeat the values of any
def repeat(n)
raise ArgumentError, "#{n} is negative!" if r
unless block_given?
return to_enum(__method__, n) do # __method
sz = size  # Call size and multiply by
sz * n if sz # but return nil if size if
end
```

```
end
each do |*val|
n.times { yield *val }
end
end
%[hello world].repeat(2) { |w| puts w }
# => Prints 'hello', 'hello', 'world', 'world'
enum = (1..14).repeat(3)
# => returns an Enumerator when called without
enum.first(4) # => [1, 1, 1, 2]
enum.size # => 42
```

# obj == other → true or false<math> option equal?(other) → true or false

#### $_{\odot}$ eql?(other) $\rightarrow$ true or false

Equality — At the object level, == returns true only if obj and other are the same object. Typically, this method is overridden in descendant classes to provide class-specific meaning.

Unlike ==, the equal? method should never be overridden by subclasses as it is used to determine object identity (that is, a.equal?(b) if and only if a is the same object as b):



The eq1? method returns true if obj and other refer to the same hash key. This is used by <u>Hash</u> to test members for equality. For objects of class object, eq1? is synonymous with ==. Subclasses normally continue this tradition by aliasing eq1? to their overridden == method, but there are exceptions.
Numeric types, for example, perform type conversion
across ==, but not across eql?, so:



## ⊚ extend(module, ...) → obj

Adds to *obj* the instance methods from each module given as a parameter.

<pre>module Mod     def hello         "Hello from     end end</pre>	Mod.\n"
class Klass def hello "Hello from end end	Klass.\n"
<pre>k = Klass.new k.hello k.extend(Mod) k.hello</pre>	

## $\odot$ freeze $\rightarrow$ obj

Prevents further modifications to *obj*. A RuntimeError will be raised if modification is attempted. There is no way to unfreeze a frozen object. See also Object#frozen?.

This method returns self.

```
a = [ "a", "b", "c" ]
a.freeze
a << "z"
```

produces:

prog.rb:3:in `<<': can't modify frozen Array (Rur from prog.rb:3

Objects of the following classes are always frozen: Fixnum, Bignum, Float, Symbol.

## frozen? → true or false

Returns the freeze status of obj.



## hash → fixnum

Generates a Fixnum hash value for this object. This function must have the property that a.eq1?(b) implies a.hash == b.hash.

The hash value is used along with <u>eql?</u> by the <u>Hash</u> class to determine if two objects reference the same hash key. Any hash value that exceeds the capacity of a <u>Fixnum</u> will be truncated before being used.

The hash value for an object may not be identical across invocations or implementations of Ruby. If you need a stable identifier across Ruby invocations and implementations you will need to generate one with a custom method.

## inspect → string

Returns a string containing a human-readable representation of *obj*. The default inspect shows the object's class name, an encoding of the object id, and a list of the instance variables and their values (by calling <u>inspect</u> on each of them). User defined classes should override this method to provide a better representation of *obj*. When overriding this method, it should return a string whose encoding is compatible with the default external encoding.



## $_{\odot}$ instance\_of?(class) $\rightarrow$ true or false

Returns true if *obj* is an instance of the given class. See also <code>Object#kind\_of?</code>.

class A; end class B < A; end class C < B; end	
b = B.new	
b.instance_of? A	
b.instance_of? B	
b.instance_of? C	

instance\_variable\_defined?(symbol)  $\rightarrow$  true  $\odot$  or false

instance\_variable\_defined?(string)  $\rightarrow$  true or  $\otimes$  false

Returns true if the given instance variable is defined

in *obj.* String arguments are converted to symbols.



## $_{\odot}$ instance\_variable\_get(symbol) → obj $_{\odot}$ instance\_variable\_get(string) → obj

Returns the value of the given instance variable, or nil if the instance variable is not set. The @ part of the variable name should be included for regular instance variables. Throws a NameError exception if the supplied symbol is not valid as an instance variable name. String arguments are converted to symbols.

```
class Fred
  def initialize(p1, p2)
    @a, @b = p1, p2
  end
end
fred = Fred.new('cat', 99)
fred.instance_variable_get(:@a) #=> "cat"
fred.instance_variable_get("@b") #=> 99
```

### ⊗ instance\_variable\_set(symbol, obj) → obj ⊗ instance variable set(string, obj) → obj

Sets the instance variable named by *symbol* to the given object, thereby frustrating the efforts of the class's author to attempt to provide proper encapsulation. The variable does not have to exist prior to this call. If the instance variable name is

passed as a string, that string is converted to a symbol.



#### instance\_variables → array

Returns an array of instance variable names for the receiver. Note that simply defining an accessor does not create the corresponding instance variable.



## $_{\odot}$ is\_a?(class) → true or false $_{\odot}$ kind\_of?(class) → true or false

Returns true if *class* is the class of *obj*, or if *class* is one of the superclasses of *obj* or modules included in *obj*.

<pre>module M; class A include M end</pre>	end
<pre>class B &lt; A;</pre>	end
class C < B;	end

b = B.new	
b.is_a? A	
b.is_a? B	
b.is_a? C	
b.is_a? M	
D.KING_OT? A	
b.kind_of? A b.kind_of? B	
b.kind_of? A b.kind_of? B b.kind_of? C	
b.kind_of? A b.kind_of? B b.kind_of? C b.kind_of? M	

## itself → an\_object

Returns obj.



## $_{\odot}$ is\_a?(class) → true or false $_{\odot}$ kind\_of?(class) → true or false

Returns true if *class* is the class of *obj*, or if *class* is one of the superclasses of *obj* or modules included in *obj*.

<pre>module M; end class A     include M end</pre>	
class B < A; end	
class C < B; end	
<pre>b = B.new b.is_a? A b.is_a? B b.is_a? C b.is_a? M</pre>	
b.kind_of? A b.kind_of? B b.kind_of? C b.kind_of? M	

#### method(sym) → method

Looks up the named method as a receiver in *obj*, returning a Method object (or raising NameError). The Method object acts as a closure in *obj*'s object instance, so instance variables and the value of self remain available.

```
class Demo
  def initialize(n)
    @iv = n
  end
  def hello()
    "Hello, @iv = #{@iv}"
  end
end
k = Demo.new(99)
m = k.method(:hello)
m.call #=> "Hello, @iv = 99"
l = Demo.new('Fred')
m = l.method("hello")
m.call #=> "Hello, @iv = Fred"
```

#### $_{\odot}$ methods(regular=true) $\rightarrow$ array

Returns a list of the names of public and protected methods of *obj*. This will include all the methods accessible in *obj*'s ancestors. If the optional parameter is false, it returns an array of *obj<i>'s public and protected singleton methods, the array will not include methods in modules included in <i>obj.* 



<pre># :hash, :&lt;=&gt;, :class, :si k.methods.length #=&gt; 56</pre>	
<pre>k.methods(false) #=&gt; [] def k.singleton_method; end k.methods(false) #=&gt; [:singleton_method]</pre>	
<pre>module M123; def m123; end end k.extend M123</pre>	
<pre>k.methods(false) #=&gt; [:singleton_method]</pre>	
	Þ

#### inil? → true or false

Only the object *nil* responds true to nil?.



# id\_\_ → integer id\_\_ integer

# object\_id → integer

Returns an integer identifier for obj.

The same number will be returned on all calls to object\_id for a given object, and no two active objects will share an id.

Note: that some objects of builtin classes are reused for optimization. This is the case for immediate values and frozen string literals.

Immediate values are not passed by reference but are passed by value: nil, true, false, Fixnums, Symbols, and some Floats.



#### ◎ private\_methods(all=true) → array

Returns the list of private methods accessible to *obj*. If the *all* parameter is set to false, only those methods in the receiver will be listed.

#### $_{\odot}$ protected\_methods(all=true) $\rightarrow$ array

Returns the list of protected methods accessible to *obj*. If the *all* parameter is set to false, only those methods in the receiver will be listed.

#### ⊚ public\_method(sym) → method

Similar to method, searches public method only.

## ● **public\_methods(all=true)** $\rightarrow$ **array** Returns the list of public methods accessible to *obj*. If

the *all* parameter is set to false, only those methods in the receiver will be listed.

# public\_send(symbol [, args...]) → obj public\_send(string [, args...]) → obj

Invokes the method identified by *symbol*, passing it any arguments specified. Unlike send, <u>#public\_send</u> calls public methods only. When the method is identified by a string, the string is converted to a symbol.

# 1.public\_send(:puts, "hello") # causes NoMethodE

## $_{\odot}$ remove\_instance\_variable(symbol) $\rightarrow$ obj

Removes the named instance variable from *obj*, returning that variable's value.



# respond\_to?(symbol, include\_all=false) $\rightarrow$ $_{\textcircled{}}$ true or false

# respond\_to?(string, include\_all=false) $\rightarrow$ $_{\odot}$ true or false

Returns true if *obj* responds to the given method. Private and protected methods are included in the search only if the optional second parameter evaluates to true.

If the method is not implemented, as <u>Process.fork</u> on Windows, <u>File.lchmod</u> on GNU/Linux, etc., false is returned.

If the method is not defined, respond\_to\_missing? method is called and the result is returned.

When the method name parameter is given as a string, the string is converted to a symbol.

respond\_to\_missing?(string, include\_all)  $\rightarrow$   $_{\textcircled{}}$  true or false

DO NOT USE THIS DIRECTLY.

Hook method to return whether the *obj* can respond to *id* method or not.

When the method name parameter is given as a string, the string is converted to a symbol.

See <u>respond\_to?</u>, and the example of <u>BasicObject</u>.

```
Send(symbol [, args...]) → obj
```

```
\odot __send_(symbol [, args...]) \rightarrow obj
```

```
\odot send(string [, args...]) \rightarrow obj
```

```
\odot __send_(string [, args...]) \rightarrow obj
```

Invokes the method identified by *symbol*, passing it any arguments specified. You can use <u>\_\_send\_\_</u> if the name send clashes with an existing method in *obj*. When the method is identified by a string, the string is converted to a symbol.



#### singleton\_class → class

Returns the singleton class of *obj*. This method creates a new singleton class if *obj* does not have one.

If *obj* is nil, true, or false, it returns <u>NilClass</u>, <u>TrueClass</u>, or <u>FalseClass</u>, respectively. If *obj* is a Fixnum or a Symbol, it raises a TypeError.

```
Object.new.singleton_class#=> #<Class:#<Object</th>String.singleton_class#=> #<Class:String>nil.singleton_class#=> NilClass
```

#### singleton\_method(sym) → method

4

Similar to *method*, searches singleton method only.

►



#### $_{\odot}$ singleton\_methods(all=true) $\rightarrow$ array

Returns an array of the names of singleton methods for *obj*. If the optional *all* parameter is true, the list will include methods in modules included in *obj*. Only public and protected singleton methods are returned.

```
module Other
  def three() end
end

class Single
  def Single.four() end
end

a = Single.new

def a.one()
end

class << a</pre>
```



## taint → obj

Mark the object as tainted.

Objects that are marked as tainted will be restricted from various built-in methods. This is to prevent insecure data, such as command-line arguments or strings read from <u>Kernel#gets</u>, from inadvertently compromising the user's system.

To check whether an object is tainted, use tainted?.

You should only untaint a tainted object if your code has inspected it and determined that it is safe. To do so use untaint.

In \$SAFE level 3, all newly created objects are tainted and you can't untaint objects.

## tainted? → true or false

Returns true if the object is tainted.

See taint for more information.

## obj tap{|x|...} → obj

Yields self to the block, and then returns self. The primary purpose of this method is to "tap into" a method chain, in order to perform operations on intermediate results within the chain.



- $_{\odot}$  to\_enum(method = :each, \*args) → enum
- enum\_for(method = :each, \*args) → enum to\_enum(method = :each, \*args) {|\*args|
- block} → enum
   enum\_for(method = :each, \*args){|\*args|
   block} → enum

Creates a new Enumerator which will enumerate by calling method on obj, passing args if any.

If a block is given, it will be used to calculate the size of the enumerator without the need to iterate it (see Enumerator#size).

## Examples



It is typical to call <u>**#to\_enum</u>** when defining methods for a generic **Enumerable**, in case no block is passed.</u>

Here is such an example, with parameter passing and a sizing block:

```
module Enumerable
  def repeat(n)
    raise ArgumentError, "#{n} is negative!" if
    unless block_given?
      return to_enum(__method__, n) do
        sz = size
        sz * n if sz
      end
    end
    each do |*val
      n.times { yield *val }
    end
  end
end
%[hello world].repeat(2) { |w| puts w }
enum = (1..14).repeat(3)
enum.first(4)
enum.size
4
```

#### os to\_s → string

Returns a string representing *obj*. The default to\_s prints the object's class and an encoding of the object id. As a special case, the top-level object that is the initial execution context of Ruby programs returns "main".

## ⊚trust → obj

Deprecated method that is equivalent to untaint.

## obj untaint → obj

Removes the tainted mark from the object.

See taint for more information.

#### ⊚ untrust → obj

Deprecated method that is equivalent to taint.

### ⊚ untrusted? → true or false

Deprecated method that is equivalent to tainted?.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# module ObjectSpace

The <u>ObjectSpace</u> module contains a number of routines that interact with the garbage collection facility and allow you to traverse all living objects with an iterator.

ObjectSpace also provides support for object finalizers, procs that will be called when a specific object is about to be destroyed by garbage collection.



produces:

Finalizer two on 537763470 Finalizer one on 537763480

# **In Files**

📄 gc.c

# **Public Class Methods**

## $_{\odot}_id2ref(object_id) \rightarrow an_object$

Converts an object id to a reference to the object. May not be called on an object id passed as a parameter to a finalizer.



#### $\odot$ count\_objects([result\_hash]) $\rightarrow$ hash

Counts objects for each type.

It returns a hash, such as:



The contents of the returned hash are implementation specific. It may be changed in future.

If the optional argument result\_hash is given, it is overwritten and returned. This is intended to avoid probe effect.

This method is only expected to work on C Ruby.

## define\_finalizer(obj, aProc=proc())

Adds *aProc* as a finalizer, to be called after *obj* was destroyed.

## each\_object([module]) {|obj| ... } $\rightarrow$ fixnum each\_object([module]) $\rightarrow$ an\_enumerator

Calls the block once for each living, nonimmediate object in this Ruby process. If *module* is specified, calls the block for only those classes or modules that match (or are a subclass of) *module*. Returns the number of objects found. Immediate objects (Fixnums, SymbolS true, false, and nil) are never returned. In the example below, each\_object returns both the numbers we defined and several constants defined in the Math module.

If no block is given, an enumerator is returned instead.



produces:

```
12345678987654321
102.7
2.71828182845905
3.14159265358979
2.22044604925031e-16
1.7976931348623157e+308
2.2250738585072e-308
Total count: 7
```

 start → nil
 garbage\_collect → nil start(full\_mark: true, immediate\_sweep:
 true) → nil garbage\_collect(full\_mark: true,
 immediate\_sweep: true) → nil Initiates garbage collection, unless manually disabled. This method is defined with keyword arguments that default to true:



Use full\_mark: false to perform a minor <u>GC</u>. Use immediate\_sweep: false to defer sweeping (use lazy sweep).

Note: These keyword arguments are implementation and version dependent. They are not guaranteed to be future-compatible, and may be ignored if the underlying implementation does not support them.

## 🐵 undefine\_finalizer(obj)

Removes all finalizers for obj.

#### Generated by <u>RDoc</u> 3.12.2.

Generated with the Darkfish Rdoc Generator 3.

# class ObjectSpace::WeakMap

An <u>ObjectSpace::WeakMap</u> object holds references to any objects, but those objects can get garbage collected.

This class is mostly used internally by WeakRef, please use lib/weakref.rb for the public interface.

# In Files

📄 gc.c

# Parent

**Object** 

# **Included Modules**

Enumerable

# **Public Instance Methods**

# 🏐 **[](p1)**

Retrieves a weakly referenced object with the given key

## 🏐 []=(p1, p2)

Creates a weak reference from the given key to the given value

#### 🐵 each()

Iterates over keys and objects in a weakly referenced object

## @ each\_key()

Iterates over keys and objects in a weakly referenced object

## @ each\_pair()

Iterates over keys and objects in a weakly referenced object

## @ each\_value()

Iterates over keys and objects in a weakly referenced object

#### include?(p1)

Returns true if key is registered

## 🏐 inspect()

## 🏐 key?(p1)

Returns true if key is registered

# 🚳 keys()

Iterates over keys and objects in a weakly referenced
object

## 🚳 length()

### member?(p1)

Returns true if key is registered

🚳 size()

### values()

Iterates over values and objects in a weakly referenced object

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class Proc

Proc objects are blocks of code that have been bound to a set of local variables. Once bound, the code may be called in different contexts and still access those variables.



# In Files

proc.c

### Parent

Object

# **Public Class Methods**

 $\otimes$  new {|...| block }  $\rightarrow$  a\_proc

#### mew → a\_proc

Creates a new Proc object, bound to the current

context. Proc::new may be called without a block only within a method with an attached block, in which case that block is converted to the Proc object.



# **Public Instance Methods**

#### oproc === obj → result\_of\_proc

Invokes the block with obj as the proc's parameter like <u>#call</u>. It is to allow a proc object to be a target of when clause in a case statement.

- obj call(params,...) → obj
- ◎ prc[params,...] → obj

#### (params,...) → obj

Invokes the block, setting the block's parameters to the values in *params* using something close to method calling semantics. Generates a warning if multiple values are passed to a proc that expects just one (previously this silently converted the parameters to an array). Note that prc.() invokes prc.call() with the parameters given. It's a syntax sugar to hide "call".

For procs created using lambda or ->() an error is generated if the wrong number of parameters are passed to a <u>Proc</u> with multiple parameters. For procs created using Proc.new Or Kernel.proc, extra parameters are silently discarded. Returns the value of the last expression evaluated in the block. See also Proc#yield.



produces:

ŧ,	1							•	i
	from	prog.rb	:5:in `	` <ma< th=""><th>in&gt;'</th><th></th><th></th><th></th><th></th></ma<>	in>'				
	from	prog.rb	:5:in `	cal	1'				
p	prog.n	ˈb:4:in	`block	in	<main>':</main>	wrong	number	of	

### arity → fixnum

Returns the number of mandatory arguments. If the block is declared to take no arguments, returns 0. If the block is known to take exactly n arguments, returns n. If the block has optional arguments, returns -n-1, where n is the number of mandatory arguments, with the exception for blocks that are not lambdas and have only a finite number of optional arguments; in this latter case, returns n. Keywords arguments will considered as a single additional argument, that argument being mandatory if any keyword argument is mandatory. A proc with no argument declarations is the same as a block declaring || as its arguments.

proc {}.arity	
<pre>proc {    }.arity</pre>	
proc {  a  }.arity	
proc {  a, b  }.arity	
proc {  a, b, c  }.arity	
<pre>proc {  *a  }.arity</pre>	
proc {  a, *b  }.arity	
proc {  a, *b, c  }.arity	
proc {  x:, y:, z:0  }.arity	
proc {  *a, x:, y:0  }.arity	

### binding → binding

Returns the binding associated with *prc*. Note that Kernel#eval accepts either a Proc Or a Binding Object as its second parameter.



## ⊚ prc[params,...] → obj

### (params,...) → obj

Invokes the block, setting the block's parameters to the values in *params* using something close to method calling semantics. Generates a warning if multiple values are passed to a proc that expects just one (previously this silently converted the parameters to an array). Note that prc.() invokes prc.call() with the parameters given. It's a syntax sugar to hide "call".

For procs created using lambda or ->() an error is

generated if the wrong number of parameters are passed to a <u>Proc</u> with multiple parameters. For procs created using Proc.new Or Kernel.proc, extra parameters are silently discarded.

Returns the value of the last expression evaluated in the block. See also Proc#yield.



produces:



### rightarrow curry ightarrow a\_proc rightarrow curry(arity) ightarrow a\_proc

Returns a curried proc. If the optional *arity* argument is given, it determines the number of arguments. A curried proc receives some arguments. If a sufficient number of arguments are supplied, it passes the supplied arguments to the original proc and returns the result. Otherwise, returns another curried proc that takes the rest of arguments.





#### $\otimes$ hash $\rightarrow$ integer

Returns a hash value corresponding to proc body.

See also Object#hash.

#### inspect()

Alias for: to\_s

### $\otimes$ lambda? $\rightarrow$ true or false

Returns true for a <u>Proc</u> object for which argument handling is rigid. Such procs are typically generated by lambda.

A <u>Proc</u> object generated by proc ignores extra arguments.



It provides nil for missing arguments.



<u>#lambda?</u> is a predicate for the tricks. It returns true if no tricks apply.

lambda {}.lambda?	
proc {}.lambda?	

::new is the same as proc.



lambda, proc and <u>::new</u> preserve the tricks of a <u>Proc</u> object given by & argument.

lambda(λ {}).lambda? proc(λ {}).lambda? Proc.new(λ {}).lambda?		
lambda(&proc {}).lambda?	#=>	false
proc(&proc {}).lambda?	#=>	false
Proc.new(&proc {}).lambda?	#=>	false

A Proc object generated by & argument has the tricks



The & argument preserves the tricks if a Proc object

is given by & argument.



A Proc object converted from a method has no tricks.



define\_method is treated the same as method definition. The defined method has no tricks.

```
class C
  define_method(:d) {}
end
C.new.d(1,2) #=> ArgumentError
C.new.method(:d).to_proc.lambda? #=> true
```

define\_method always defines a method without the tricks, even if a non-lambda <u>Proc</u> object is given. This is the only exception for which the tricks are not preserved.

```
class C
  define_method(:e, &proc {})
end
C.new.e(1,2) #=> ArgumentError
C.new.method(:e).to_proc.lambda? #=> true
```

This exception insures that methods never have tricks and makes it easy to have wrappers to define methods that behave as usual.

```
class C
  def self.def2(name, &body)
    define_method(name, &body)
  end
  def2(:f) {}
```

end C.new.f(1,2) #=> ArgumentError

The wrapper *def2* defines a method which has no tricks.

#### ◎ parameters → array

Returns the parameter information of this proc.



### $\odot$ source\_location $\rightarrow$ [String, Fixnum]

Returns the Ruby source filename and line number containing this proc or nil if this proc was not defined in Ruby (i.e. native)

### o to\_proc → proc

Part of the protocol for converting objects to Proc objects. Instances of class Proc simply return themselves.

### os to\_s → string

Returns the unique identifier for this proc, along with an indication of where the proc was defined.

Also aliased as: inspect

### call(params,...) → obj

oprc[params,...] → obj

### (params,...) → obj

Invokes the block, setting the block's parameters to the values in *params* using something close to

method calling semantics. Generates a warning if multiple values are passed to a proc that expects just one (previously this silently converted the parameters to an array). Note that prc.() invokes prc.call() with the parameters given. It's a syntax sugar to hide "call".

For procs created using lambda or ->() an error is generated if the wrong number of parameters are passed to a <u>Proc</u> with multiple parameters. For procs created using Proc.new Or Kernel.proc, extra parameters are silently discarded.

Returns the value of the last expression evaluated in the block. See also Proc#yield.



produces:



Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# **module Process**

# **In Files**

process.c

l ruby.c

Constants

CLOCK\_BOOTTIME

CLOCK\_BOOTTIME\_ALARM

**CLOCK\_MONOTONIC** 

CLOCK\_MONOTONIC\_COARSE

CLOCK\_MONOTONIC\_FAST

CLOCK\_MONOTONIC\_PRECISE

CLOCK\_MONOTONIC\_RAW

CLOCK\_PROCESS\_CPUTIME\_ID

CLOCK\_PROF

**CLOCK\_REALTIME** 

CLOCK\_REALTIME\_ALARM

CLOCK\_REALTIME\_COARSE

#### CLOCK\_REALTIME\_FAST

#### **CLOCK\_REALTIME\_PRECISE**

#### CLOCK\_SECOND

#### CLOCK\_THREAD\_CPUTIME\_ID

#### CLOCK\_UPTIME

#### CLOCK\_UPTIME\_FAST

#### CLOCK\_UPTIME\_PRECISE

#### CLOCK\_VIRTUAL

#### PRIO\_PGRP

see ::setpriority

#### **PRIO\_PROCESS**

see ::setpriority

#### PRIO\_USER

see ::setpriority

#### **RLIMIT\_AS**

Maximum size of the process's virtual memory (address space) in bytes.

see the system getrlimit(2) manual for details.

#### **RLIMIT\_CORE**

Maximum size of the core file.

see the system getrlimit(2) manual for details.

#### RLIMIT\_CPU

CPU time limit in seconds.

see the system getrlimit(2) manual for details.

#### RLIMIT\_DATA

Maximum size of the process's data segment.

see the system getrlimit(2) manual for details.

#### **RLIMIT\_FSIZE**

Maximum size of files that the process may create.

see the system getrlimit(2) manual for details.

#### RLIMIT\_MEMLOCK

Maximum number of bytes of memory that may be locked into RAM.

see the system getrlimit(2) manual for details.

#### RLIMIT\_MSGQUEUE

Specifies the limit on the number of bytes that can be allocated for POSIX message queues for the real user ID of the calling process.

see the system getrlimit(2) manual for details.

#### RLIMIT\_NICE

Specifies a ceiling to which the process's nice value can be raised.

see the system getrlimit(2) manual for details.

#### **RLIMIT\_NOFILE**

Specifies a value one greater than the maximum file descriptor number that can be opened by this process.

see the system getrlimit(2) manual for details.

#### RLIMIT\_NPROC

The maximum number of processes that can be created for the real user ID of the calling process.

see the system getrlimit(2) manual for details.

#### RLIMIT\_RSS

Specifies the limit (in pages) of the process's resident set.

see the system getrlimit(2) manual for details.

#### **RLIMIT\_RTPRIO**

Specifies a ceiling on the real-time priority that may be set for this process.

see the system getrlimit(2) manual for details.

#### RLIMIT\_RTTIME

Specifies limit on CPU time this process scheduled under a real-time scheduling policy can consume.

see the system getrlimit(2) manual for details.

#### RLIMIT\_SBSIZE

Maximum size of the socket buffer.

#### **RLIMIT\_SIGPENDING**

Specifies a limit on the number of signals that may be queued for the real user ID of the calling process.

see the system getrlimit(2) manual for details.

#### **RLIMIT\_STACK**

Maximum size of the stack, in bytes.

see the system getrlimit(2) manual for details.

#### **RLIM\_INFINITY**

see ::setrlimit

#### RLIM\_SAVED\_CUR

see ::setrlimit

#### RLIM\_SAVED\_MAX

see ::setrlimit

#### WNOHANG

see ::wait

#### WUNTRACED

see ::wait

# **Public Class Methods**

### 🚳 abort

Kernel::abort([msg])

Process::abort([msg])

Terminate execution immediately, effectively by

calling Kernel.exit(false). If *msg* is given, it is written to STDERR prior to terminating.

#### argv0 → frozen\_string

Returns the name of the script being executed. The value is not affected by assigning a new value to \$0.

This method first appeared in Ruby 2.1 to serve as a global variable free means to get the script name.

#### $\otimes$ clock\_getres(clock\_id [, unit]) $\rightarrow$ number

Returns the time resolution returned by POSIX ::clock\_getres() function.

clock\_id specifies a kind of clock. See the document
Of Process.clock\_gettime for details.

clock\_id can be a symbol as Process.clock\_gettime. However the result may not be accurate. For example,

+::clock\_getres(:GETTIMEOFDAY\_BASED\_CLOCK\_REAl returns 1.0e-06 which means 1 microsecond, but actual resolution can be more coarse.

If the given clock\_id is not supported, Errno::EINVAL is raised.

unit specifies a type of the return value. Process.clock\_getres accepts unit as Process.clock\_gettime. The default value, :float\_second, is also same as Process.clock\_gettime.

Process.clock\_getres also accepts :hertz as unit. :hertz means a the reciprocal of :float\_second.

:hertz can be used to obtain the exact value of the clock ticks per second for times() function and

CLOCKS\_PER\_SEC for clock() function.

+::clock\_getres(:TIMES\_BASED\_CLOCK\_PROCESS\_CP :hertz)+ returns the clock ticks per second.

+::clock\_getres(:CLOCK\_BASED\_CLOCK\_PROCESS\_CF :hertz)+ returns CLOCKS\_PER\_SEC.



### or clock\_gettime(clock\_id [, unit]) → number or provide the set of the set

Returns a time returned by POSIX <u>::clock\_gettime()</u> function.



clock\_id specifies a kind of clock. It is specifed as a constant which begins with Process::CLOCK\_such as Process::CLOCK\_REALTIME and Process::CLOCK\_MONOTONIC.

The supported constants depends on OS and version. Ruby provides following types of clock\_id if available.

#### CLOCK\_REALTIME

SUSv2 to 4, Linux 2.5.63, FreeBSD 3.0, NetBSD 2.0, OpenBSD 2.1

#### CLOCK\_MONOTONIC

SUSv3 to 4, Linux 2.5.63, FreeBSD 3.0, NetBSD 2.0, OpenBSD 3.4

#### CLOCK\_PROCESS\_CPUTIME\_ID

SUSv3 to 4, Linux 2.5.63, OpenBSD 5.4

CLOCK\_THREAD\_CPUTIME\_ID

SUSv3 to 4, Linux 2.5.63, FreeBSD 7.1, OpenBSD 5.4

CLOCK\_VIRTUAL

FreeBSD 3.0, OpenBSD 2.1

CLOCK\_PROF FreeBSD 3.0, OpenBSD 2.1

CLOCK\_REALTIME\_FAST

FreeBSD 8.1

CLOCK\_REALTIME\_PRECISE

FreeBSD 8.1

CLOCK\_REALTIME\_COARSE

Linux 2.6.32

CLOCK\_REALTIME\_ALARM

Linux 3.0

CLOCK\_MONOTONIC\_FAST FreeBSD 8.1

CLOCK\_MONOTONIC\_PRECISE

FreeBSD 8.1

CLOCK\_MONOTONIC\_COARSE

Linux 2.6.32

CLOCK\_MONOTONIC\_RAW

Linux 2.6.28

#### **CLOCK\_BOOTTIME**

Linux 2.6.39

#### CLOCK\_BOOTTIME\_ALARM

Linux 3.0

#### **CLOCK\_UPTIME**

FreeBSD 7.0, OpenBSD 5.5

CLOCK\_UPTIME\_FAST

FreeBSD 8.1

#### CLOCK\_UPTIME\_PRECISE

FreeBSD 8.1

#### CLOCK\_SECOND

FreeBSD 8.1

Note that SUS stands for Single Unix Specification. SUS contains POSIX and ::clock\_gettime is defined in the POSIX part. SUS defines CLOCK\_REALTIME mandatory but CLOCK\_MONOTONIC, CLOCK\_PROCESS\_CPUTIME\_ID and CLOCK\_THREAD\_CPUTIME\_ID are optional.

Also, several symbols are accepted as clock\_id. There are emulations for ::clock\_gettime().

For example, Process::CLOCK\_REALTIME is defined as :GETTIMEOFDAY\_BASED\_CLOCK\_REALTIME when ::clock\_gettime() is not available.

Emulations for CLOCK\_REALTIME:

#### :GETTIMEOFDAY\_BASED\_CLOCK\_REALTIME

Use gettimeofday() defined by SUS. (SUSv4 obsoleted it, though.) The

resolution is 1 microsecond.

:TIME\_BASED\_CLOCK\_REALTIME

Use time() defined by ISO C. The resolution is 1 second.

Emulations for CLOCK\_MONOTONIC:

### :MACH\_ABSOLUTE\_TIME\_BASED\_CLOCK\_MONO

Use mach\_absolute\_time(), available on Darwin. The resolution is CPU dependent.

### :TIMES\_BASED\_CLOCK\_MONOTONIC

Use the result value of times() defined by POSIX. POSIX defines it as "times() shall return the elapsed real time, in clock ticks, since an arbitrary point in the past (for example, system start-up time)". For example, GNU/Linux returns a value based on jiffies and it is monotonic. However, 4.4BSD uses gettimeofday() and it is not monotonic. (FreeBSD uses ::clock gettime(CLOCK MONOTONIC) instead, though.) The resolution is the clock tick. "getconf CLK TCK" command shows the clock ticks per second. (The clock ticks per second is defined by HZ macro in older systems.) If it is 100 and clock t is 32 bits integer type, the resolution is 10 millisecond and cannot represent over 497 days.

Emulations for CLOCK\_PROCESS\_CPUTIME\_ID:

#### :GETRUSAGE\_BASED\_CLOCK\_PROCESS\_CPUTI

Use getrusage() defined by SUS. getrusage() is used with RUSAGE\_SELF to obtain the time only for the calling process (excluding the time for child processes). The result is addition of user time (ru\_utime) and system time (ru\_stime). The resolution is 1 microsecond.

#### :TIMES\_BASED\_CLOCK\_PROCESS\_CPUTIME\_ID

Use times() defined by POSIX. The result is addition of user time (tms\_utime) and system time (tms\_stime). tms\_cutime and tms\_cstime are ignored to exclude the time for child processes. The resolution is the clock tick. "getconf CLK\_TCK" command shows the clock ticks per second. (The clock ticks per second is defined by HZ macro in older systems.) If it is 100, the resolution is 10 millisecond.

### :CLOCK\_BASED\_CLOCK\_PROCESS\_CPUTIME\_ID

Use clock() defined by ISO C. The resolution is 1/CLOCKS\_PER\_SEC. CLOCKS\_PER\_SEC is the C-level macro defined by time.h. SUS defines CLOCKS\_PER\_SEC is 1000000. Non-Unix systems may define it a different value, though. If CLOCKS\_PER\_SEC is 1000000 as SUS, the resolution is 1 microsecond. If CLOCKS\_PER\_SEC is 1000000 and clock\_t is 32 bits integer type, it cannot represent over 72 minutes.

If the given clock\_id is not supported, Errno::EINVAL is raised.

unit specifies a type of the return value.

### :float\_second number of seconds as a float (default)

:float\_millisecond

number of milliseconds as a float

:float\_microsecond

number of microseconds as a float

:second

number of seconds as an integer

:millisecond

number of milliseconds as an integer

:microsecond

number of microseconds as an integer

:nanosecond

number of nanoseconds as an integer

The underlying function, :::clock\_gettime(), returns a number of nanoseconds. Float object (IEEE 754 double) is not enough to represent the return value for CLOCK\_REALTIME. If the exact nanoseconds value is required, use :nanoseconds as the unit.

The origin (zero) of the returned value varies. For example, system start up time, process start up time, the Epoch, etc.

The origin in <u>CLOCK\_REALTIME</u> is defined as the Epoch (1970-01-01 00:00:00 UTC). But some systems count leap seconds and others doesn't. So the result can be interpreted differently across systems. <u>Time.now</u> is recommended over <u>CLOCK\_REALTIME</u>.

### $_{\odot}$ daemon() → 0 $_{\odot}$ daemon(nochdir=nil,noclose=nil) → 0

Detach the process from controlling terminal and run in the background as system daemon. Unless the argument nochdir is true (i.e. non false), it changes the current working directory to the root ("/"). Unless the argument noclose is true, daemon() will redirect standard input, standard output and standard error to /dev/null. Return zero on success, or raise one of Errno::\*.

### $_{\odot}$ detach(pid) $\rightarrow$ thread

Some operating systems retain the status of terminated child processes until the parent collects that status (normally using some variant of wait(). If the parent never collects this status, the child stays around as a *zombie* process. Process::detach prevents this by setting up a separate Ruby thread whose sole job is to reap the status of the process *pid* when it terminates. Use detach only when you do not intent to explicitly wait for the child to terminate.

The waiting thread returns the exit status of the detached process when it terminates, so you can use Thread#join to know the result. If specified *pid* is not a valid child process ID, the thread returns nil immediately.

The waiting thread has pid method which returns the pid.

In this first example, we don't reap the first child process, so it appears as a zombie in the process status display.

```
p1 = fork { sleep 0.1 }
p2 = fork { sleep 0.2 }
```

```
Process.waitpid(p2)
sleep 2
system("ps -ho pid,state -p #{p1}")
```

produces:

27389 Z

In the next example, Process::detach is used to reap the child automatically.

```
p1 = fork { sleep 0.1 }
p2 = fork { sleep 0.2 }
Process.detach(p1)
Process.waitpid(p2)
sleep 2
system("ps -ho pid,state -p #{p1}")
```

(produces no output)

```
    egid → fixnum
```

Process::GID.eid → fixnum

#### Process::Sys.geteid → fixnum

Returns the effective group ID for this process. Not available on all platforms.

Process.egid #=> 500

#### egid = fixnum → fixnum

Sets the effective group ID for this process. Not available on all platforms.

euid → fixnum

Process::UID.eid → fixnum

#### Process::Sys.geteuid → fixnum

Returns the effective user ID for this process.

#### Process.euid #=> 501

#### 🐵 euid= user

Sets the effective user ID for this process. Not available on all platforms.

### @ exec([env,] command... [,options])

Replaces the current process by running the given external *command*, which can take one of the following forms:

```
exec(commandline)
```

command line string which is passed to the standard shell

```
exec(cmdname, arg1, ...)
```

```
command name and one or more arguments (no shell)
```

```
exec([cmdname, argv0], arg1, ...)
```

command name,  $\underline{\text{argv}}$  and zero or more arguments (no shell)

In the first form, the string is taken as a command line that is subject to shell expansion before being executed.

The standard shell always means "/bin/sh" on Unixlike systems, same as ENV["RUBYSHELL"] (or ENV["COMSPEC"] on Windows NT series), and similar.

If the string from the first form (exec("command")) follows these simple rules:

- no meta characters
- local no shell reserved word and no special built-in

Ruby invokes the command directly without shell

You can force shell invocation by adding ";" to the string (because ";" is a meta character).

Note that this behavior is observable by pid obtained (return value of spawn() and <u>IO#pid</u> for <u>IO.popen</u>) is the pid of the invoked command, not shell.

In the second form (exec("command1", "arg1", ...)), the first is taken as a command name and the rest are passed as parameters to command with no shell expansion.

In the third form (exec(["command", "argv0"], "arg1", ...)), starting a two-element array at the beginning of the command, the first element is the command to be executed, and the second argument is used as the argv[0] value, which may show up in process listings.

In order to execute the command, one of the exec(2) system calls are used, so the running command may inherit some of the environment of the original program (including open file descriptors).

This behavior is modified by the given env and options parameters. See ::spawn for details.

If the command fails to execute (typically Errno::ENOENT when it was not found) a SystemCallError exception is raised.

This method modifies process attributes according to given options before exec(2) system call. See ::spawn for more details about the given options.

The modified attributes may be retained when exec(2) system call fails.

For example, hard resource limits are not restorable.

Consider to create a child process using <u>::spawn</u> or Kernel#system if this is not acceptable.



### exit(status=true)

Sernel::exit(status=true)

### Process::exit(status=true)

Initiates the termination of the Ruby script by raising the systemExit exception. This exception may be caught. The optional parameter is used to return a status code to the invoking environment. true and FALSE of *status* means success and failure respectively. The interpretation of other integer values are system dependent.

```
begin
  exit
  puts "never get here"
rescue SystemExit
  puts "rescued a SystemExit exception"
end
puts "after begin block"
```

produces:

```
rescued a SystemExit exception after begin block
```

Just prior to termination, Ruby executes any at\_exit functions (see Kernel::at\_exit) and runs any object finalizers (see ObjectSpace.define\_finalizer).



### exit!(status=false)

Exits the process immediately. No exit handlers are run. *status* is returned to the underlying system as the exit status.

```
Process.exit!(true)
```

### rightarrow fork [{ block }] → fixnum or nil rightarrow fork [{ block }] → fixnum or nil

Creates a subprocess. If a block is specified, that block is run in the subprocess, and the subprocess terminates with a status of zero. Otherwise, the fork call returns twice, once in the parent, returning the process ID of the child, and once in the child, returning *nil*. The child process can exit using Kernel.exit! to avoid running any at\_exit functions. The parent process should use Process.wait to collect the termination statuses of its children or use Process.detach to register disinterest in their status; otherwise, the operating system may accumulate zombie processes.

The thread calling fork is the only thread in the created child process. fork doesn't copy other threads.

If fork is not usable, Process.respond\_to?(:fork)

returns false.

Note that fork(2) is not available on some platforms like Windows and NetBSD 4. Therefore you should use spawn() instead of fork().

### getpgid(pid) → integer

Returns the process group ID for the given process id. Not available on all platforms.

```
Process.getpgid(Process.ppid()) #=> 25527
```

### ⊚ getpgrp → integer

Returns the process group ID for this process. Not available on all platforms.

Process.getpgid(0) #=> 25527 Process.getpgrp #=> 25527

### getpriority(kind, integer) → fixnum

Gets the scheduling priority for specified process, process group, or user. *kind* indicates the kind of entity to find: one of Process::PRIO\_PGRP,

Process::PRIO\_USER, Or Process::PRIO\_PROCESS. integer is an id indicating the particular process, process group, or user (an id of 0 means *current*). Lower priorities are more favorable for scheduling. Not available on all platforms.



# getrlimit(resource) → [cur\_limit, max\_limit]

Gets the resource limit of the process. cur\_limit

means current (soft) limit and *max\_limit* means maximum (hard) limit.

*resource* indicates the kind of resource to limit. It is specified as a symbol such as :core, a string such as "CORE" or a constant such as Process::RLIMIT\_CORE. See ::setrlimit for details.

*cur\_limit* and *max\_limit* may be Process::RLIM\_INFINITY, Process::RLIM\_SAVED\_MAX Or Process::RLIM\_SAVED\_CUR. See <u>::setrlimit</u> and the system getrlimit(2) manual for details.

### ⊚ getsid() → integer ⊚ getsid(pid) → integer

Returns the session ID for for the given process id. If not give, return current process sid. Not available on all platforms.



### igid → fixnum

#### ◎ Process::GID.rid → fixnum

#### Search Process::Sys.getgid → fixnum

Returns the (real) group ID for this process.

Process.gid #=> 500

#### gid= fixnum → fixnum

Sets the group ID for this process.

groups → array

Get an Array of the gids of groups in the supplemental group access list for this process.



### $_{\odot}$ initgroups(username, gid) $\rightarrow$ array

Initializes the supplemental group access list by reading the system group database and using all groups of which the given user is a member. The group with the specified *gid* is also added to the list. Returns the resulting Array of the gids of all the groups in the supplementary group access list. Not available on all platforms.



### $_{\odot}$ kill(signal, pid, ...) → fixnum

Sends the given signal to the specified process id(s) if *pid* is positive. If *pid* is zero *signal* is sent to all processes whose group ID is equal to the group ID of the process. *signal* may be an integer signal number or a POSIX signal name (either with or without a sig prefix). If *signal* is negative (or starts with a minus sign), kills process groups instead of processes. Not

all signals are available on all platforms. The keys and values of signal.list are known signal names and numbers, respectively.

```
pid = fork do
   Signal.trap("HUP") { puts "Ouch!"; exit }
   # ... do some work ...
end
# ...
Process.kill("HUP", pid)
Process.wait
```

produces:

#### Ouch!

If *signal* is an integer but wrong for signal, Errno::EINVAL OF RangeError will be raised. Otherwise unless *signal* is a String Or a Symbol, and a known signal name, ArgumentError will be raised.

Also, Errno::ESRCH Or RangeError for invalid *pid*, Errno::EPERM when failed because of no privilege, will be raised. In these cases, signals may have been sent to preceding processes.

#### maxgroups → fixnum

Returns the maximum number of gids allowed in the supplemental group access list.



#### maxgroups= fixnum → fixnum

Sets the maximum number of gids allowed in the supplemental group access list.



Returns the process id of this process. Not available on all platforms.

Process.pid #=> 27415

### ◎ ppid → fixnum

Returns the process id of the parent of this process. Returns untrustworthy value on Win32/64. Not available on all platforms.

```
puts "I am #{Process.pid}"
Process.fork { puts "Dad is #{Process.ppid}" }
```

produces:

I am 27417 Dad is 27417

### $_{\odot}$ setpgid(pid, integer) $\rightarrow$ 0

Sets the process group ID of *pid* (0 indicates this process) to *integer*. Not available on all platforms.

#### $\odot$ setpgrp $\rightarrow$ 0

Equivalent to setpgid(0,0). Not available on all platforms.

### $_{\odot}$ setpriority(kind, integer, priority) $\rightarrow$ 0

See Process#getpriority.

Process.setpriority(Process::PRIO\_USER, 0, 19)
Process.setpriority(Process::PRIO\_PROCESS, 0, 19)
Process.getpriority(Process::PRIO\_USER, 0)
Process.getpriority(Process::PRIO\_PROCESS, 0)

### Setproctitle(string) → string

Sets the process title that appears on the ps(1) command. Not necessarily effective on all platforms. No exception will be raised regardless of the result, nor will <u>NotImplementedError</u> be raised even if the platform does not support the feature.

Calling this method does not affect the value of \$0.

```
Process.setproctitle('myapp: worker #%d' % worker
```

This method first appeared in Ruby 2.1 to serve as a global variable free means to change the process title.

### setrlimit(resource, cur\_limit, max\_limit) → ⊚ nil

### setrlimit(resource, cur\_limit) → nil

Sets the resource limit of the process. *cur\_limit* means current (soft) limit and *max\_limit* means maximum (hard) limit.

If *max\_limit* is not given, *cur\_limit* is used.

*resource* indicates the kind of resource to limit. It should be a symbol such as :core, a string such as "CORE" or a constant such as Process::RLIMIT\_CORE. The available resources are OS dependent. Ruby may support following resources.

#### AS

total available memory (bytes) (SUSv3, NetBSD, FreeBSD, OpenBSD but 4.4BSD-Lite)

### CORE

core size (bytes) (SUSv3)

CPU

CPU time (seconds) (SUSv3)

#### DATA

data segment (bytes) (SUSv3)

### FSIZE

file size (bytes) (SUSv3)

### MEMLOCK

total size for mlock(2) (bytes) (4.4BSD, GNU/Linux)

### MSGQUEUE

allocation for POSIX message queues (bytes) (GNU/Linux)

### NICE

ceiling on process's nice(2) value (number) (GNU/Linux)

### NOFILE

file descriptors (number) (SUSv3)

### NPROC

number of processes for the user (number) (4.4BSD, GNU/Linux)

### RSS

resident memory size (bytes) (4.2BSD, GNU/Linux)

### **RTPRIO**
ceiling on the process's real-time priority (number) (GNU/Linux)

#### RTTIME

CPU time for real-time process (us) (GNU/Linux)

#### SBSIZE

all socket buffers (bytes) (NetBSD, FreeBSD)

#### SIGPENDING

number of queued signals allowed (signals) (GNU/Linux)

## STACK

stack size (bytes) (SUSv3)

*cur\_limit* and *max\_limit* may be :INFINITY, "INFINITY" Or Process::RLIM\_INFINITY, which means that the resource is not limited. They may be Process::RLIM\_SAVED\_MAX, Process::RLIM\_SAVED\_CUR and corresponding symbols and strings too. See system setrlimit(2) manual for details.

The following example raises the soft limit of core size to the hard limit to try to make core dump possible.



## Setsid → fixnum

Establishes this process as a new session and process group leader, with no controlling tty. Returns the session id. Not available on all platforms.

## spawn([env,] command... [,options]) → pid spawn([env,] command... [,options]) → pid

spawn executes specified command and return its pid.



This method is similar to Kernel#system but it doesn't wait for the command to finish.

The parent process should use Process.wait to collect the termination status of its child or use Process.detach to register disinterest in their status; otherwise, the operating system may accumulate zombie processes.

spawn has bunch of options to specify process attributes:

```
env: hash
  name => val : set the environment variable
  name => nil : unset the environment variable
command...:
  commandline
                                : command line stri
  cmdname, arg1, ...
                               : command name and
  [cmdname, argv0], arg1, ... : command name, arg
options: hash
  clearing environment variables:
    :unsetenv_others => true : clear environmer
    :unsetenv_others => false : don't clear (def
  process group:
    :pgroup => true or 0 : make a new process gro
    :pgroup => pgid : join to specified prod
:pgroup => nil : don't change the proce
```

```
create new process group: Windows only
    :new_pgroup => true : the new process is the
    :new_pgroup => false : don't create a new pro
  resource limit: resourcename is core, cpu, data
    :rlimit resourcename => limit
    :rlimit_resourcename => [cur_limit, max_limit
  umask:
     :umask => int
  redirection:
    key:
      FD
      FD : single file descriptor i
[FD, FD, ...] : multiple file descriptor
                       : single file descriptor in
    value:
      FD
                                 : redirect to the
      string
                                 : redirect to fil
      [string]
                                 : redirect to fil
      [string, open_mode] : redirect to fil
      [string, open_mode, perm] : redirect to fi
                               : redirect to the
      [:child, FD]
      :close
                                 : close the file
    FD is one of follows
              : the file descriptor 0 which is the
       :in
              : the file descriptor 1 which is the
      :out
      err : the file descriptor 2 which is th
      integer : the file descriptor of specified
          : the file descriptor specified as
      io
  file descriptor inheritance: close non-redirect
     :close_others => true : don't inherit
  current directory:
    :chdir => str
  The 'cmdname, arg1, ...' form does not use the
  on different OSes, different things are provide
  commands. An example of this is 'echo', which
  on Windows, but is a normal program on Linux ar
  This means that %xProcess.spawn 'echo', '%Path%
  the contents of the %x%Path%` environment varia
  but %xProcess.spawn 'echo', '$PATH'`
                                        prints the
•
```

If a hash is given as env, the environment is updated by env before exec(2) in the child process. If a pair in env has nil as the value, the variable is deleted.



▲

If a hash is given as options, it specifies process group, create new process group, resource limit, current directory, umask and redirects for the child process. Also, it can be specified to clear environment variables.

The :unsetenv\_others key in options specifies to clear environment variables, other than specified by env.



The :pgroup key in options specifies a process group. The corresponding value should be true, zero or positive integer. true and zero means the process should be a process leader of a new process group. Other values specifies a process group to be belongs.



The :new\_pgroup key in options specifies to pass CREATE\_NEW\_PROCESS\_GROUP flag to CreateProcessW() that is Windows API. This option is only for Windows. true means the new process is the root process of the new process group. The new process has CTRL+C disabled. This flag is necessary for Process.kill(:SIGINT, pid) on the subprocess. :new\_pgroup is false by default.



The :rlimit\_foo key specifies a resource limit. foo

 $\mathbf{F}$ 

should be one of resource types such as core. The corresponding value should be an integer or an array which have one or two integers: same as cur\_limit and max\_limit arguments for ::setrlimit.



The :umask key in options specifies the umask.

```
pid = spawn(command, :umask=>077)
```

The :in, :out, :err, a fixnum, an <u>IO</u> and an array key specifies a redirection. The redirection maps a file descriptor in the child process.

For example, stderr can be merged into stdout as follows:

pid =	=	<pre>spawn(command,</pre>	:err=>:out)
pid =	=	<pre>spawn(command,</pre>	2=>1)
pid =	=	<pre>spawn(command,</pre>	<pre>STDERR=&gt;:out)</pre>
pid =	=	spawn(command,	STDERR=>STDOUT)

The hash keys specifies a file descriptor in the child process started by spawn. :err, 2 and STDERR specifies the standard error stream (stderr).

The hash values specifies a file descriptor in the parent process which invokes spawn. :out, 1 and STDOUT specifies the standard output stream (stdout).

In the above example, the standard output in the child process is not specified. So it is inherited from the parent process.

The standard input stream (stdin) can be specified by :in, 0 and STDIN.

A filename can be specified as a hash value.



For stdout and stderr (and combination of them), it is opened in write mode. Otherwise read mode is used.

For specifying flags and permission of file creation explicitly, an array is used instead.



The array specifies a filename, flags and permission. The flags can be a string or an integer. If the flags is omitted or nil, File::RDONLY is assumed. The permission should be an integer. If the permission is omitted or nil, 0644 is assumed.

If an array of IOs and integers are specified as a hash key, all the elements are redirected.



Another way to merge multiple file descriptors is [:child, fd]. [:child, fd] means the file descriptor in the child process. This is different from fd. For example, :err=>:out means redirecting child stderr to parent stdout. But :err=>[:child, :out] means redirecting child stderr to child stdout. They differ if stdout is redirected in the child process as follows.



[:child, :out] can be used to merge stderr into stdout in <u>IO.popen</u>. In this case, <u>IO.popen</u> redirects stdout to a pipe in the child process and [:child, :out] refers the redirected stdout.



The :chdir key in options specifies the current directory.

```
pid = spawn(command, :chdir=>"/var/tmp")
```

spawn closes all non-standard unspecified descriptors by default. The "standard" descriptors are 0, 1 and 2. This behavior is specified by

:close\_others option. :close\_others doesn't affect the standard descriptors which are closed only if :close is specified explicitly.



:close\_others is true by default for spawn and IO.popen.

Note that fds which close-on-exec flag is already set are closed regardless of :close\_others option.

So IO.pipe and spawn can be used as IO.popen.



:close is specified as a hash value to close a fd individually.



If a file descriptor need to be inherited, io=>io can be used.



It is also possible to exchange file descriptors.

```
pid = spawn(command, :out=>:err, :err=>:out)
```

The hash keys specify file descriptors in the child process. The hash values specifies file descriptors in the parent process. So the above specifies exchanging stdout and stderr. Internally, spawn uses an extra file descriptor to resolve such cyclic file descriptor mapping.

See Kernel.exec for the standard shell.

## on times → aProcessTms

Returns a Tms structure (see Process::Tms) that contains user and system CPU times for this process, and also for children processes.



## oid → fixnum

## Process::UID.rid → fixnum

#### Process::Sys.getuid → fixnum

Returns the (real) user ID of this process.

Process.uid #=> 501

#### ouid= user → numeric

Sets the (user) user ID for this process. Not available on all platforms.

## ⊚ wait() → fixnum

## $_{\odot}$ wait(pid=-1, flags=0) → fixnum

## $_{\odot}$ waitpid(pid=-1, flags=0) → fixnum

Waits for a child process to exit, returns its process id, and sets \$? to a Process::Status object containing information on that process. Which child it waits on depends on the value of *pid*:

#### > 0

Waits for the child whose process ID equals pid.

#### 0

Waits for any child whose process group ID equals that of the calling process.

#### -1

Waits for any child process (the default if no *pid* is given).

#### < -1

Waits for any child whose process group ID equals the absolute value of *pid*.

The *flags* argument may be a logical or of the flag values Process::WNOHANG (do not block if no child

available) or Process::WUNTRACED (return stopped children that haven't been reported). Not all flags are available on all platforms, but a flag value of zero will work on all platforms.

Calling this method raises a <u>SystemCallError</u> if there are no child processes. Not available on all platforms.

<pre>include Process fork { exit 99 } wait \$?.exitstatus</pre>	
<pre>pid = fork { sleep 3 } Time.now waitpid(pid, Process::WNOHANG) Time.now waitpid(pid, 0) Time.now</pre>	
	J J

 $_{\odot}$  wait2(pid=-1, flags=0) → [pid, status]  $_{\odot}$  waitpid2(pid=-1, flags=0) → [pid, status]

Waits for a child process to exit (see <u>::waitpid</u> for exact semantics) and returns an array containing the process id and the exit status (a Process::Status object) of that child. Raises a <u>SystemCallError</u> if there are no child processes.



## $_{\odot}$ waitall → [[pid1,status1], ...]

Waits for all children, returning an array of *pid/status* pairs (where *status* is a Process::Status object).



#### sleep 0.1; exit 1 } fork fork { exit 0 } p Process.waitall

#### produces:

[[30982,	<pre>#<process::status:< pre=""></process::status:<></pre>	pid	30982	exit	0>],
[30979,	<pre>#<process::status:< pre=""></process::status:<></pre>	pid	30979	exit	1>],
[30976,	<pre>#<process::status:< pre=""></process::status:<></pre>	pid	30976	exit	2>]]

## i wait() → fixnum

## $\otimes$ wait(pid=-1, flags=0) $\rightarrow$ fixnum $\otimes$ waitpid(pid=-1, flags=0) $\rightarrow$ fixnum

Waits for a child process to exit, returns its process id, and sets \$? to a Process::Status object containing information on that process. Which child it waits on depends on the value of *pid*:

#### > 0

Waits for the child whose process ID equals pid.

#### 0

Waits for any child whose process group ID equals that of the calling process.

#### -1

Waits for any child process (the default if no *pid* is given).

#### < -1

Waits for any child whose process group ID equals the absolute value of pid.

The *flags* argument may be a logical or of the flag values Process: : WNOHANG (do not block if no child available) or process::WUNTRACED (return stopped children that haven't been reported). Not all flags are available on all platforms, but a flag value of zero will work on all platforms.

Calling this method raises a <u>SystemCallError</u> if there are no child processes. Not available on all platforms.



## rightarrow wait2(pid=-1, flags=0) → [pid, status] rightarrow waitpid2(pid=-1, flags=0) → [pid, status]

Waits for a child process to exit (see <u>::waitpid</u> for exact semantics) and returns an array containing the process id and the exit status (a Process::Status object) of that child. Raises a <u>SystemCallError</u> if there are no child processes.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## module Process::GID

The Process::GID module contains a collection of module functions which can be used to portably get, set, and switch the current process's real, effective, and saved group IDs.

## **In Files**

process.c

## **Public Class Methods**

## Process::GID.change\_privilege(group) → ⊚ fixnum

Change the current process's real and effective group ID to that specified by *group*. Returns the new group ID. Not available on all platforms.



## $\odot$ egid $\rightarrow$ fixnum

Process::GID.eid → fixnum

## Process::Sys.geteid → fixnum

Returns the effective group ID for this process. Not available on all platforms.



## $\odot$ Process::GID.from\_name(name) $\rightarrow$ gid

Get the group ID by the *name*. If the group is not found, ArgumentError will be raised.



## Process::GID.grant\_privilege(group) → ⊚ fixnum

## Process::GID.eid = group → fixnum

Set the effective group ID, and if possible, the saved group ID of the process to the given *group*. Returns the new effective group ID. Not available on all platforms.



## $\odot$ Process::GID.re\_exchange $\rightarrow$ fixnum

Exchange real and effective group IDs and return the new effective group ID. Not available on all platforms.



## Process::GID.re\_exchangeable? $\rightarrow$ true or $\odot$ false

Returns true if the real and effective group IDs of a process may be exchanged on the current platform.

# gid → fixnum Process::GID.rid → fixnum Process::Sys.getgid → fixnum Returns the (real) group ID for this process. Process.gid #=> 500

Process::GID.sid\_available? → true or false Returns true if the current platform has saved group ID functionality.

# Process::GID.switch → fixnum Process::GID.switch {|| block} → object Switch the offective and real group IDs of the curr

Switch the effective and real group IDs of the current process. If a *block* is given, the group IDs will be switched back after the block is executed. Returns the new effective group ID if called without a block, and the return value of the block if one is given.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class Process::Status

Process::Status encapsulates the information on the status of a running or terminated system process. The built-in variable \$? is either nil or a Process::Status Object.



Posix systems record information on processes using a 16-bit integer. The lower bits record the process status (stopped, exited, signaled) and the upper bits possibly contain additional information (for example the program's return code in the case of exited processes). Pre Ruby 1.8, these bits were exposed directly to the Ruby program. Ruby now encapsulates these in a Process::Status object. To maximize compatibility, however, these objects retain a bitoriented interface. In the descriptions that follow, when we talk about the integer value of *stat*, we're referring to this 16 bit value.

## **In Files**

process.c

## Parent

Object

## **Public Instance Methods**

## stat & num → fixnum

Logical AND of the bits in *stat* with *num*.



#### stat == other → true or false

Returns true if the integer value of stat equals other.

## stat >> num → fixnum

Shift the bits in *stat* right *num* places.



## $\odot$ coredump? $\rightarrow$ true or false

Returns true if *stat* generated a coredump when it terminated. Not available on all platforms.

## ⊚ exited? → true or false

Returns true if *stat* exited normally (for example using an exit() call or finishing the program).

## ⊚ exitstatus → fixnum or nil

Returns the least significant eight bits of the return code of *stat*. Only available if exited? is true.

<pre>fork { } Process.wait \$?.exited? \$?.exitstatus</pre>	
<pre>fork { exit 99 } Process.wait \$?.exited? \$?.exitstatus</pre>	

## inspect → string

Override the inspection method.



## is pid → fixnum

Returns the process ID that this status object represents.



## $_{\odot}$ signaled? $\rightarrow$ true or false

Returns true if *stat* terminated because of an uncaught signal.

## $_{\odot}$ stopped? $\rightarrow$ true or false

Returns true if this process is stopped. This is only returned if the corresponding wait call had the WUNTRACED flag set.

### stopsig → fixnum or nil

Returns the number of the signal that caused *stat* to stop (or nil if self is not stopped).

#### $_{\odot}$ success? → true, false or nil

Returns true if *stat* is successful, false if not. Returns nil if exited? is not true.

#### termsig → fixnum or nil

Returns the number of the signal that caused *stat* to terminate (or nil if self was not terminated by an uncaught signal).

## osto\_i → fixnum

#### to\_int → fixnum

Returns the bits in *stat* as a Fixnum. Poking around in these bits is platform dependent.



## $\odot$ to\_s $\rightarrow$ string

Show pid and exit status as a string.



Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## module Process::Sys

The Process::Sys module contains UID and GID functions which provide direct bindings to the system calls of the same names instead of the more-portable versions of the same functionality found in the Process, Process::UID, and Process::GID modules.

## In Files

process.c

## **Public Class Methods**

- segid → fixnum
- Process::GID.eid → fixnum

## Process::Sys.geteid → fixnum

Returns the effective group ID for this process. Not available on all platforms.





## Process::Sys.issetugid → true or false

Returns true if the process was created as a result of an execve(2) system call which had either of the setuid or setgid bits set (and extra privileges were given as a result) or if it has changed any of its real, effective or saved user or group IDs since it began execution.

#### Process::Sys.setegid(group) → nil

Set the effective group ID of the calling process to *group*. Not available on all platforms.

#### Process::Sys.seteuid(user) → nil

Set the effective user ID of the calling process to *user*. Not available on all platforms.

## Process::Sys.setgid(group) → nil

Set the group ID of the current process to *group*. Not available on all platforms.

#### Process::Sys.setregid(rid, eid) → nil

Sets the (group) real and/or effective group IDs of the current process to *rid* and *eid*, respectively. A value of -1 for either means to leave that ID unchanged. Not available on all platforms.

## 

Sets the (group) real, effective, and saved user IDs of the current process to *rid*, *eid*, and *sid* respectively. A value of -1 for any value means to leave that ID unchanged. Not available on all platforms.

## 

Sets the (user) real, effective, and saved user IDs of the current process to *rid*, *eid*, and *sid* respectively. A value of -1 for any value means to leave that ID unchanged. Not available on all platforms.

#### Process::Sys.setreuid(rid, eid) → nil

Sets the (user) real and/or effective user IDs of the current process to *rid* and *eid*, respectively. A value of -1 for either means to leave that ID unchanged. Not available on all platforms.

## Process::Sys.setrgid(group) → nil

Set the real group ID of the calling process to *group*. Not available on all platforms.

## Process::Sys.setruid(user) → nil

Set the real user ID of the calling process to *user*. Not available on all platforms.

## Process::Sys.setuid(user) → nil

Set the user ID of the current process to *user*. Not available on all platforms.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## module Process::UID

The Process::UID module contains a collection of module functions which can be used to portably get, set, and switch the current process's real, effective, and saved user IDs.

## **In Files**

process.c

## **Public Class Methods**

#### 

Change the current process's real and effective user ID to that specified by *user*. Returns the new user ID. Not available on all platforms.



## euid → fixnum

- Process::UID.eid → fixnum
- Process::Sys.geteuid → fixnum

Returns the effective user ID for this process.



## $\odot$ Process::UID.from\_name(name) $\rightarrow$ uid

Get the user ID by the *name*. If the user is not found, ArgumentError will be raised.



# Process::UID.grant\_privilege(user) → fixnum Process::UID.eid= user → fixnum

Set the effective user ID, and if possible, the saved user ID of the process to the given *user*. Returns the new effective user ID. Not available on all platforms.



## Process::UID.re\_exchange → fixnum

Exchange real and effective user IDs and return the new effective user ID. Not available on all platforms.



# Process::UID.re\_exchangeable? $\rightarrow$ true or $_{\odot}$ false

Returns true if the real and effective user IDs of a process may be exchanged on the current platform.

ightarrow uid → fixnum
 Process::UID.rid → fixnum

### Process::Sys.getuid → fixnum

Returns the (real) user ID of this process.

Process.uid #=> 501

#### Process::UID.sid\_available? → true or false Returns true if the current platform has saved user ID functionality.

# Process::UID.switch → fixnum Process::UID.switch {|| block} → object

Switch the effective and real user IDs of the current process. If a *block* is given, the user IDs will be switched back after the block is executed. Returns the new effective user ID if called without a block, and the return value of the block if one is given.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class Process::Waiter

## In Files

process.c

## Parent

Thread

## **Public Instance Methods**

## pid()

Generated by <u>RDoc</u> 3.12.2. Generated with the Darkfish Rdoc Generator 3.

## class Random

Random provides an interface to Ruby's pseudo-random number generator, or PRNG. The PRNG produces a deterministic sequence of bits which approximate true randomness. The sequence may be represented by integers, floats, or binary strings.

The generator may be initialized with either a system-generated or user-supplied seed value by using <u>::srand</u>.

The class method <u>#rand</u> provides the base functionality of <u>Kernel#rand</u> along with better handling of floating point values. These are both interfaces to Random::DEFAULT, the Ruby system PRNG.

::new will create a new PRNG with a state independent of Random::DEFAULT, allowing multiple generators with different seed values or sequence positions to exist simultaneously. <u>Random</u> objects can be marshaled, allowing sequences to be saved and resumed.

PRNGs are currently implemented as a modified Mersenne Twister with a period of 2\*\*19937-1.

## **In Files**

📄 random.c

## Parent

Object

## Constants

## DEFAULT

Direct access to Ruby's Pseudorandom number generator (PRNG).

## **Public Class Methods**

## onew(seed = Random.new\_seed) → prng

Creates a new PRNG using seed to set the initial state. If seed is omitted, the generator is initialized with ::new\_seed.

See <u>::srand</u> for more information on the use of seed values.

## onew\_seed → integer

Returns an arbitrary seed value. This is used by <u>::new</u> when no seed value is specified as an argument.

Random.new_seed	
<b>↓</b>	•

 $rightarrow rand \rightarrow float$ 

## on rand(max) → number

Alias of Random::DEFAULT.rand.

# srand(number = Random.new\_seed) → old\_seed

Seeds the system pseudo-random number generator, Random::DEFAULT, with number. The previous seed value is returned.

If number is omitted, seeds the generator using a source of entropy provided by the operating system, if available (/dev/urandom on Unix systems or the RSA cryptographic provider on Windows), which is then combined with the time, the process id, and a sequence number.

srand may be used to ensure repeatable sequences of pseudo-random numbers between different runs of the program. By setting the seed to a known value, programs can be made deterministic during testing.



## **Public Instance Methods**

## $_{\odot}$ prng1 == prng2 → true or false

Returns true if the two generators have the same internal state, otherwise false. Equivalent generators will return the same sequence of pseudo-random numbers. Two generators will generally have the same state only if they were initialized with the same seed

```
Random.new == Random.new# => falseRandom.new(1234) == Random.new(1234) # => true
```

and have the same invocation history.



## bytes(size) → a\_string

Returns a random binary string containing size bytes.



# rand → float (max) → number

When max is an <u>Integer</u>, rand returns a random integer greater than or equal to zero and less than max. Unlike <u>Kernel#rand</u>, when max is a negative integer or zero, rand raises an <u>ArgumentError</u>.

When max is a Float, rand returns a random floating point number between 0.0 and max, including 0.0 and excluding max.



When max is a <u>Range</u>, rand returns a random number where range.member?(number) == true.



Both the beginning and ending values of the range must respond to subtract (-) and add (+)methods, or rand will raise an ArgumentError.

#### $\odot$ seed $\rightarrow$ integer

Returns the seed value used to initialize the generator. This may be used to initialize another generator with the same state at a later time, causing it to produce the same sequence of numbers.

```
prng1 = Random.new(1234)
prng1.seed  #=> 1234
prng1.rand(100)  #=> 47
prng2 = Random.new(prng1.seed)
prng2.rand(100)  #=> 47
```

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class Range

A Range represents an interval—a set of values with a beginning and an end. Ranges may be constructed using the s..e and s...e literals, or with <u>::new</u>. Ranges constructed using ... run from the beginning to the end inclusively. Those created using ... exclude the end value. When used as an iterator, ranges return each value in the sequence.



## **Custom Objects in Ranges**

Ranges can be constructed using any objects that can be compared using the <=> operator. Methods that treat the range as a sequence (each and methods inherited from Enumerable) expect the begin object to implement a succ method to return the next object in sequence. The step and include? methods require the begin object to implement succ or to be numeric.

In the xs class below both <=> and succ are implemented so xs can be used to construct ranges. Note that the <u>Comparable</u> module is included so the == method is defined in terms of <=>.

```
class Xs
  include Comparable
  attr :length
  def initialize(n)
   @length = n
  end
  def succ
   Xs.new(@length + 1)
  end
  def <=>(other)
   @length <=> other.length
  end
  def to_s
    sprintf "%2d #{inspect}", @length
  end
  def inspect
    'x' * @length
  end
```

end 

An example of using xs to construct a range:



## In Files

📄 range.c

## Parent

**Object** 

## **Included Modules**

Enumerable

## **Public Class Methods**

Some provide the end object; otherwise, it will be excluded.

## **Public Instance Methods**
#### on rng == obj → true or false

Returns true only if obj is a <u>Range</u>, has equivalent begin and end items (by comparing them with ==), and has the same <u>exclude\_end?</u> setting as the range.



#### on rng === obj → true or false

Returns true if obj is an element of the range, false otherwise. Conveniently, === is the comparison operator used by case statements.

```
case 79
when 1..50 then print "low\n"
when 51..75 then print "medium\n"
when 76..100 then print "high\n"
end
```

produces:

```
high
```

#### Solution → obj

Returns the object that defines the beginning of the range.



#### bsearch {|obj| block } → value

By using binary search, finds a value in range which meets the given condition in O(log n) where n is the size of the range.

You can use this method in two use cases: a findminimum mode and a find-any mode. In either case, the elements of the range must be monotone (or sorted) with respect to the block.

In find-minimum mode (this is a good choice for typical use case), the block must return true or false, and there must be a value x so that:

- the block returns false for any value which is less than x, and
- the block returns true for any value which is greater than or equal to x.

If x is within the range, this method returns the value x. Otherwise, it returns nil.



In find-any mode (this behaves like libc's bsearch(3)), the block must return a number, and there must be two values x and y (x  $\leq$  y) so that:

- the block returns a positive number for v if v < x,</p>
- The block returns zero for v if  $x \le v \le y$ , and
- the block returns a negative number for v if y <= v.</p>

This method returns any value which is within the intersection of the given range and x...y (if any). If there is no value that satisfies the condition, it returns nil.



You must not mix the two modes at a time; the block must always return either true/false, or always return a number. It is undefined which value is actually picked up at each iteration.

#### over?(obj) → true or false

Returns true if obj is between the begin and end of the range.

This tests begin <= obj <= end when <u>exclude\_end?</u> is false and begin <= obj < end when <u>exclude\_end?</u> is true.



### each {| i | block } → rng each → an\_enumerator

Iterates over the elements of range, passing each in turn to the block.

The each method can only be used if the begin object of the range supports the succ method. A <u>TypeError</u> is raised if the object does not have succ method defined (like Float).

If no block is given, an enumerator is returned instead.



### $\odot$ end $\rightarrow$ obj

Returns the object that defines the end of the range.

(1..10).end #=> 10 (1...10).end #=> 10

### eql?(obj) → true or false

Returns true only if obj is a <u>Range</u>, has equivalent begin and end items (by comparing them with eq1?), and has the same <u>exclude\_end?</u> setting as the range.



#### ⊚ exclude\_end? → true or false

Returns true if the range excludes its end value.



## ⊚ first → obj

#### $\odot$ first(n) $\rightarrow$ an\_array

Returns the first object in the range, or an array of the first n elements.

#### 

Compute a hash-code for this range. Two ranges with equal begin and end points (using eq1?), and the same <u>exclude\_end?</u> value will generate the same hash-code.

See also Object#hash.

#### 

Returns true if obj is an element of the range, false otherwise. If begin and end are numeric, comparison is done according to the magnitude of the values.

#### inspect → string

Convert this range object to a printable form (using inspect to convert the begin and end objects).

#### 

Returns the last object in the range, or an array of the last n elements.

Note that with no arguments last will return the object that defines the end of the range even if exclude\_end? is true.



```
⊚ max → obj
```

- max {| a,b | block } → obj
- max(n) → obj
- $max(n) \{| a,b | block \} \rightarrow obj$

Returns the maximum value in the range. Returns nil if the begin value of the range larger than the end value.

Can be given an optional block to override the default comparison method a  $\leq b$ .



#### rightarrow member?(obj) → true or false rightarrow include?(obj) → true or false

Returns true if obj is an element of the range, false otherwise. If begin and end are numeric, comparison is done according to the magnitude of the values.



#### min → obj

- $_{\odot}$  min {| a,b | block }  $\rightarrow$  obj
- $min(n) \rightarrow array$
- $min(n) \{| a,b | block \} \rightarrow array$

Returns the minimum value in the range. Returns nil if the begin value of the range is larger than the end value.

Can be given an optional block to override the default comparison method a  $\leq b$ .

#### (10..20).min #=> 10

#### size → num

Returns the number of elements in the range. Both the begin and the end of the <u>Range</u> must be Numeric, otherwise nil is returned.



#### $rightarrow step(n=1) {| obj | block } → rng$ $<math>rightarrow step(n=1) → an_enumerator$

Iterates over the range, passing each nth element to the block. If begin and end are numeric, n is added for each iteration. Otherwise step invokes succ to iterate through range elements.

If no block is given, an enumerator is returned instead.

```
range = Xs.new(1)..Xs.new(10)
range.step(2) {|x| puts x}
puts
range.step(3) {|x| puts x}
```

produces:

1 3 5 7 9	x xxx xxxx xxxxx xxxxxx xxxxxxx
1 4 7 10	x xxxx xxxxxxx xxxxxxxx xxxxxxxx

See Range for the definition of class Xs.

#### os to\_s → string

Convert this range object to a printable form (using to\_s to convert the begin and end objects).

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class RangeError

Raised when a given numerical value is out of range.

```
[1, 2, 3].drop(1 << 100)
```

raises the exception:

RangeError: bignum too big to convert into `lo

In Files

Parent

StandardError

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class Rational

A rational number can be represented as a paired integer number; a/b (b>0). Where a is numerator and b is denominator. Integer a equals rational a/1 mathematically.

In ruby, you can create rational object with <u>Rational</u>, <u>#to\_r</u>, rationalize method or suffixing r to a literal. The return values will be irreducible.



You can also create rational object from floatingpoint numbers or strings.



A rational object is an exact number, which helps you to write program without any rounding errors.



However, when an expression has inexact factor (numerical value or operation), will produce an inexact result.



# In Files

📄 rational.c

## Parent

Numeric

## **Public Instance Methods**

#### on at \* numeric → numeric

Performs multiplication.

Rational(2, 3)	*	Rational(2, 3)	
Rational(900)	*	Rational(-9, 2)	
Rational(9, 8)	*	4	
Rational(20, 9)	*	9.8	
<b>▲</b>			

#### on rat \*\* numeric → numeric

Performs exponentiation.

Rational(2)	* *	Rational(3)	
Rational(10)	* *	-2	
Rational(10)	* *	-2.0	
Rational(-4)	* *	Rational(1,2)	
Rational(1, 2)	* *	Θ	
Rational(1, 2)	* *	0.0	
<b>I</b>			Þ

#### on at + numeric → numeric

Performs addition.



#### on rat - numeric → numeric

Performs subtraction.

Rational(2, 3) Rational(900) Rational(-2, 9) Rational(9, 8) Rational(20, 9)	- Rational(2, 3) - Rational(1) - Rational(-9, 2) - 4 - 9.8	
<b>↓</b>		•

#### rat / numeric → numeric rat / numeric → numericrat / numeric

Performs division.

Rational(2, 3)	1	Rational(2, 3)	
Rational(900)	/	Rational(1)	
Rational(-2, 9)	/	Rational(-9, 2)	
Rational(9, 8)	7	4	



#### $_{\odot}$ rational <=> numeric $\rightarrow$ -1, 0, +1 or nil

Performs comparison and returns -1, 0, or +1.

nil is returned if the two values are incomparable.



#### $_{\odot}$ rat == object $\rightarrow$ true or false

Returns true if rat equals object numerically.

Rational(2, 3)	==	Rational(2,	3)	
Rational(5)	==	5		
Rational(0)	==	0.0		
Rational('1/3')	==	0.33		
Rational('1/2')	==	'1/2'		

#### $\odot$ ceil $\rightarrow$ integer

#### $\odot$ ceil(precision=0) $\rightarrow$ rational

Returns the truncated value (toward positive infinity).



#### og denominator → integer

Returns the denominator (always positive).

Rational(7).denominator	
Rational(7, 1).denominator	
Rational(9, -4).denominator	
Rational(-2, -10).denominator	
<pre>rat.numerator.gcd(rat.denominator)</pre>	

#### fdiv(numeric) → float

Performs division and returns the value as a float.



## $\odot$ floor $\rightarrow$ integer

#### $_{\odot}$ floor(precision=0) $\rightarrow$ rational

Returns the truncated value (toward negative infinity).

Rational(3).floor       #=> 3         Rational(2, 3).floor       #=> 0         Rational(-3, 2).floor       #=> -1
decimal - 1 2 3 . 4 5 6
precision -3 -2 -1 0 +1 +2
'%f' % Rational('-123.456').floor(+1) #=> "-123 '%f' % Rational('-123.456').floor(-1) #=> "-130

#### inspect → string

Returns the value as a string for inspection.



#### ommerator → integer

Returns the numerator.

Rational(7).numerator	
Rational(7, 1).numerator	
Rational(9, -4).numerator	
Rational(-2, -10).numerator	

on rat / numeric → numeric

og quo(numeric) → numeric

Performs division.



#### $_{\odot}$ rationalize $\rightarrow$ self $_{\odot}$ rationalize(eps) $\rightarrow$ rational

Returns a simpler approximation of the value if the optional argument eps is given (rat-|eps| <= result <= rat+|eps|), self otherwise.



### ound → integer

#### $\odot$ round(precision=0) $\rightarrow$ rational

Returns the truncated value (toward the nearest integer;  $0.5 \Rightarrow 1$ ;  $-0.5 \Rightarrow -1$ ).



#### $\odot$ to\_f $\rightarrow$ float

Return the value as a float.



#### $\odot$ to\_i $\rightarrow$ integer

Returns the truncated value as an integer.

Equivalent to

rat.truncate.
Rational(2, 3).to\_i #=> 0
Rational(3).to\_i #=> 3
Rational(300.6).to\_i #=> 300
Rational(98,71).to\_i #=> 1
Rational(-30,2).to\_i #=> -15

## to\_r → self Deturne colf

Returns self.

Rational(2).to_r	
Rational(-8, 6).to_r	

#### $\odot$ to\_s $\rightarrow$ string

Returns the value as a string.

Rational(2).to_s	
Rational(-8, 6).to_s	
Rational('1/2').to_s	

#### $_{\odot}$ truncate → integer $_{\odot}$ truncate(precision=0) → rational

Returns the truncated value (toward zero).



Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class Rational::compatible

## In Files

rational.c

## Parent

Object

Generated by <u>RDoc</u> 3.12.2. Generated with the Darkfish Rdoc Generator 3.

## class Regexp

A Regexp holds a regular expression, used to match a pattern against strings. Regexps are created using the /.../ and %r{...} literals, and by the Regexp::new constructor.

Regular expressions (*regexps*) are patterns which describe the contents of a string. They're used for testing whether a string contains a given pattern, or extracting the portions that match. They are created with the */pat/* and %r {*pat*} literals or the Regexp.new constructor.

A regexp is usually delimited with forward slashes (/). For example:

If a string contains the pattern it is said to *match*. A literal string matches itself.

Here 'haystack' does not contain the pattern 'needle', so it doesn't match:

```
/needle/.match('haystack') #=> nil
```

Here 'haystack' contains the pattern 'hay', so it matches:



Specifically, /st/ requires that the string contains the letter *s* followed by the letter *t*, so it matches *haystack*, also.

## =~ and <u>#match</u>

Pattern matching may be achieved by using =~ operator or  $\frac{\#match}{match}$  method.

#### =~ operator

=~ is Ruby's basic pattern-matching operator. When one operand is a regular expression and the other is a string then the regular expression is used as a pattern to match against the string. (This operator is equivalently defined by Regexp and <u>String</u> so the order of <u>String</u> and <u>Regexp</u> do not matter. Other classes may have different implementations of =~.) If a match is found, the operator returns index of first match in string, otherwise it returns nil.

Using =~ operator with a <u>String</u> and <u>Regexp</u> the \$~ global variable is set after a successful match. \$~ holds a <u>MatchData</u> object. <u>::last\_match</u> is equivalent to \$~.

#### <u>#match</u> method

The match method returns a MatchData object:



## **Metacharacters and Escapes**

The following are *metacharacters* (, ), [, ], {, }, ., ?, +, \*. They have a specific meaning when appearing in a pattern. To match them literally they must be backslash-escaped. To match a backslash literally backslash-escape that: <tt>\\ </tt>

Patterns behave like double-quoted strings so can contain the same backslash escapes.

```
/\s\u{6771 4eac 90fd}/.match("Go to 東京都")
#=> #<MatchData " 東京都">
```

Arbitrary Ruby expressions can be embedded into patterns with the  $#{\ldots}$  construct.



## **Character Classes**

A character class is delimited with square brackets ([, ]) and lists characters that may appear at that point in the match. /[ab]/ means a or b, as opposed to /ab/ which means a followed by b.

```
/W[aeiou]rd/.match("Word") #=> #<MatchData "W
</pre>
```

Within a character class the hyphen (-) is a metacharacter denoting an inclusive range of characters. [abcd] is equivalent to [a-d]. A range can be followed by another range, so [abcdwxyz] is equivalent to [a-dw-z]. The order in which ranges or individual characters appear inside a character class is irrelevant.

If the first character of a character class is a caret (^) the class is inverted: it matches any character *except* those named.

A character class may contain another character class. By itself this isn't useful because [a-z[0-9]] describes the same set as [a-z0-9].

However, character classes also support the && operator which performs set intersection on its arguments. The two can be combined as follows:

This is equivalent to:

/[abh-w]/

The following metacharacters also behave like character classes:

- /./ Any character except a newline.
- /./m Any character (the m modifier enables multiline mode)
- /\w/ A word character ([a-zA-z0-9\_])
- /\w/ A non-word character ([^a-zA-z0-9\_]).
   Please take a look at <u>Bug #4044</u> if using /\w/ with the /i modifier.
- /\d/ A digit character ([0-9])
- /\D/ A non-digit character ([^0-9])
- /\h/ A hexdigit character ([0-9a-fA-F])
- /\H/ A non-hexdigit character ([^0-9a-fA-F])
- /\s/ A whitespace character: /[ \t\r\n\f]/
- /\s/ A non-whitespace character: /[^ \t\r\n\f]/

POSIX *bracket expressions* are also similar to character classes. They provide a portable alternative to the above, with the added benefit that they encompass non-ASCII characters. For instance, /\d/ matches only the ASCII decimal digits (0-9); whereas /[[:digit:]]/ matches any character in the Unicode *Nd* category.

- /[[:alnum:]]/ Alphabetic and numeric character
- /[[:alpha:]]/ Alphabetic character
- /[[:blank:]]/ Space or tab
- /[[:cntrl:]]/ Control character
- /[[:digit:]]/ Digit
- /[[:graph:]]/ Non-blank character (excludes spaces, control characters, and similar)
- /[[:lower:]]/ Lowercase alphabetical character
- /[[:print:]]/ Like [:graph:], but includes the space character
- /[[:punct:]]/ Punctuation character
- /[[:space:]]/ Whitespace character
   ([:blank:], newline, carriage return, etc.)
- /[[:upper:]]/ Uppercase alphabetical
- /[[:xdigit:]]/ Digit allowed in a hexadecimal number (i.e., 0-9a-fA-F)

Ruby also supports the following non-POSIX character classes:

- /[[:word:]]/ A character in one of the following Unicode general categories Letter, Mark, Number, Connector\_Punctuation
- /[[:ascii:]]/ A character in the ASCII character set



## Repetition

The constructs described so far match a single character. They can be followed by a repetition metacharacter to specify how many times they need to occur. Such metacharacters are called *quantifiers*.

- \* Zero or more times
- + One or more times
- ? Zero or one times (optional)
- {*n*} Exactly *n* times
- {*n*, } *n* or more times
- {,*m*} *m* or less times
- $\{n, m\}$  At least *n* and at most *m* times

At least one uppercase character ('H'), at least one lowercase character ('e'), two 'l' characters, then one 'o':



Repetition is *greedy* by default: as many occurrences as possible are matched while still allowing the overall match to succeed. By contrast, *lazy* matching makes the minimal amount of matches necessary for overall success. A greedy metacharacter can be made lazy by following it with ?. Both patterns below match the string. The first uses a greedy quantifier so '.+' matches '<a> <b>'; the second uses a lazy quantifier so '.+?' matches '<a>':



A quantifier followed by + matches *possessively*: once it has matched it does not backtrack. They behave like greedy quantifiers, but having matched they refuse to "give up" their match even if this jeopardises the overall match.

## Capturing

Parentheses can be used for *capturing*. The text enclosed by the *n*<sup>th</sup> group of parentheses can be subsequently referred to with *n*. Within a pattern use the *backreference* \n; outside of the pattern use MatchData[n].

'at' is captured by the first group of parentheses, then referred to later with 1:



<u>#match</u> returns a <u>MatchData</u> object which makes the captured text available with its #[] method:

Capture groups can be referred to by name when defined with the (?<*name*>) or (?'*name*') constructs.



Named groups can be backreferenced with \k<*name*>, where *name* is the group name.



**Note**: A regexp can't use named backreferences and numbered backreferences simultaneously.

When named capture groups are used with a literal regexp on the left-hand side of an expression and the =~ operator, the captured text is also assigned to local variables with corresponding names.



## Grouping

Parentheses also *group* the terms they enclose, allowing them to be quantified as one *atomic* whole.

The pattern below matches a vowel followed by 2 word characters:



Whereas the following pattern matches a vowel followed by a word character, twice, i.e. [aeiou]\w[aeiou]\w: 'enor'.



The (?:...) construct provides grouping without capturing. That is, it combines the terms it contains into an atomic whole without creating a backreference. This benefits performance at the slight expense of readability.

The first group of parentheses captures 'n' and the second 'ti'. The second group is referred to later with the backreference  $\2$ :



The first group of parentheses is now made noncapturing with '?:', so it still matches 'n', but doesn't create the backreference. Thus, the backreference \1 now refers to 'ti'.



#### **Atomic Grouping**

Grouping can be made *atomic* with (?>*pat*). This causes the subexpression *pat* to be matched independently of the rest of the expression such that what it matches becomes fixed for the remainder of the match, unless the entire subexpression must be abandoned and subsequently revisited. In this way *pat* is treated as a non-divisible whole. Atomic grouping is typically used to optimise patterns so as to prevent the regular expression engine from backtracking needlessly.

The " in the pattern below matches the first character of the string, then .\* matches *Quote*". This causes the overall match to fail, so the text matched by .\* is backtracked by one position, which leaves the final character of the string available to match "



If .\* is grouped atomically, it refuses to backtrack *Quote"*, even though this means that the overall match fails



## **Subexpression Calls**

The \g<name> syntax matches the previous subexpression named *name*, which can be a group name or number, again. This differs from backreferences in that it re-executes the group rather than simply trying to re-match the same text.

This pattern matches a ( character and assigns it to the paren group, tries to call that the paren sub-expression again but fails, then matches a literal ):



- 1. Matches at the beginning of the string, i.e. before the first character.
- 2. Enters a named capture group called paren
- 3. Matches a literal (, the first character in the string

- 4. Calls the paren group again, i.e. recurses back to the second step
- 5. Re-enters the paren group
- 6. Matches a literal (, the second character in the string
- Try to call paren a third time, but fail because doing so would prevent an overall successful match
- 8. Match a literal ), the third character in the string. Marks the end of the second recursive call
- 9. Match a literal ), the fourth character in the string
- 10. Match the end of the string
## Alternation

The vertical bar metacharacter (|) combines two expressions into a single one that matches either of the expressions. Each expression is an *alternative*.



## **Character Properties**

The  $p{}$  construct matches characters with the named property, much like POSIX bracket classes.

- /\p{Alnum}/ Alphabetic and numeric character
- /\p{Alpha}/ Alphabetic character
- /\p{Blank}/ Space or tab
- /\p{Cntrl}/ Control character
- /\p{Digit}/ Digit
- /\p{Graph}/ Non-blank character (excludes spaces, control characters, and similar)
- /\p{Lower}/ Lowercase alphabetical character
- /\p{Print}/ Like \p{Graph}, but includes the space character
- /\p{Punct}/ Punctuation character
- /\p{Space}/ Whitespace character ([:blank:], newline, carriage return, etc.)
- /\p{Upper}/ Uppercase alphabetical
- /\p{XDigit}/ Digit allowed in a hexadecimal number (i.e., 0-9a-fA-F)
- /\p{Word}/ A member of one of the following Unicode general category Letter, Mark, Number, Connector\_Punctuation

- /\p{ASCII}/ A character in the ASCII character set
- /\p{Any}/ Any Unicode character (including unassigned characters)
- /\p{Assigned}/ An assigned character

A Unicode character's *General Category* value can also be matched with  $p{Ab}$  where *Ab* is the category's abbreviation as described below:

- /\p{L}/ 'Letter'
- /\p{L1}/ 'Letter: Lowercase'
- /\p{Lm}/ 'Letter: Mark'
- /\p{Lo}/ 'Letter: Other'
- /\p{Lt}/ 'Letter: Titlecase'
- /\p{Lu}/ 'Letter: Uppercase
- /\p{Lo}/ 'Letter: Other'
- /\p{M}/ 'Mark'
- /\p{Mn}/ 'Mark: Nonspacing'
- /\p{Mc}/ 'Mark: Spacing Combining'
- /\p{Me}/ 'Mark: Enclosing'
- /\p{N}/ 'Number'
- /\p{Nd}/ 'Number: Decimal Digit'
- /\p{N1}/ 'Number: Letter'
- /\p{No}/ 'Number: Other'
- /\p{P}/ 'Punctuation'
- /\p{Pc}/ 'Punctuation: Connector'

- /\p{Pd}/ 'Punctuation: Dash'
- /\p{Ps}/ 'Punctuation: Open'
- /\p{Pe}/ 'Punctuation: Close'
- /\p{Pi}/ 'Punctuation: Initial Quote'
- /\p{Pf}/ 'Punctuation: Final Quote'
- /\p{Po}/ 'Punctuation: Other'
- /\p{s}/ 'Symbol'
- /\p{sm}/ 'Symbol: Math'
- /\p{sc}/ 'Symbol: Currency'
- /\p{sc}/ 'Symbol: Currency'
- /\p{sk}/ 'Symbol: Modifier'
- /\p{so}/ 'Symbol: Other'
- /\p{z}/ 'Separator'
- /\p{zs}/ 'Separator: Space'
- /\p{z1}/ 'Separator: Line'
- /\p{zp}/ 'Separator: Paragraph'
- /\p{C}/ 'Other'
- /\p{Cc}/ 'Other: Control'
- /\p{Cf}/ 'Other: Format'
- /\p{cn}/ 'Other: Not Assigned'
- /\p{Co}/ 'Other: Private Use'
- /\p{Cs}/ 'Other: Surrogate'

Lastly, \p{} matches a character's Unicode script. The following scripts are supported: Arabic, Armenian, Balinese, Bengali, Bopomofo, Braille, Buginese, Buhid, Canadian\_Aboriginal, Carian, Cham, Cherokee, Common, Coptic, Cuneiform, Cypriot, Cyrillic, Deseret, Devanagari, Ethiopic, Georgian, Glagolitic, Gothic, Greek, Gujarati, Gurmukhi, Han, Hangul, Hanunoo, Hebrew, Hiragana, Inherited, Kannada, Katakana, Kayah\_Li, Kharoshthi, Khmer, Lao, Latin, Lepcha, Limbu, Linear\_B, Lycian, Lydian, Malayalam, Mongolian, Myanmar, New\_Tai\_Lue, Nko, Ogham, Ol\_Chiki, Old\_Italic, Old\_Persian, Oriya, Osmanya, Phags\_Pa, Phoenician, Rejang, Runic, Saurashtra, Shavian, Sinhala, Sundanese, Syloti\_Nagri, Syriac, Tagalog, Tagbanwa, Tai\_Le, Tamil, Telugu, Thaana, Thai, Tibetan, Tifinagh, Ugaritic, Vai, and Yi.

Unicode codepoint U+06E9 is named "ARABIC PLACE OF SAJDAH" and belongs to the Arabic script:

All character properties can be inverted by prefixing their name with a caret (^).

Letter 'A' is not in the Unicode LI (Letter; Lowercase) category, so this match succeeds:

## Anchors

Anchors are metacharacter that match the zerowidth positions between characters, *anchoring* the match to a specific position.

- ^ Matches beginning of line
- \$ Matches end of line
- \A Matches beginning of string.
- \z Matches end of string. If string ends with a newline, it matches just before newline
- \z Matches end of string
- \G Matches point where last match finished
- \b Matches word boundaries when outside brackets; backspace (0x08) when inside brackets
- \B Matches non-word boundaries
- (?=pat) Positive lookahead assertion: ensures that the following characters match pat, but doesn't include those characters in the matched text
- (?!pat) Negative lookahead assertion: ensures that the following characters do not match pat, but doesn't include those characters in the matched text
- (?<=pat) Positive lookbehind assertion: ensures that the preceding characters match

*pat*, but doesn't include those characters in the matched text

 (?<!pat) - Negative lookbehind assertion: ensures that the preceding characters do not match pat, but doesn't include those characters in the matched text

If a pattern isn't anchored it can begin at any point in the string:

Anchoring the pattern to the beginning of the string forces the match to start there. 'real' doesn't occur at the beginning of the string, so now the match fails:

```
/\Areal/.match("surrealist") #=> nil
```

The match below fails because although 'Demand' contains 'and', the pattern does not occur at a word boundary.

```
/\band/.match("Demand")
```

Whereas in the following example 'and' has been anchored to a non-word boundary so instead of matching the first 'and' it matches from the fourth letter of 'demand' instead:



The pattern below uses positive lookahead and positive lookbehind to match text appearing in tags without including the tags in the match:

F



## Options

The end delimiter for a regexp can be followed by one or more single-letter options which control how the pattern can match.

- /pat/i Ignore case
- /pat/m Treat a newline as a character matched by .
- /pat/x Ignore whitespace and comments in the pattern
- /pat/o Perform #{} interpolation only once

i, m, and x can also be applied on the subexpression level with the (?on-off) construct, which enables options *on*, and disables options *off* for the expression enclosed by the parentheses.

Options may also be used with Regexp.new:

F F F	Regexp.new("abc", Regexp.new("abc", Regexp.new("abc # Regexp.new("abc",	Regexp::IGNORECASE) Regexp::MULTILINE) Comment", Regexp::EXTENDED) Regexp::IGNORECASE   Regexp
•		•

## **Free-Spacing Mode and Comments**

As mentioned above, the × option enables *free-spacing* mode. Literal white space inside the pattern is ignored, and the octothorpe (#) character introduces a comment until the end of the line. This allows the components of the pattern to be organized in a potentially more readable fashion.

A contrived pattern to match a number with optional decimal places:



There are a number of strategies for matching whitespace:

- Use a pattern such as \s or \p{Space}.
- Use escaped whitespace such as \ , i.e. a space preceded by a backslash.
- Use a character class such as [ ].

Comments can be included in a non-x pattern with the (?#comment) construct, where

*comment* is arbitrary text ignored by the regexp engine.

## Encoding

Regular expressions are assumed to use the source encoding. This can be overridden with one of the following modifiers.

- /pat/u UTF-8
- /pat/e EUC-JP
- /pat/s Windows-31J
- /pat/n ASCII-8BIT

A regexp can be matched against a string when they either share an encoding, or the regexp's encoding is *US-ASCII* and the string's encoding is ASCII-compatible.

If a match between incompatible encodings is attempted an Encoding::CompatibilityError exception is raised.

The Regexp#fixed\_encoding? predicate indicates whether the regexp has a *fixed* encoding, that is one incompatible with ASCII. A regexp's encoding can be explicitly fixed by supplying Regexp::FIXEDENCODING as the second argument Of Regexp.new:



## **Special global variables**

Pattern matching sets some global variables :

- \$~ is equivalent to <u>::last\_match;</u>
- \$& contains the complete matched text;
- \$` contains string before match;
- \$' contains string after match;
- \$1, \$2 and so on contain text matching first, second, etc capture group;
- \$+ contains last capture group.

Example:



These global variables are thread-local and method-local variables.

## Performance

Certain pathological combinations of constructs can lead to abysmally bad performance.

Consider a string of 25 *a*s, a *d*, 4 *a*s, and a *c*.

The following patterns match instantly as you would expect:

However, the following pattern takes appreciably longer:

#### /(b|a+)\*c/ =~ s **#=> 26**

This happens because an atom in the regexp is quantified by both an immediate + and an enclosing \* with nothing to differentiate which is in control of any particular character. The nondeterminism that results produces superlinear performance. (Consult *Mastering Regular Expressions* (3rd ed.), pp 222, by *Jeffery Friedl*, for an in-depth analysis). This particular case can be fixed by use of atomic grouping, which prevents the unnecessary backtracking:



A similar case is typified by the following example, which takes approximately 60 seconds to execute for me:

Match a string of 29 as against a pattern of 29 optional as followed by 29 mandatory as:

```
Regexp.new('a?' * 29 + 'a' * 29) =~ 'a' * 29
```

The 29 optional *a*s match the string, but this prevents the 29 mandatory *a*s that follow from matching. Ruby must then backtrack repeatedly so as to satisfy as many of the optional matches as it can while still matching the mandatory 29. It is plain to us that none of the optional matches can succeed, but this fact unfortunately eludes Ruby.

The best way to improve performance is to significantly reduce the amount of backtracking needed. For this case, instead of individually matching 29 optional as, a range of optional as can be matched all at once with  $a\{0,29\}$ :

Regexp.new('a{0,29}' + 'a' \* 29) =~ 'a' \* 29

## **In Files**

📄 re.c

## Parent

**Object** 

## Constants

#### EXTENDED

see *#options* and ::new

#### FIXEDENCODING

see *#options* and ::new

#### IGNORECASE

see #options and ::new

#### MULTILINE

see #options and ::new

#### NOENCODING

see <u>#options</u> and <u>::new</u>

## **Public Class Methods**

## Synonym for Regexp.new

#### 

Escapes any characters that would have special meaning in a regular expression. Returns a new escaped string, or self if no characters are escaped. For any string, Regexp.new(Regexp.escape(*str*))=~*str* will be true.

Regexp.escape('\\*?{}.') #=> \\\\*\?\{\}\.

# $_{\odot}$ last\_match → matchdata $_{\odot}$ last\_match(n) → str

The first form returns the <u>MatchData</u> object generated by the last successful pattern match. Equivalent to reading the special global variable \$~ (see Special global variables in Regexp for details).

The second form returns the *n*th field in this <u>MatchData</u> object. *n* can be a string or symbol to reference a named capture.

Note that the <u>::last\_match</u> is local to the thread and method scope of the method that did the pattern match.



 $_{\odot}$  new(string, [options [, kcode]]) → regexp

#### onew(regexp) → regexp

 $_{\odot}$  compile(string, [options [, kcode]]) → regexp  $_{\odot}$  compile(regexp) → regexp

Constructs a new regular expression from pattern, which can be either a <u>String</u> or a <u>Regexp</u> (in which case that regexp's options are propagated), and new options may not be specified (a change as of Ruby 1.8).

If options is a <u>Fixnum</u>, it should be one or more of the constants Regexp::EXTENDED,

Regexp::IGNORECASE, and Regexp::MULTILINE, or-ed together. Otherwise, if options is not nil or false, the regexp will be case insensitive.

When the kcode parameter is `n' or `N' sets the regexp no encoding. It means that the regexp is for binary strings.



## $_{\odot}$ escape(str) $\rightarrow$ string

#### on quote(str) → string

Escapes any characters that would have special meaning in a regular expression. Returns a new escaped string, or self if no characters are escaped. For any string, Regexp.new(Regexp.escape(str))=~str will be true.

Regexp.escape('\\*?{}.') #=> \\\\*\?\{\}\.

### ⊚ try\_convert(obj) → re or nil

Try to convert *obj* into a <u>Regexp</u>, using to\_regexp method. Returns converted regexp or nil if *obj* cannot be converted for any reason.



#### $_{\odot}$ union(pat1, pat2, ...) → new\_regexp $_{\odot}$ union(pats\_ary) → new\_regexp

Return a Regexp object that is the union of the given *patterns*, i.e., will match any of its parts. The *patterns* can be <u>Regexp</u> objects, in which case their options will be preserved, or Strings. If no patterns are given, returns /(?!)/. The behavior is unspecified if any given *pattern* contains capture.



Note: the arguments for <u>::union</u> will try to be converted into a regular expression literal via to\_regexp.

## **Public Instance Methods**

 $rxp == other_rxp → true or false$  $<math>rxp = other_rxp → true or false$  Equality—Two regexps are equal if their patterns are identical, they have the same character set code, and their casefold? values are the same.



#### $rxp == str \rightarrow true or false$

Case Equality—Used in case statements.

```
a = "HELLO"
case a
when /^[a-z]*$/; print "Lower case\n"
when /^[A-Z]*$/; print "Upper case\n"
else; print "Mixed case\n"
end
#=> "Upper case"
```

Following a regular expression literal with the #=== operator allows you to compare against a String.

/^[a-z]\*\$/ === "HELLO" #=> false /^[A-Z]\*\$/ === "HELLO" #=> true

#### $_{\odot}$ rxp =~ str $\rightarrow$ integer or nil

Match—Matches rxp against str.



If =~ is used with a regexp literal with named captures, captured strings (or nil) is assigned to local variables named by the capture names.



If it is not matched, nil is assigned for the variables.



This assignment is implemented in the Ruby parser. The parser detects 'regexp-literal =~ expression' for the assignment. The regexp must be a literal without interpolation and placed at left hand side.

The assignment does not occur if the regexp is not a literal.



A regexp interpolation, #{}, also disables the assignment.

```
rhs_pat = /(?<rhs>\w+)/
/(?<lhs>\w+)\s*=\s*#{rhs_pat}/ =~ "x = y"
p lhs  # undefined local variable
```

The assignment does not occur if the regexp is placed at the right hand side.

```
" x = y " =~ /(?<lhs>\w+)\s*=\s*(?<rhs>\w+)/
p lhs, rhs # undefined local variable
```

#### $\odot$ casefold? $\rightarrow$ true or false

Returns the value of the case-insensitive flag.



 $_{\odot}$  encoding  $\rightarrow$  encoding

Returns the Encoding object that represents the encoding of obj.

#### $rxp == other_rxp → true or false$ $<math>rxp = other_rxp → true or false$

Equality—Two regexps are equal if their patterns are identical, they have the same character set code, and their casefold? values are the same.



#### ixed\_encoding? → true or false

Returns false if rxp is applicable to a string with any ASCII compatible encoding. Returns true otherwise.



#### 

Produce a hash based on the text and options of this regular expression.

See also Object#hash.

#### inspect → string

Produce a nicely formatted string-version of *rxp*. Perhaps surprisingly, #inspect actually produces the more natural version of the string than #to\_s.

/ab+c/x.inspect #=> "/ab+c/ix"

# $match(str) \rightarrow matchdata or nil match(str,pos) \rightarrow matchdata or nil$

Returns a MatchData object describing the match, or nil if there was no match. This is equivalent to retrieving the value of the special variable \$~ following a normal match. If the second parameter is present, it specifies the position in the string to begin the search.

If a block is given, invoke the block with <u>MatchData</u> if match succeed, so that you can write

```
pat.match(str) {|m| ...}
```

#### instead of

```
if m = pat.match(str)
    ...
end
```

The return value is a value from block execution in

this case.

#### on named\_captures → hash

Returns a hash representing information about named captures of *rxp*.

A key of the hash is a name of the named captures. A value of the hash is an array which is list of indexes of corresponding named captures.



If there are no named captures, an empty hash is returned.

```
/(.)(.)/.named_captures 
#=> {}
```

### $_{\odot}$ names $\rightarrow$ [name1, name2, ...]

Returns a list of names of captures as an array of strings.



#### options → fixnum

Returns the set of bits corresponding to the options used when creating this Regexp (see Regexp::new for

details. Note that additional bits may be set in the returned options: these are used internally by the regular expression code. These extra bits are ignored if the options are passed to Regexp::new.



#### Source → str

Returns the original string of the pattern.

/ab+c/x.source #=> "ab+c"

Note that escape sequences are retained as is.

#### ⊚to\_s → str

Returns a string containing the regular expression and its options (using the (?opts:source) notation. This string can be fed back in to Regexp::new to a regular expression with the same semantics as the original. (However, Regexp#== may not return true when comparing the two, as the source of the regular expression itself may differ, as the example shows). Regexp#inspect produces a generally more readable version of *rxp*.





#### $_{\odot}$ ~ rxp $\rightarrow$ integer or nil

Match—Matches *rxp* against the contents of  $\_$ . Equivalent to *rxp* =~  $\_$ .

\$\_ = "input data" ~ /at/ #=> 7

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class RegexpError

Raised when given an invalid regexp expression.

Regexp.new("?")

raises the exception:

RegexpError: target of repeat operator is not

In Files

Parent

StandardError

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class Ripper

## **In Files**

📄 parse.c

Parent

Object

## Constants

#### Version

version of Ripper

## **Public Class Methods**

# new(src, filename="(ripper)", lineno=1) $\rightarrow$ $_{\odot}$ ripper

Create a new <u>Ripper</u> object. *src* must be a <u>String</u>, an <u>IO</u>, or an <u>Object</u> which has gets method.

This method does not starts parsing. See also  $\frac{\text{#parse}}{\text{#parse}}$  and  $\frac{\text{#parse}}{\text{#parse}}$ .

## **Public Instance Methods**

#### os ripper#column → Integer

Return column number of current parsing line. This number starts from 0.

#### $\odot$ ripper#encoding $\rightarrow$ encoding

Return encoding of the source.

#### ipper#end\_seen? → Boolean

Return true if parsed source ended by +\_\_END\_\_+.

#### ipper#error? → Boolean

Return true if parsed source has errors.

#### ◎ ripper#filename → String

Return current parsing filename.

#### os ripper#lineno → Integer

Return line number of current parsing line. This number starts from 1.

#### ripper#parse

Start parsing and returns the value of the root action.

#### ⊚yydebug → true or false

Get yydebug.

#### 🐵 yydebug = flag

Set yydebug.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class RubyVM

::RubyVM

## **In Files**

📄 iseq.c

📄 vm.c

## Parent

Object

## Constants

#### DEFAULT\_PARAMS

**DEFAULT\_PARAMS** This constant variable shows VM's default parameters. Note that changing these values does not affect VM execution. Specification is not stable and you should not depend on this value. Of course, this constant is MRI specific.

#### **INSTRUCTION\_NAMES**

INSTRUCTION\_NAMES

#### OPTS

OPTS, which shows vm build options

## **Public Class Methods**

```
⊚stat → Hash
```

```
stat(hsh) → hsh
```

#### stat(Symbol) → Numeric

Returns a <u>Hash</u> containing implementationdependent counters inside the VM.

This hash includes information about method/constant cache serials:



The contents of the hash are implementation specific and may be changed in the future.

This method is only expected to work on C Ruby.

```
Generated by <u>RDoc</u> 3.12.2.
Generated with the <u>Darkfish Rdoc Generator</u> 3.
```

## class RubyVM::Env

::RubyVM::Env

## In Files

iseq.c

### Parent

Object

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class RubyVM::InstructionSequen

The InstructionSequence class represents a compiled sequence of instructions for the Ruby Virtual Machine.

With it, you can get a handle to the instructions that make up a method or a proc, compile strings of Ruby code down to VM instructions, and disassemble instruction sequences to strings for easy inspection. It is mostly useful if you want to learn how the Ruby VM works, but it also lets you control various settings for the Ruby iseq compiler.

You can find the source for the VM instructions in insns.def in the Ruby source.

The instruction sequence results will almost certainly change as Ruby changes, so example output in this documentation may be different from what you see.

## **In Files**

📄 iseq.c

### Parent
#### Object

## **Public Class Methods**

Takes source, a <u>String</u> of Ruby code and compiles it to an InstructionSequence.

Optionally takes file, path, and line which describe the filename, absolute path and first line number of the ruby code in source which are metadata attached to the returned iseq.

options, which can be true, false or a Hash, is used to modify the default behavior of the Ruby iseq compiler.

For details regarding valid compile options see ::compile\_option=.



## compile\_file(file[, options]) → iseq

Takes file, a <u>String</u> with the location of a Ruby source file, reads, parses and compiles the file, and returns iseq, the compiled <u>InstructionSequence</u> with source location metadata set.

Optionally takes options, which can be true, false or a Hash, to modify the default behavior of the Ruby iseq compiler.

For details regarding valid compile options see ::compile\_option=.



## compile\_option → options

Returns a hash of default options used by the Ruby iseq compiler.

For details, see ::compile\_option=.

### compile\_option = options

Sets the default values for various optimizations in the Ruby iseq compiler.

Possible values for options include true, which enables all options, false which disables all options, and nil which leaves all options unchanged.

You can also pass a Hash of options that you want to change, any options not present in the hash will be left unchanged.

Possible option names (which are keys in options) which can be set to true or false include:

- inline\_const\_cache
- instructions\_unification
- :operands\_unification
- :peephole\_optimization
- specialized\_instruction

- :stack\_caching
- :tailcall\_optimization
- :trace\_instruction

Additionally, :debug\_level can be set to an integer.

These default options can be overwritten for a single run of the iseq compiler by passing any of the above values as the options parameter to <u>::new</u>, <u>::compile</u> and <u>::compile\_file</u>.

#### 

Takes body, a <u>Method</u> or <u>Proc</u> object, and returns a String with the human readable instructions for body.

For a Method object:



Produces:

== dis	sasm: <rubyvm::ir< th=""><th>nstructionSequence:hello@/tr</th></rubyvm::ir<>	nstructionSequence:hello@/tr
0000	trace	8
0002 1	trace	1
0004	putself	
0005	putstring	"hello, world"
0007 9	send	:puts, 1, nil, 8, <ic:0></ic:0>
0013	trace	16
0015	leave	
4		•

For a Proc:

```
# /tmp/proc.rb
p = proc { num = 1 + 2 }
```

#### puts RubyVM::InstructionSequence.disasm(p)

Produces:

```
== disasm: <RubyVM::InstructionSequence:block in
== catch table
| catch type: next _____st: 0000 ed: 0012 sp: 0000 c
local table (size: 2, argc: 0 [opts: 0, rest: -1,
[ 2] num
0000 trace
                  1
0002 putobject
                 1
0004 putobject
                  2
                 <ic:1>
0006 opt_plus
0008 dup
0009 setlocal num, 0
0012 leave
4
```

# omodeline of the second se

Takes body, a <u>Method</u> or <u>Proc</u> object, and returns a String with the human readable instructions for body.

For a Method object:

```
# /tmp/method.rb
def hello
   puts "hello, world"
end
puts RubyVM::InstructionSequence.disasm(method(:h
]
```

**Produces:** 

== di	lsasm: <rubyvm::in< th=""><th>nstructionSequence:hello@/tr</th></rubyvm::in<>	nstructionSequence:hello@/tr
0000	trace	8
0002	trace	1
0004	putself	
0005	putstring	"hello, world"
0007	send	:puts, 1, nil, 8, <ic:0></ic:0>
0013	trace	16

0015 leave

For a Proc:

•

```
# /tmp/proc.rb
p = proc { num = 1 + 2 }
puts RubyVM::InstructionSequence.disasm(p)
```

#### Produces:

<pre>== disasm: <rubyvm::i =="catch" catch="" next<="" pre="" redo="" table="" type:=""  =""></rubyvm::i></pre>	InstructionSequence:block in st: 0000 ed: 0012 sp: 0000 o st: 0000 ed: 0012 sp: 0000 o
local table (size: 2,	argc: 0 [opts: 0, rest: -1,
[ 2] num	
0000 trace	1
0002 putobject	1
0004 putobject	2
0006 opt_plus	<ic:1></ic:1>
0008 dup	
0009 setlocal	num, O
0012 leave	
4	•

## compile(source[, file[, path[, line[,

#### options]]]) → iseq

# new(source[, file[, path[, line[, options]]]]) $\rightarrow$ $_{\odot}$ iseq

Takes source, a <u>String</u> of Ruby code and compiles it to an <u>InstructionSequence</u>.

Optionally takes file, path, and line which describe the filename, absolute path and first line number of the ruby code in source which are metadata attached to the returned iseq.

options, which can be true, false or a Hash, is used to modify the default behavior of the Ruby iseq

compiler.

For details regarding valid compile options see ::compile\_option=.



## 🏐 of(p1)

Returns the instruction sequence containing the given proc or method.

For example, using irb:



Using ::compile\_file:





# **Public Instance Methods**

### absolute\_path()

Returns the absolute path of this instruction sequence.

nil if the iseq was evaluated from a string.

For example, using ::compile\_file:



## base\_label()

Returns the base label of this instruction sequence.

For example, using irb:



Using ::compile\_file:





#### $\odot$ disasm $\rightarrow$ str $\odot$ disassemble $\rightarrow$ str

Returns the instruction sequence as a string in human readable form.



Produces:

== di	isasm: <rubyvm::i< th=""><th>nstructionSequence:<compile< th=""></compile<></th></rubyvm::i<>	nstructionSequence: <compile< th=""></compile<>
0000	trace	1
0002	putobject	1
0004	putobject	2
0006	opt_plus	<ic:1></ic:1>
0008	leave	
4		

## o disasm → str

#### o disassemble → str

Returns the instruction sequence as a string in human readable form.



Produces:

== disasm: <rubyvm< th=""><th>::InstructionSequence:<compiled< th=""></compiled<></th></rubyvm<>	::InstructionSequence: <compiled< th=""></compiled<>
0000 trace	1
0002 putobject	1
0004 putobject	2
0006 opt_plus	<ic:1></ic:1>
0008 leave	
•	· · · · · · · · · · · · · · · · · · ·

## obj eval → obj

Evaluates the instruction sequence and returns the result.



## first\_lineno()

Returns the number of the first source line where the instruction sequence was loaded from.

For example, using irb:



## inspect()

Returns a human-readable string representation of this instruction sequence, including the <u>label</u> and <u>path</u>.

## 🚳 label()

Returns the label of this instruction sequence.

<main> if it's at the top level, <compiled> if it was evaluated from a string.

For example, using irb:



Using ::compile\_file:



## line\_trace\_all()

Experimental MRI specific feature, only available as C level api.

Returns all specified\_line events.

## line\_trace\_specify(p1, p2)

Experimental MRI specific feature, only available as C level api.

Set a specified\_line event at the given line position, if the set parameter is true.

This method is useful for building a debugger breakpoint at a specific line.

A TypeError is raised if set is not boolean.

If pos is a negative integer a <u>TypeError</u> exception is raised.

## 🐵 path()

Returns the path of this instruction sequence.

<compiled> if the iseq was evaluated from a string.

For example, using irb:



Using ::compile\_file:



#### osto\_a → ary

Returns an <u>Array</u> with 14 elements representing the instruction sequence with the following data:

#### magic

A string identifying the data format. **Always** YARVInstructionSequence/SimpleDataFormat.

#### major\_version

The major version of the instruction sequence.

#### minor\_version

The minor version of the instruction sequence.

#### format\_type

A number identifying the data format. **Always 1**.

misc

A hash containing:

:arg\_size

the total number of arguments taken by the method or the block (0 if *iseq* doesn't represent a method or block)

:local\_size

the number of local variables + 1

:stack\_max

used in calculating the stack depth at which a SystemStackError is thrown.

#### label

The name of the context (block, method, class, module, etc.) that this instruction sequence belongs to.

<main> if it's at the top level, <compiled> if it was evaluated from a string.

#### path

The relative path to the Ruby file where the instruction sequence was loaded from.

<compiled> if the iseq was evaluated from a string.

## absolute\_path

The absolute path to the Ruby file where the instruction sequence was loaded from.

nil if the iseq was evaluated from a string.

#### first\_lineno

The number of the first source line where the instruction sequence was loaded from.

#### type

The type of the instruction sequence.

Valid values are :top, :method, :block,

:class, :rescue, :ensure, :eval, :main, and :defined\_guard.

#### locals

An array containing the names of all arguments and local variables as symbols.

#### params

An <u>Hash</u> object containing parameter information.

More info about these values can be found in  $\ensuremath{\mathsf{vm\_core.h.}}$ 

#### catch\_table

A list of exceptions and control flow operators (rescue, next, redo, break, etc.).

#### bytecode

An array of arrays containing the instruction names and operands that make up the body of the instruction sequence.

Note that this format is MRI specific and version dependent.

#### Generated by <u>RDoc</u> 3.12.2.

Generated with the Darkfish Rdoc Generator 3.

# class RuntimeError

A generic error class raised when an invalid operation is attempted.

```
[1, 2, 3].freeze << 4
```

raises the exception:

```
RuntimeError: can't modify frozen Array
```

Kernel#raise will raise a RuntimeError if no Exception class is specified.

```
raise "ouch"
```

raises the exception:

RuntimeError: ouch

# **In Files**

error.c

## Parent

StandardError

Generated by RDoc 3.12.2.

Generated with the Darkfish Rdoc Generator 3.

# class ScriptError

ScriptError is the superclass for errors raised when a script can not be executed because of a LoadError, NotImplementedError Of a SyntaxError. Note these type of ScriptErrors are not StandardError and will not be rescued unless it is specified explicitly (or its ancestor Exception).

# **In Files**

error.c

## Parent

Exception

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class SecurityError

Raised when attempting a potential unsafe operation, typically when the \$SAFE level is raised above 0.

```
foo = "bar"
proc = Proc.new do
   $SAFE = 3
   foo.untaint
end
proc.call
```

raises the exception:



In Files

error.c

Parent

Exception

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# module Signal

Many operating systems allow signals to be sent to running processes. Some signals have a defined effect on the process, while others may be trapped at the code level and acted upon. For example, your process may trap the USR1 signal and use it to toggle debugging, and may use TERM to initiate a controlled shutdown.

```
pid = fork do
  Signal.trap("USR1") do
    $debug = !$debug
    puts "Debug now: #$debug"
  end
  Signal.trap("TERM") do
    puts "Terminating..."
    shutdown()
  end
  # . . . do some work . . .
end
Process.detach(pid)
# Controlling program:
Process.kill("USR1", pid)
# ...
Process.kill("USR1", pid)
# ...
Process.kill("TERM", pid)
```

produces:

```
Debug now: true
Debug now: false
```

#### Terminating...

The list of available signal names and their interpretation is system dependent. <u>Signal</u> delivery semantics may also vary between systems; in particular signal delivery may not always be reliable.

## **In Files**

📄 signal.c

# **Public Class Methods**

## osist → a\_hash

Returns a list of signal names mapped to the corresponding underlying signal numbers.



## signame(signo) → string

convert signal number to signal name



produces:

INT

 $_{\odot}$  trap( signal, command )  $\rightarrow$  obj

## $_{\odot}$ trap( signal ) {| | block } $\rightarrow$ obj

Specifies the handling of signals. The first parameter is a signal name (a string such as "SIGALRM", "SIGUSR1", and so on) or a signal number. The characters "SIG" may be omitted from the signal name. The command or block specifies code to be run when the signal is raised. If the command is the string "IGNORE" or "SIG IGN", the signal will be ignored. If the command is "DEFAULT" or "SIG DFL", the Ruby's default handler will be invoked. If the command is "EXIT", the script will be terminated by the signal. If the command is "SYSTEM DEFAULT", the operating system's default handler will be invoked. Otherwise, the given command or block will be run. The special signal name "EXIT" or signal number zero will be invoked just prior to program termination. trap returns the previous handler for the given signal.

Signal.trap(0, proc { puts "Terminating: #{\$\$}"
Signal.trap("CLD") { puts "Child died" }
fork && Process.wait

produces:

Terminating: 27461 Child died Terminating: 27460

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class SignalException

Raised when a signal is received.



produces:

received Exception SIGHUP

In Files

error.c

📄 signal.c

## Parent

Exception

# **Public Class Methods**

new(sig\_name) → signal\_exception
 new(sig\_number [, name]) →
 signal\_exception

Construct a new <u>SignalException</u> object. sig\_name should be a known signal name.

# **Public Instance Methods**

isigno → num Returns a signal number.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class StandardError

The most standard error types are subclasses of <u>StandardError</u>. A rescue clause without an explicit <u>Exception</u> class will rescue all StandardErrors (and only those).



On the other hand:

require 'does/not/exist' rescue "Hi"

raises the exception:



# **In Files**

error.c

# Parent

Exception

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class StopIteration

Raised to stop the iteration, in particular by Enumerator#next. It is rescued by Kernel#loop.

```
loop do
   puts "Hello"
   raise StopIteration
   puts "World"
end
puts "Done!"
```

produces:

Hello Done!

In Files

enumerator.c

Parent

IndexError

# **Public Instance Methods**

#### oresult → value

Returns the return value of the iterator.

o = Object.new

def o.each	
yield 1	
yield 2	
yield 3	
100	
end	
e = o.to_enum	
puts e.next	
puts e.next	
puts e.next	
begin	
e.next	
rescue Stopiteration => ex	
puts ex.result	
ena	

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class String

A string object holds and manipulates an arbitrary sequence of bytes, typically representing characters. <u>String</u> objects may be created using string::new or as literals.

Because of aliasing issues, users of strings should be aware of the methods that modify the contents of a string object. Typically, methods with names ending in "!" modify their receiver, while those without a "!" return a new string. However, there are exceptions, such as string# []=.

## **In Files**

- complex.c
- 📄 pack.c
- 📄 rational.c
- string.c
- ltranscode.c

## Parent

Object

# **Included Modules**

Comparable

## **Public Class Methods**

## onew(str="") → new\_str

Returns a new string object containing a copy of str.

#### onvert(obj) → string or nil

Try to convert *obj* into a <u>String</u>, using <u>#to\_str</u> method. Returns converted string or nil if *obj* cannot be converted for any reason.

```
String.try_convert("str")
String.try_convert(/re/)
```

# **Public Instance Methods**

#### ostr % arg → new\_str

Format—Uses *str* as a format specification, and returns the result of applying it to *arg*. If the format specification contains more than one substitution, then *arg* must be an Array or Hash containing the values to be substituted. See Kernel::sprintf for details of the format string.



#### ostr \* integer → new\_str

Copy — Returns a new <u>String</u> containing integer copies of the receiver. integer must be greater than or equal to 0.



#### ostr + other\_str → new\_str

Concatenation—Returns a new string containing *other\_str* concatenated to *str*.



- ostr << integer → str</p>
- concat(integer) → str
- ⊚ str << obj → str
- oncat(obj) → str

Append—Concatenates the given object to *str*. If the object is a Integer, it is considered as a codepoint, and is converted to a character before concatenation.



## $_{\odot}$ string <=> other\_string $\rightarrow$ -1, 0, +1 or nil

Comparison—Returns -1, 0, +1 or nil depending on whether string is less than, equal to, or greater than other\_string.

nil is returned if the two values are incomparable.

If the strings are of different lengths, and the strings are equal when compared up to the shortest length, then the longer string is considered greater than the shorter one.

<=> is the basis for the methods <, <=, >, >=, and

between?, included from module <u>Comparable</u>. The method String#== does not use Comparable#==.



```
_{\odot} str == obj → true or false
_{\odot} str === obj → true or false
```

## Equality

Returns whether str == obj, similar to Object#==.

If obj is not an instance of <u>String</u> but responds to to\_str, then the two strings are compared using case equality Object#===.

Otherwise, returns similarly to <u>#eql?</u>, comparing length and content.

str == obj → true or false
 str === obj → true or false

## Equality

Returns whether str == obj, similar to Object#==.

If obj is not an instance of <u>String</u> but responds to to\_str, then the two strings are compared using case equality Object#===.

Otherwise, returns similarly to <u>#eql?</u>, comparing length and content.

#### ostr =~ obj → fixnum or nil

Match—If *obj* is a Regexp, use it as a pattern to match against *str*,and returns the position the match starts, or nil if there is no match. Otherwise, invokes *obj.*=~, passing *str* as an argument. The default =~ in object returns nil.

Note: str =~ regexp is not the same as regexp =~ str. Strings captured from named capture groups are assigned to local variables only in the second case.

```
"cat o' 9 tails" =~ /\d/ #=> 7
"cat o' 9 tails" =~ 9 #=> nil
```

```
str[index] → new_str or nil
\otimes str[start, length] \rightarrow new_str or nil
rightarrow str[range] \rightarrow new_str or nil
 str[regexp] \rightarrow new str or nil
\implies str[regexp, capture] \rightarrow new str or nil
str[match str] → new str or nil
slice(index) → new_str or nil
\otimes slice(start, length) \rightarrow new_str or nil
rightarrow slice(range) \rightarrow new str or nil
slice(regexp) → new_str or nil
\otimes slice(regexp, capture) \rightarrow new str or nil
Slice(match str) → new str or nil
  Element Reference — If passed a single index,
  returns a substring of one character at that index. If
  passed a start index and a length, returns a
  substring containing length characters starting at the
  index. If passed a range, its beginning and end are
  interpreted as offsets delimiting the substring to be
  returned.
```

In these three cases, if an index is negative, it is

counted from the end of the string. For the start and range cases the starting index is just before a character and an index matching the string's size. Additionally, an empty string is returned when the starting index for a character range is at the end of the string.

Returns nil if the initial index falls outside the string or the length is negative.

If a Regexp is supplied, the matching portion of the string is returned. If a capture follows the regular expression, which may be a capture group index or name, follows the regular expression that component of the MatchData is returned instead.

If a match\_str is given, that string is returned if it occurs in the string.

Returns nil if the regular expression does not match or the match string cannot be found.

a = "hello there"	
a[1] a[2, 3] a[23]	
a[-3, 2] a[72] a[-42] a[-24]	
a[11, 0] a[11] a[12, 0] a[121]	
a[/[aeiou](.)\1/] a[/[aeiou](.)\1/, 0] a[/[aeiou](.)\1/, 1] a[/[aeiou](.)\1/, 2]	



- str[fixnum] = new\_str
- str[fixnum, fixnum] = new\_str
- str[range] = aString
- str[regexp] = new\_str
- str[regexp, fixnum] = new\_str
- str[regexp, name] = new\_str

#### str[other\_str] = new\_str

Element Assignment—Replaces some or all of the content of *str*. The portion of the string affected is determined using the same criteria as string#[]. If the replacement string is not the same length as the text it is replacing, the string will be adjusted accordingly. If the regular expression or string is used as the index doesn't match a position in the string, IndexError is raised. If the regular expression form is used, the optional second Fixnum allows you to specify which portion of the match to replace (effectively using the MatchData indexing rules. The forms that take a Fixnum will raise an IndexError if the value is out of range; the Range form will raise a RangeError, and the Regexp and string will raise an IndexError on negative match.

## ascii\_only? → true or false

Returns true for a string which has only ASCII characters.

"abc".force\_encoding("UTF-8").ascii\_only?

## ⊚b → str

Returns a copied string whose encoding is ASCII-8BIT.

## ⓑ bytes → an\_array

Returns an array of bytes in *str*. This is a shorthand for str.each\_byte.to\_a.

If a block is given, which is a deprecated form, works the same as each\_byte.

## bytesize → integer integer

Returns the length of str in bytes.

- byteslice(fixnum) → new\_str or nil
- byteslice(fixnum, fixnum) → new\_str or nil

#### $_{\odot}$ byteslice(range) $\rightarrow$ new\_str or nil

Byte Reference—If passed a single Fixnum, returns a substring of one byte at that position. If passed two Fixnum objects, returns a substring starting at the offset given by the first, and a length given by the second. If given a Range, a substring containing bytes at offsets given by the range is returned. In all three cases, if an offset is negative, it is counted from the end of *str*. Returns nil if the initial offset falls outside the string, the length is negative, or the beginning of the range is greater than the end. The encoding of

the resulted string keeps original encoding.



#### $\odot$ capitalize $\rightarrow$ new\_str

Returns a copy of *str* with the first character converted to uppercase and the remainder to lowercase. Note: case conversion is effective only in ASCII region.

"hello".capitalize	
"HELLO".capitalize	
"123ABC".capitalize	

## 

Modifies *str* by converting the first character to uppercase and the remainder to lowercase. Returns nil if no changes are made. Note: case conversion is effective only in ASCII region.



# $_{\odot}$ casecmp(other\_str) → -1, 0, +1 or nil

Case-insensitive version of string#<=>.



### center(width, padstr=' ') → new\_str

Centers str in width. If width is greater than the length of str, returns a new <u>String</u> of length width with str centered and padded with padstr; otherwise, returns str.



#### rightarrow chars $\rightarrow$ an\_array

Returns an array of characters in *str*. This is a shorthand for str.each\_char.to\_a.

If a block is given, which is a deprecated form, works the same as each\_char.

#### $\odot$ chomp(separator=\$/) $\rightarrow$ new\_str

Returns a new string with the given record separator removed from the end of *str* (if present). If \$/ has not been changed from the default Ruby record separator, then chomp also removes carriage return characters (that is it will remove \n, \r, and \r\n). If \$/ is an empty string, it will remove all trailing newlines from the string.

"hello".chomp	
"hello\n".chomp	
"hello\r\n".chomp	
"hello\n\r".chomp	
"hello\r".chomp	
"hello \n there".chomp	
"hello".chomp("llo")	
"hello\r\n\r\n".chomp('')	
"hello\r\n\r\r\n".chomp('')	
## $_{\odot}$ chomp!(separator=\$/) $\rightarrow$ str or nil

Modifies *str* in place as described for *string#chomp*, returning *str*, or nil if no modifications were made.

#### $\odot$ chop $\rightarrow$ new\_str

Returns a new string with the last character removed. If the string ends with \r\n, both characters are removed. Applying chop to an empty string returns an empty string. string#chomp is often a safer alternative, as it leaves the string unchanged if it doesn't end in a record separator.



## ohop! → str or nil

Processes *str* as for *string#chop*, returning *str*, or nil if *str* is the empty string. See also *string#chomp*!.

## ⇔ chr → string

Returns a one-character string at the beginning of the string.



## 

Makes string empty.

a = "abcde"

#### codepoints → an\_array

Returns an array of the Integer ordinals of the characters in *str*. This is a shorthand for str.each\_codepoint.to\_a.

If a block is given, which is a deprecated form, works the same as each\_codepoint.

```
    str << integer → str</li>
    concat(integer) → str
    str << obj → str</li>
```

#### concat(obj) → str

Append—Concatenates the given object to *str*. If the object is a Integer, it is considered as a codepoint, and is converted to a character before concatenation.



## count([other\_str]+) → fixnum

Each other\_str parameter defines a set of characters to count. The intersection of these sets defines the characters to count in str. Any other\_str that starts with a caret ^ is negated. The sequence c1-c2 means all characters between c1 and c2. The backslash character \</code> can be used to escape <code>^ or - and is otherwise ignored unless it appears at the end of a sequence or the end of a other\_str.

```
a = "hello world"
a.count "lo"
```



## orypt(salt\_str) → new\_str

Applies a one-way cryptographic hash to *str* by invoking the standard library function crypt(3) with the given salt string. While the format and the result are system and implementation dependent, using a salt matching the regular expression \A[a-zA-Z0-9./] {2} should be valid and safe on any platform, in which only the first two characters are significant.

This method is for use in system specific scripts, so if you want a cross-platform hash function consider using Digest or OpenSSL instead.

## objective ([other\_str]+) → new\_str

Returns a copy of *str* with all characters in the intersection of its arguments deleted. Uses the same rules for building the set of characters as String#count.

```
"hello".delete "l","lo"  #=> "heo"
"hello".delete "lo"  #=> "he"
"hello".delete "aeiou", "^e"  #=> "hell"
"hello".delete "ej-m"  #=> "ho"
```

```
objective delete!([other_str]+) → str or nil
```

Performs a delete operation in place, returning *str*, or nil if *str* was not modified.

#### owncase → new\_str

Returns a copy of *str* with all uppercase letters replaced with their lowercase counterparts. The operation is locale insensitive—only characters "A" to "Z" are affected. Note: case replacement is effective only in ASCII region.

"hEllO".downcase #=> "hello"

#### $\odot$ downcase! $\rightarrow$ str or nil

Downcases the contents of *str*, returning nil if no changes were made. Note: case replacement is effective only in ASCII region.

#### odump → new\_str

Produces a version of str with all non-printing characters replaced by \nnn notation and all special characters escaped.

"hello \n ''".dump #=> "\"hello \\n ''\"

## $● each_byte {|fixnum| block } → str$ $<math> ● each_byte → an_enumerator$

Passes each byte in *str* to the given block, or returns an enumerator if no block is given.

```
"hello".each_byte {|c| print c, ' ' }
```

produces:



## each\_char {|cstr| block } → str each\_char → an\_enumerator

Passes each character in *str* to the given block, or returns an enumerator if no block is given.

```
"hello".each_char {|c| print c, ' ' }
```

produces:

h e l l o

## each\_codepoint {|integer| block } → str each\_codepoint → an\_enumerator

Passes the Integer ordinal of each character in *str*, also known as a *codepoint* when applied to Unicode strings to the given block.

If no block is given, an enumerator is returned instead.



produces:

104 101 108 108 111 1593

# each\_line(separator=\$/) {|substr| block } $\rightarrow \$ str

Splits str using the supplied parameter as the record separator (\$/ by default), passing each substring in turn to the supplied block. If a zero-length record separator is supplied, the string is split into paragraphs delimited by multiple successive newlines.

If no block is given, an enumerator is returned instead.

```
print "Example one\n"
"hello\nworld".each_line {|s| p s}
print "Example two\n"
"hello\nworld".each_line('l') {|s| p s}
print "Example three\n"
"hello\n\n\nworld".each_line('') {|s| p s}
```

produces:

Example one
"hello\n"
"world"
Example two
"hel"
"1"
"o\nworl"
"d"
Example three
"hello\n\n\n"
"world"

## empty? → true or false

Returns true if str has a length of zero.

"hello".empty?	
" ".empty?	
"".empty?	

- encode(encoding [, options]) → str
   encode(dst\_encoding, src\_encoding [,
   options]) → str
- encode([options]) → str

The first form returns a copy of str transcoded to encoding encoding. The second form returns a copy

of str transcoded from src\_encoding to dst\_encoding. The last form returns a copy of str transcoded to Encoding.default\_internal.

By default, the first and second form raise Encoding::UndefinedConversionError for characters that are undefined in the destination encoding, and Encoding::InvalidByteSequenceError for invalid byte sequences in the source encoding. The last form by default does not raise exceptions but uses replacement strings.

The options <u>Hash</u> gives details for conversion and can have the following keys:

#### :invalid

If the value is :replace, <u>encode</u> replaces invalid byte sequences in str with the replacement character. The default is to raise the <u>Encoding::InvalidByteSequenceError</u> exception

#### :undef

If the value is :replace, <u>encode</u> replaces characters which are undefined in the destination encoding with the replacement character. The default is to raise the Encoding::UndefinedConversionError.

#### :replace

Sets the replacement string to the given value. The default replacement string is "uFFFD" for Unicode encoding forms, and "?" otherwise.

#### :fallback

Sets the replacement string by the given object for undefined character. The object should be a <u>Hash</u>, a <u>Proc</u>, a <u>Method</u>, or an object which has [] method. Its key is an undefined character encoded in the source encoding of current transcoder. Its value can be any encoding until it can be converted into the destination encoding of the transcoder.

#### :xml

The value must be :text or :attr. If the value is :text <u>encode</u> replaces undefined characters with their (upper-case hexadecimal) numeric character references. '&', '<', and '>' are converted to "&", "<", and "&gt;", respectively. If the value is :attr, <u>encode</u> also quotes the replacement result (using ""), and replaces "" with """.

#### :cr\_newline

Replaces LF ("n") with CR ("r") if value is true.

#### :crlf\_newline

Replaces LF ("n") with CRLF ("rn") if value is true.

#### :universal\_newline

Replaces CRLF ("rn") and CR ("r") with LF ("n") if value is true.

# a encode!(encoding [, options]) → str encode!(dst\_encoding, src\_encoding [, a options]) → str

The first form transcodes the contents of *str* from str.encoding to encoding. The second form transcodes the contents of *str* from src\_encoding to dst\_encoding. The options <u>Hash</u> gives details for conversion. See <u>#encode</u> for details. Returns the string even if no changes were made.

## $_{\odot}$ encoding $\rightarrow$ encoding

Returns the <u>Encoding</u> object that represents the encoding of obj.

## end\_with?([suffixes]+) → true or false

Returns true if str ends with one of the suffixes given.

## $_{\odot}$ eql?(other) $\rightarrow$ true or false

Two strings are equal if they have the same length and content.

#### on force\_encoding(encoding) → str

Changes the encoding to encoding and returns self.

## $_{\odot}$ getbyte(index) $\rightarrow$ 0 .. 255

returns the *index*th byte as an integer.

#### ogsub(pattern, replacement) → new\_str

- $\odot$  gsub(pattern, hash)  $\rightarrow$  new\_str
- $\odot$  gsub(pattern) {|match| block }  $\rightarrow$  new\_str

gsub(pattern) → enumerator

Returns a copy of *str* with the *all* occurrences of *pattern* substituted for the second argument. The *pattern* is typically a Regexp; if given as a string, any regular expression metacharacters it contains will be interpreted literally, e.g. '\\d' will match a backlash followed by 'd', instead of a digit.

If *replacement* is a string it will be substituted for the matched text. It may contain back-references to the pattern's capture groups of the form  $\d$ , where *d* is a group number, or  $\k<n>$ , where *n* is a group name. If it is a double-quoted string, both back-references must be preceded by an additional backslash.

However, within *replacement* the special match variables, such as \$&, will not refer to the current match.

If the second argument is a Hash, and the matched text is one of its keys, the corresponding value is the replacement string.

In the block form, the current match string is passed in as a parameter, and variables such as \$1, \$2, \$`, \$&, and \$' will be set appropriately. The value returned by the block will be substituted for the match on each call.

The result inherits any tainting in the original string or any supplied replacement string.

When neither a block nor a second argument is supplied, an Enumerator is returned.



Performs the substitutions of String#gsub in place, returning *str*, or nil if no substitutions were performed. If no block and no *replacement* is given, an enumerator is returned instead.

## 

Return a hash based on the string's length, content and encoding.

See also Object#hash.

## $_{\odot}$ hex $\rightarrow$ integer

Treats leading characters from *str* as a string of hexadecimal digits (with an optional sign and an optional 0×) and returns the corresponding number. Zero is returned on error.



## $_{\odot}$ include? other\_str → true or false

Returns true if *str* contains the given string or character.

"hello".include?	"lo"	
"hello".include?	"ol"	
"hello".include?	h	

## ∞ index(substring [, offset]) → fixnum or nil ∞ index(regexp [, offset]) → fixnum or nil

Returns the index of the first occurrence of the given *substring* or pattern (*regexp*) in *str*. Returns nil if not found. If the second parameter is present, it specifies the position in the string to begin the search.



 $_{\odot}$  replace(other\_str)  $\rightarrow$  str

Replaces the contents and taintedness of *str* with the corresponding values in *other\_str*.



## insert(index, other\_str) → str

Inserts *other\_str* before the character at the given *index*, modifying *str*. Negative indices count from the end of the string, and insert *after* the given character. The intent is insert *a*<u>String</u> so that it starts at the given *index*.



## inspect → string

Returns a printable version of *str*, surrounded by quote marks, with special characters escaped.



## intern → symbol

## osym → symbol

Returns the symbol corresponding to *str*, creating the symbol if it did not previously exist. See Symbol#id2name.



s == :@cat

This can also be used to create symbols that cannot be represented using the :xxx notation.

'cat and dog'.to\_sym #=> :"cat and dog"

## length → integer

## integer → integer

Returns the character length of str.

## $\otimes$ lines(separator=\$/) $\rightarrow$ an\_array

Returns an array of lines in *str* split using the supplied record separator (\$/ by default). This is a shorthand for str.each\_line(separator).to\_a.

If a block is given, which is a deprecated form, works the same as each\_line.

## bijust(integer, padstr=' ') → new\_str

If *integer* is greater than the length of *str*, returns a new string of length *integer* with *str* left justified and padded with *padstr*; otherwise, returns *str*.



## 

Returns a copy of *str* with leading whitespace removed. See also string#rstrip and string#strip.

" hello ".lstrip #=> "hello " "hello".lstrip #=> "hello"

## 

Removes leading whitespace from *str*, returning nil if no change was made. See also *string#rstrip*! and *String#strip*!.



## rightarrow match(pattern) → matchdata or nil rightarrow match(pattern, pos) → matchdata or nil

Converts *pattern* to a Regexp (if it isn't already one), then invokes its match method on *str*. If the second parameter is present, it specifies the position in the string to begin the search.



If a block is given, invoke the block with <u>MatchData</u> if match succeed, so that you can write

```
str.match(pat) {|m| ...}
```

instead of



The return value is a value from block execution in this case.

Succ → new\_str
Inext → new\_str

Returns the successor to *str*. The successor is calculated by incrementing characters starting from the rightmost alphanumeric (or the rightmost character if there are no alphanumerics) in the string. Incrementing a digit always results in another digit, and incrementing a letter results in another letter of the same case. Incrementing nonalphanumerics uses the underlying character set's collating sequence.

If the increment generates a "carry," the character to the left of it is incremented. This process repeats until there is no carry, adding an additional character if necessary.



 $\otimes$  succ!  $\rightarrow$  str

#### mext! → str

Equivalent to string#succ, but modifies the receiver in place.

#### $\odot$ oct $\rightarrow$ integer

Treats leading characters of *str* as a string of octal digits (with an optional sign) and returns the corresponding number. Returns 0 if the conversion fails.

"123".oct	
"-377".oct	
"bad".oct	
"0377bad".oct	

## $\odot$ ord $\rightarrow$ integer

Return the Integer ordinal of a one-character string.



## rightarrow partition(sep) → [head, sep, tail] rightarrow partition(regexp) → [head, match, tail]

Searches *sep* or pattern (*regexp*) in the string and returns the part before it, the match, and the part after it. If it is not found, returns two empty strings and *str*.



## ◎ prepend(other\_str) → str

Prepend—Prepend the given string to str.



## oreplace(other\_str) → str

Replaces the contents and taintedness of *str* with the corresponding values in *other\_str*.



## or reverse → new\_str

Returns a new string with the characters from *str* in reverse order.

#### "stressed".reverse #=> "desserts"

is reverse! → str

Reverses str in place.

# rindex(substring [, fixnum]) → fixnum or nil rindex(regexp [, fixnum]) → fixnum or nil

Returns the index of the last occurrence of the given *substring* or pattern (*regexp*) in *str*. Returns nil if not found. If the second parameter is present, it specifies the position in the string to end the search— characters beyond this point will not be considered.



## ost(integer, padstr=' ') → new\_str

If *integer* is greater than the length of *str*, returns a new string of length *integer* with *str* right justified and padded with *padstr*; otherwise, returns *str*.



# ◎ rpartition(sep) → [head, sep, tail] ◎ rpartition(regexp) → [head, match, tail] Searches sep or pattern (regexp) in the string from the str

Searches *sep* or pattern (*regexp*) in the string from the end of the string, and returns the part before it, the match, and the part after it. If it is not found,

returns two empty strings and str.



## or strip → new\_str

Returns a copy of *str* with trailing whitespace removed. See also string#lstrip and string#strip.



#### In the self or nil In th

Removes trailing whitespace from *str*, returning nil if no change was made. See also *string#lstrip!* and *String#strip!*.



## $_{\odot}$ scan(pattern) → array $_{\odot}$ scan(pattern) {|match, ...| block } → str

Both forms iterate through *str*, matching the pattern (which may be a Regexp or a String). For each match, a result is generated and either added to the result array or passed to the block. If the pattern contains no groups, each individual result consists of the matched string, \$&. If the pattern contains groups, each individual result array containing one entry per group.





And the block form:

a.scan(/\w+/) {|w| print "<<#{w}>> " }
print "\n"
a.scan(/(.)(.)/) {|x,y| print y, x }
print "\n"

produces:

<<cruel>> <<world>> rceu lowlr

## Scrub → new\_str

⊚ scrub(repl) → new\_str

## oscrub{|bytes|} → new\_str

If the string is invalid byte sequence then replace invalid bytes with given replacement character, else returns self. If block is given, replace invalid bytes with returned value of the block.



## ⊚ scrub! → str ⊚ scrub!(repl) → str

## oscrub!{|bytes|} → str

If the string is invalid byte sequence then replace invalid bytes with given replacement character, else returns self. If block is given, replace invalid bytes with returned value of the block.

```
"abc\u3042\x81".scrub! #=> "abc\u3042\uFFFD"
"abc\u3042\x81".scrub!("*") #=> "abc\u3042*"
```

Setbyte(index, integer) → integer modifies the *index*th byte as *integer*.

integer
 size → integer
 Returns the character length of str.

4

str[index] → new\_str or nil  $\otimes$  str[start, length]  $\rightarrow$  new\_str or nil  $\odot$  str[range]  $\rightarrow$  new\_str or nil str[regexp] → new\_str or nil str[regexp, capture]  $\rightarrow$  new str or nil str[match str] → new str or nil slice(index) → new str or nil  $\otimes$  slice(start, length)  $\rightarrow$  new\_str or nil slice(range) → new\_str or nil  $\otimes$  slice(regexp)  $\rightarrow$  new\_str or nil  $\otimes$  slice(regexp, capture)  $\rightarrow$  new str or nil Slice(match str) → new str or nil Element Reference — If passed a single index, returns a substring of one character at that index. If passed a start index and a length, returns a substring containing length characters starting at the index. If passed a range, its beginning and end are interpreted as offsets delimiting the substring to be returned.

In these three cases, if an index is negative, it is counted from the end of the string. For the start and

range cases the starting index is just before a character and an index matching the string's size. Additionally, an empty string is returned when the starting index for a character range is at the end of the string.

Returns nil if the initial index falls outside the string or the length is negative.

If a Regexp is supplied, the matching portion of the string is returned. If a capture follows the regular expression, which may be a capture group index or name, follows the regular expression that component of the MatchData is returned instead.

If a match\_str is given, that string is returned if it occurs in the string.

Returns nil if the regular expression does not match or the match string cannot be found.

a = "hello there"	
a[1]	#=> "e"
a[2, 3]	#=> "llo"
a[23]	#=> "ll"
a[-3, 2]	#=> "er"
a[72]	#=> "her"
a[-42]	#=> "her"
a[-24]	#=> ""
a[11, 0]	#=> ""
a[11]	#=> nil
a[12, 0]	#=> nil
a[121]	#=> nil
a[/[aeiou](.)\1/] a[/[aeiou](.)\1/, 0] a[/[aeiou](.)\1/, 1] a[/[aeiou](.)\1/, 2]	<pre>#=&gt; "ell" #=&gt; "ell" #=&gt; "l" #=&gt; nil</pre>
a[/(? <vowel>[aeiou])(? a[/(?<vowel>[aeiou])(?</vowel></vowel>	<pre>P<non_vowel>[^aeiou])/, "nor P<non_vowel>[^aeiou])/, "vow</non_vowel></non_vowel></pre>



```
slice!(fixnum) → new_str or nil
```

- Slice!(fixnum, fixnum) → new\_str or nil
- slice!(range) → new\_str or nil
- slice!(regexp) → new\_str or nil
- slice!(other\_str) → new\_str or nil

Deletes the specified portion from *str*, and returns the portion deleted.



## $_{\odot}$ split(pattern=\$;, [limit]) $\rightarrow$ anArray

Divides *str* into substrings based on a delimiter, returning an array of these substrings.

If *pattern* is a string, then its contents are used as the delimiter when splitting *str*. If *pattern* is a single space, *str* is split on whitespace, with leading whitespace and runs of contiguous whitespace characters ignored.

If *pattern* is a Regexp, *str* is divided where the pattern matches. Whenever the pattern matches a zero-length string, *str* is split into individual characters. If *pattern* contains groups, the respective matches will be returned in the array as well.

If *pattern* is omitted, the value of \$; is used. If \$; is nil (which is the default), *str* is split on whitespace as

if ` ' were specified.

If the *limit* parameter is omitted, trailing null fields are suppressed. If *limit* is a positive number, at most that number of fields will be returned (if *limit* is 1, the entire string is returned as the only entry in an array). If negative, there is no limit to the number of fields returned, and trailing null fields are not suppressed.

When the input str is empty an empty <u>Array</u> is returned as the string is considered to have no fields to split.



## squeeze([other\_str]\*) → new\_str

Builds a set of characters from the *other\_str* parameter(s) using the procedure described for string#count. Returns a new string where runs of the same character that occur in this set are replaced by a single character. If no arguments are given, all runs of identical characters are replaced by a single character.





## squeeze!([other\_str]\*) → str or nil

Squeezes *str* in place, returning either *str*, or nil if no changes were made.

## $_{\odot}$ start\_with?([prefixes]+) → true or false

Returns true if str starts with one of the prefixes given.



## ostrip → new\_str

Returns a copy of *str* with leading and trailing whitespace removed.



## strip! → str or nil

Removes leading and trailing whitespace from *str*. Returns nil if *str* was not altered.

 $_{\odot}$  sub(pattern, replacement)  $\rightarrow$  new\_str

```
_{\odot} sub(pattern, hash) \rightarrow new_str
```

Sub(pattern) {|match| block } → new\_str Returns a copy of str with the *first* occurrence of pattern replaced by the second argument. The pattern is typically a <u>Regexp</u>; if given as a <u>String</u>, any regular expression metacharacters it contains will be interpreted literally, e.g. '\\d' will match a backlash followed by 'd', instead of a digit.

If replacement is a <u>String</u> it will be substituted for the matched text. It may contain back-references to the pattern's capture groups of the form "\d", where *d* is a group number, or "\k<n>", where *n* is a group name. If it is a double-quoted string, both back-references must be preceded by an additional backslash. However, within replacement the special match variables, such as &\$, will not refer to the current match. If replacement is a <u>String</u> that looks like a pattern's capture group but is actaully not a pattern capture group e.g. "\'", then it will have to be preceded by two backslashes like so "\\\".

If the second argument is a <u>Hash</u>, and the matched text is one of its keys, the corresponding value is the replacement string.

In the block form, the current match string is passed in as a parameter, and variables such as \$1, \$2, \$`, \$&, and \$' will be set appropriately. The value returned by the block will be substituted for the match on each call.

The result inherits any tainting in the original string or any supplied replacement string.



 $_{\odot}$  sub!(pattern, replacement)  $\rightarrow$  str or nil

## $_{\odot}$ sub!(pattern) {|match| block } $\rightarrow$ str or nil

Performs the same substitution as **#sub** in-place.

Returns str if a substitution was performed or nil if no substitution was performed.

#### Succ → new\_str

#### mext → new\_str

Returns the successor to *str*. The successor is calculated by incrementing characters starting from the rightmost alphanumeric (or the rightmost character if there are no alphanumerics) in the string. Incrementing a digit always results in another digit, and incrementing a letter results in another letter of the same case. Incrementing nonalphanumerics uses the underlying character set's collating sequence.

If the increment generates a "carry," the character to the left of it is incremented. This process repeats until there is no carry, adding an additional character if necessary.



## $\odot$ succ! $\rightarrow$ str

#### onext! → str

Equivalent to string#succ, but modifies the receiver in place.

## osum(n=16) → integer

Returns a basic *n*-bit checksum of the characters in

*str*, where *n* is the optional Fixnum parameter, defaulting to 16. The result is simply the sum of the binary value of each byte in *str* modulo 2\*\*n - 1. This is not a particularly good checksum.

#### Swapcase → new\_str

Returns a copy of *str* with uppercase alphabetic characters converted to lowercase and lowercase characters converted to uppercase. Note: case conversion is effective only in ASCII region.



#### $_{\odot}$ swapcase! $\rightarrow$ str or nil

Equivalent to string#swapcase, but modifies the receiver in place, returning *str*, or nil if no changes were made. Note: case conversion is effective only in ASCII region.

#### $\odot$ to\_c $\rightarrow$ complex

Returns a complex which denotes the string form. The parser ignores leading whitespaces and trailing garbage. Any digit sequences can be separated by an underscore. Returns zero for null or garbage string.

'9'.to_c	
'2.5'.to_c	
'2.5/1'.to_c	
'-3/2'.to_c	
'-i'.to_c	
'45i'.to_c	
'3-4i'.to_c	
'-4e2-4e-2i'.to_c	
'-0.0-0.0i'.to_c	
'1/2+3/4i'.to_c	

'ruby'.to\_c

See Kernel.Complex.

## $\odot$ to\_f $\rightarrow$ float

Returns the result of interpreting leading characters in *str* as a floating point number. Extraneous characters past the end of a valid number are ignored. If there is not a valid number at the start of *str*, 0.0 is returned. This method never raises an exception.



## $_{\odot}$ to\_i(base=10) $\rightarrow$ integer

Returns the result of interpreting leading characters in *str* as an integer base *base* (between 2 and 36). Extraneous characters past the end of a valid number are ignored. If there is not a valid number at the start of *str*,  $_{0}$  is returned. This method never raises an exception when *base* is valid.

"12345".to_i	
"99 red balloons".to_i	
"0a".to_i	
"0a".to_i(16)	
"hello".to_i	
"1100101".to_i(2)	
"1100101".to_i(8)	
"1100101".to_i(10)	
"1100101".to_i(16)	

## $\odot$ to\_r $\rightarrow$ rational

Returns a rational which denotes the string form. The parser ignores leading whitespaces and trailing

garbage. Any digit sequences can be separated by an underscore. Returns zero for null or garbage string.

NOTE: '0.3'.<u>**#to\_r**</u> isn't the same as 0.3.to\_r. The former is equivalent to '3/10'.<u>**#to\_r**</u>, but the latter isn't so.



See Kernel.Rational.

to\_s → str
 to\_str → str
 Returns the receiver.

to\_s → str
 to\_str → str
 Returns the receiver.

## intern → symbol

## osym → symbol

Returns the symbol corresponding to *str*, creating the symbol if it did not previously exist. See Symbol#id2name.



This can also be used to create symbols that cannot be represented using the :xxx notation.

'cat and dog'.to\_sym #=> :"cat and dog"

## on tr(from\_str, to\_str) → new\_str

Returns a copy of str with the characters in from\_str replaced by the corresponding characters in to\_str. If to\_str is shorter than from\_str, it is padded with its last character in order to maintain the correspondence.



Both strings may use the c1-c2 notation to denote ranges of characters, and from\_str may start with a ^, which denotes all characters except those listed.

The backslash character </code> can be used to escape <code>^ or - and is otherwise ignored unless it appears at the end of a range or the end of the from\_str Or to\_str:



## ostr!(from\_str, to\_str) → str or nil

Translates *str* in place, using the same rules as *string#tr*. Returns *str*, or nil if no changes were made.

## ostr\_s(from\_str, to\_str) → new\_str

Processes a copy of *str* as described under string#tr, then removes duplicate characters in regions that were affected by the translation.



## ostr\_s!(from\_str, to\_str) → str or nil

Performs string#tr\_s processing on *str* in place, returning *str*, or nil if no changes were made.

## on unpack(format) → anArray

Decodes *str* (which may contain binary data) according to the format string, returning an array of each value extracted. The format string consists of a sequence of single-character directives, summarized in the table at the end of this entry. Each directive may be followed by a number, indicating the number of times to repeat with this directive. An asterisk ("\*") will use up all remaining elements. The directives ssiIIL may each be followed by an underscore ("\_") or exclamation mark ("!") to use the underlying platform's native size for the specified type; otherwise, it uses a platform-independent consistent size. Spaces are ignored in the format string. See also Array#pack.

"abc \00\\00aabc \00\\00"".unpack	('A6Z6'	)	
"abc \00\\00"".unpack('a3a3')			
<pre>"abc \00aabc \00"".unpack('Z*Z*')</pre>			
"aa".unpack('b8B8')			
"aaa".unpack('h2H2c')			
"\xfe\xff\xfe\xff".unpack('sS')			
"now=20is".unpack('M*')			
"whole".unpack('xax2aX2aX1aX2a')			
<b>↓</b>			

This table summarizes the various formats and the Ruby classes returned by each.

Integer Directive	Returns	Meaning
C S L Q	Integer Integer Integer Integer	8-bit unsigned (unsigned 16-bit unsigned, native 32-bit unsigned, native 64-bit unsigned, native
c s l q	Integer Integer Integer Integer	8-bit signed (signed cha 16-bit signed, native er 32-bit signed, native er 64-bit signed, native er
S_, S! I, I_, I! L_, L! Q_, Q!	Integer Integer Integer Integer	unsigned short, native e unsigned int, native end unsigned long, native er unsigned long long, nati if the platform has no l (Q_ and Q! is available
s_, s! i, i_, i! l_, l! q_, q!	Integer Integer Integer Integer	signed short, native end signed int, native endia signed long, native endi signed long long, native if the platform has no l (q_ and q! is available
S> L> Q> s> l> q> S!> I!> L!> Q!> s!> i!> l!> q!>	Integer	same as the directives w big endian (available since Ruby 1. "S>" is same as "n" "L>" is same as "N"

S< L< Q< s< l< q< S!< I!< L!< Q!< s!< i!< l!< q!<	Integer     	same as the directives v little endian (available since Ruby 1. "S<" is same as "v" "L<" is same as "V"
n N V V	Integer   Integer   Integer   Integer	16-bit unsigned, network 32-bit unsigned, network 16-bit unsigned, VAX (li 32-bit unsigned, VAX (li
U W	Integer   Integer	UTF-8 character BER-compressed integer (
Float   Directive	   Returns	Meaning
D, d F, f E G g	Float Float Float Float Float Float	double-precision, native single-precision, native double-precision, little single-precision, little double-precision, networ single-precision, networ
String Directive	 Returns	Meaning
A a Z B b H h u M m P p	String String String String String String String String String String String String String	arbitrary binary string arbitrary binary string null-terminated string bit string (MSB first) bit string (LSB first) hex string (high nibble hex string (low nibble 1 UU-encoded string quoted-printable, MIME e base64 encoded string (F base64 encoded string (F pointer to a structure ( pointer to a null-termin
Misc.   Directive	   Returns	Meaning
@ X		skip to the offset giver skip backward one byte



#### oupcase → new\_str

Returns a copy of *str* with all lowercase letters replaced with their uppercase counterparts. The operation is locale insensitive—only characters "a" to "z" are affected. Note: case replacement is effective only in ASCII region.

## oupcase! → str or nil

Upcases the contents of *str*, returning nil if no changes were made. Note: case replacement is effective only in ASCII region.

# upto(other\_str, exclusive=false) {|s| block } $\Rightarrow str$

## upto(other\_str, exclusive=false) $\rightarrow$ an enumerator

Iterates through successive values, starting at *str* and ending at *other\_str* inclusive, passing each value in turn to the block. The string#succ method is used to generate each value. If optional second argument exclusive is omitted or is false, the last value will be included; otherwise it will be excluded.

If no block is given, an enumerator is returned instead.

```
"a8".upto("b6") {|s| print s, ' ' }
for s in "a8".."b6"
    print s, ' '
end
```

produces:

a8 a9 b0 b1 b2 b3 b4 b5 b6 a8 a9 b0 b1 b2 b3 b4 b5 b6

If *str* and *other\_str* contains only ascii numeric characters, both are recognized as decimal numbers. In addition, the width of string (e.g. leading zeros) is handled appropriately.



## $\odot$ valid\_encoding? $\rightarrow$ true or false

Returns true for a string which encoded correctly.



Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class Struct

## In Files

process.c

struct.c

## Parent

**Object** 

## **Included Modules**

Enumerable

## Constants

Tms

## **Public Class Methods**

new([class\_name] [, member\_name]+>)

{|StructClass| block }  $\rightarrow$  StructClass

```
omew(value, ...) → obj
```

↔ StructClass[value, ...] → obj The first two forms are used to create a new Struct
subclass class\_name that can contain a value for each member\_name. This subclass can be used to create instances of the structure like any other Class.

If the class\_name is omitted an anonymous structure class will be created. Otherwise, the name of this struct will appear as a constant in class <u>Struct</u>, so it must be unique for all Structs in the system and must start with a capital letter. Assigning a structure class to a constant also gives the class the name of the constant.



If a block is given it will be evaluated in the context of structClass, passing the created class as a parameter:



This is the recommended way to customize a struct. Subclassing an anonymous struct creates an extra anonymous class that will never be used.

The last two forms create a new instance of a struct subclass. The number of value parameters must be less than or equal to the number of attributes defined for the structure. Unset parameters default to nil. Passing more parameters than number of attributes will raise an ArgumentError.



## **Public Instance Methods**

## $_{\odot}$ struct == other $\rightarrow$ true or false

Equality—Returns true if other has the same struct subclass and has equal member values (according to Object#==).

```
Customer = Struct.new(:name, :address, :zip)
joe = Customer.new("Joe Smith", "123 Maple, Any
joejr = Customer.new("Joe Smith", "123 Maple, Any
jane = Customer.new("Jane Doe", "456 Elm, Anytow
joe == joejr #=> true
joe == jane #=> false
```

## 

Attribute Reference—Returns the value of the given struct member or the member at the given index. Raises <u>NameError</u> if the member does not exist and IndexError if the index is out of range.

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

# struct[name] = obj → obj struct[index] = obj → obj

Attribute Assignment—Sets the value of the given struct member or the member at the given index. Raises <u>NameError</u> if the name does not exist and <u>IndexError</u> if the index is out of range.

## $ⓐ each {|obj| block } → struct$ $ⓐ each → an_enumerator$

Yields the value of each struct member in order. If no block is given an enumerator is returned.



Produces:

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

## each\_pair {|sym, obj| block } → struct each\_pair → an\_enumerator

Yields the name and value of each struct member in order. If no block is given an enumerator is returned.



Produces:

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

## $_{\odot}$ eql?(other) $\rightarrow$ true or false

Hash equality—other and struct refer to the same hash key if they have the same struct subclass and have equal member values (according to Object#eql?).

## 

Returns a hash value based on this struct's contents (see Object#hash).

See also Object#hash.

## os to\_s → string

#### $\odot$ inspect $\rightarrow$ string

Describe the contents of this struct in a string.

Also aliased as: to\_s

## length → fixnum

#### isize → fixnum

Returns the number of struct members.

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

#### members → array

Returns the struct members as an array of symbols:



## select {|i| block } $\rightarrow$ array

#### select → an\_enumerator

Yields each member value from the struct to the block and returns an <u>Array</u> containing the member values from the struct for which the given block returns a true value (equivalent to Enumerable#select).



## $_{\odot}$ length $\rightarrow$ fixnum

#### isize → fixnum

Returns the number of struct members.



# to\_a → array values → array Returns the values for this struct as an <u>Array</u>. Customer = Struct.new(:name, :address, :zip)



## osto\_h → hash

Returns a <u>Hash</u> containing the names and values for the struct's members.

Customer = Struct.new(:name, :address, :zip)
joe = Customer.new("Joe Smith", "123 Maple, Anyto
joe.to\_h[:address] #=> "123 Maple, Anytown NC"
4

to\_s()
 Alias for: inspect

#### ⊚to\_a → array

#### $\otimes$ values $\rightarrow$ array

Returns the values for this struct as an Array.



#### $\odot$ values\_at(selector, ...) $\rightarrow$ an\_array

Returns the struct member values for each selector as an <u>Array</u>. A selector may be either an <u>Integer</u> offset or a <u>Range</u> of offsets (as in <u>Array#values\_at</u>).



Generated by <u>RDoc</u> 3.12.2.

Generated with the Darkfish Rdoc Generator 3.

# class Symbol

Symbol objects represent names and some strings inside the Ruby interpreter. They are generated using the :name and :"string" literals syntax, and by the various to\_sym methods. The same symbol object will be created for a given name or string for the duration of a program's execution, regardless of the context or meaning of that name. Thus if Fred is a constant in one context, a method in another, and a class in a third, the symbol :Fred will be the same object in all three contexts.

## In Files

string.c

## Parent

Object

## **Included Modules**

Comparable

## **Public Class Methods**

## all\_symbols → array

Returns an array of all the symbols currently in Ruby's symbol table.



## **Public Instance Methods**

Symbol <=> other\_symbol → -1, 0, +1 or nil Compares symbol with other\_symbol after calling to\_s on each of the symbols. Returns -1, 0, +1 or nil depending on whether symbol is less than, equal to, or greater than other\_symbol.

+nil+ is returned if the two values are incompara

See String#<=> for more information.

#### osym == obj → true or false

Equality—If *sym* and *obj* are exactly the same symbol, returns true.

## sym == obj → true or false

Equality—If *sym* and *obj* are exactly the same symbol, returns true.

rightarrow sym =~ obj → fixnum or nil rightarrow match(obj) → fixnum or nil

Returns sym.to\_s =~ obj.

 sym[idx] → char
 sym[b, n] → string
 slice(idx) → char
 slice(b, n) → string Returns sym.to\_s[].

Same as sym.to\_s.capitalize.intern.

## $\odot$ casecmp(other) $\rightarrow$ -1, 0, +1 or nil

Case-insensitive version of symbol#<=>.

#### owncase → symbol

Same as sym.to\_s.downcase.intern.

## empty? $\rightarrow$ true or false

Returns that *sym* is :"" or not.

## $_{\odot}$ encoding $\rightarrow$ encoding

Returns the Encoding object that represents the encoding of *sym*.

## $\odot$ id2name $\rightarrow$ string

## oto\_s → string

Returns the name or string corresponding to sym.

:fred.id2name #=> "fred"

## $\odot$ inspect $\rightarrow$ string

Returns the representation of sym as a symbol literal.

#### osym → sym

#### intern → sym

In general, to\_sym returns the symbol corresponding to an object. As *sym* is already a symbol, self is returned in this case.

#### $\odot$ length $\rightarrow$ integer

#### integer → integer

Same as sym.to\_s.length.

## ⊚ sym =~ obj → fixnum or nil

match(obj) → fixnum or nil

Returns sym.to\_s =~ obj.

#### 🏐 SUCC

Same as sym.to\_s.succ.intern.

## $\odot$ length $\rightarrow$ integer

#### integer → integer

Same as sym.to\_s.length.

## $\odot$ sym[idx] $\rightarrow$ char

osym[b, n] → string

- Slice(idx) → char
- $\odot$  slice(b, n)  $\rightarrow$  string

Returns sym.to\_s[].

#### 🏐 SUCC

Same as sym.to\_s.succ.intern.

#### Swapcase → Symbol

Same as sym.to\_s.swapcase.intern.

## to\_proc

Returns a <u>*Proc*</u> object which respond to the given method by *sym*.



## id2name → string

## os to\_s → string

Returns the name or string corresponding to sym.



## $\odot$ to\_sym $\rightarrow$ sym

## ⊚ intern → sym

In general, to\_sym returns the symbol corresponding to an object. As *sym* is already a symbol, self is returned in this case.

#### oupcase → symbol

Same as sym.to\_s.upcase.intern.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class SyntaxError

Raised when encountering Ruby code with an invalid syntax.

eval("1+1=2")

raises the exception:

SyntaxError: (eval):1: syntax error, unexpecte

In Files

Parent

ScriptError

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

## class SystemCallError

SystemCallError is the base class for all low-level platform-dependent errors.

The errors available on the current platform are subclasses of <u>SystemCallError</u> and are defined in the <u>Errno</u> module.

```
File.open("does/not/exist")
```

raises the exception:

Errno::ENOENT: No such file or directory - do

In Files

error.c

Parent

StandardError

## **Public Class Methods**

System\_call\_error === other → true or false Return true if the receiver is a generic SystemCallError, or if the error numbers self and other are the same.

## new(msg, errno) → ⊚ system\_call\_error\_subclass

If *errno* corresponds to a known system error code, constructs the appropriate Errno class for that error, otherwise constructs a generic SystemCallError object. The error number is subsequently available via the errno method.

## **Public Instance Methods**

errno → fixnum

Return this SystemCallError's error number.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class SystemExit

Raised by exit to initiate the termination of the script.

## **In Files**

error.c

## Parent

Exception

## **Public Class Methods**

```
onew → system_exit
```

```
onew(status) → system_exit
```

```
    mew(status, msg) → system_exit
```

```
    mew(msg) → system_exit
    Create a new systemExit exception with the given
    status and message. Status is true, false, or an
```

integer. If status is not given, true is used.

## **Public Instance Methods**

## status → fixnum

Return the status value associated with this system exit.

## $\odot$ success? $\rightarrow$ true or false

Returns true if exiting successful, false if not.

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class SystemStackError

Raised in case of a stack overflow.

```
def me_myself_and_i
    me_myself_and_i
end
me_myself_and_i
```

raises the exception:

SystemStackError: stack level too deep

## **In Files**

proc.c

Parent

Exception

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class Thread

Threads are the Ruby implementation for a concurrent programming model.

Programs that require multiple threads of execution are a perfect candidate for Ruby's <u>Thread</u> class.

For example, we can create a new thread separate from the main thread's execution using ::new.

```
thr = Thread.new { puts "Whats the big deal"
```

Then we are able to pause the execution of the main thread and allow our new thread to finish, using join:

```
thr.join #=> "Whats the big deal"
```

If we don't call thr.join before the main thread terminates, then all other threads including thr will be killed.

Alternatively, you can use an array for handling multiple threads at once, like in the following example:

```
threads = []
```

threads << Thread.new { puts "Whats the big de threads << Thread.new { 3.times { puts "Thread

After creating a few threads we wait for them all to finish consecutively.

threads.each { |thr| thr.join }

## **Thread** initialization

In order to create new threads, Ruby provides <u>::new</u>, <u>::start</u>, and <u>::fork</u>. A block must be provided with each of these methods, otherwise a <u>ThreadError</u> will be raised.

When subclassing the <u>Thread</u> class, the initialize method of your subclass will be ignored by <u>::start</u> and <u>::fork</u>. Otherwise, be sure to call super in your initialize method.

## Thread termination

For terminating threads, Ruby provides a variety of ways to do this.

The class method <u>::kill</u>, is meant to exit a given thread:

```
thr = Thread.new { ... }
Thread.kill(thr) # sends exit() to thr
```

Alternatively, you can use the instance method

exit, or any of its aliases kill or terminate.

thr.exit

## **Thread** status

Ruby provides a few instance methods for querying the state of a given thread. To get a string with the current thread's state use <u>status</u>



You can also use <u>alive?</u> to tell if the thread is running or sleeping, and <u>stop?</u> if the thread is dead or sleeping.

## **Thread** variables and scope

Since threads are created with blocks, the same rules apply to other Ruby blocks for variable scope. Any local variables created within this block are accessible to only this thread.

## Fiber-local vs. Thread-local

Each fiber has its own bucket for Thread#[] storage. When you set a new fiber-local it is only accessible within this Fiber. To illustrate:

```
Thread.new {
```

```
Thread.current[:foo] = "bar"
Fiber.new {
    p Thread.current[:foo] # => nil
}.resume
}.join
```

This example uses #[] for getting and #[]= for setting fiber-locals, you can also use keys to list the fiber-locals for a given thread and key? to check if a fiber-local exists.

When it comes to thread-locals, they are accessible within the entire scope of the thread. Given the following example:



You can see that the thread-local : foo carried over into the fiber and was changed to 2 by the end of the thread.

This example makes use of <u>thread\_variable\_set</u> to create new thread-locals, and <u>thread\_variable\_get</u> to reference them.

There is also <u>thread\_variables</u> to list all threadlocals, and <u>thread\_variable?</u> to check if a given thread-local exists.

## **Exception** handling

Any thread can raise an exception using the <u>raise</u> instance method, which operates similarly to Kernel#raise.

However, it's important to note that an exception that occurs in any thread except the main thread depends on <u>abort\_on\_exception</u>. This option is false by default, meaning that any unhandled exception will cause the thread to terminate silently when waited on by either join or value. You can change this default by either <u>abort\_on\_exception=</u> true or setting \$::DEBUG to true.

With the addition of the class method ::handle\_interrupt, you can now handle exceptions asynchronously with threads.

## Scheduling

Ruby provides a few ways to support scheduling threads in your program.

The first way is by using the class method <u>::stop</u>, to put the current running thread to sleep and schedule the execution of another thread.

Once a thread is asleep, you can use the instance method wakeup to mark your thread as

eligible for scheduling.

You can also try <u>::pass</u>, which attempts to pass execution to another thread but is dependent on the OS whether a running thread will switch or not. The same goes for <u>priority</u>, which lets you hint to the thread scheduler which threads you want to take precedence when passing execution. This method is also dependent on the OS and may be ignored on some platforms.

## **In Files**

- 📄 thread.c
- 📄 vm.c
- vm\_trace.c

## Parent

Object

## **Public Class Methods**

## $\odot$ DEBUG $\rightarrow$ num

Returns the thread debug level. Available only if compiled with THREAD\_DEBUG=-1.

## DEBUG = num

Sets the thread debug level. Available only if compiled with THREAD\_DEBUG=-1.

## abort\_on\_exception → true or false

Returns the status of the global "abort on exception" condition.

The default is false.

When set to true, all threads will abort (the process will exit(0)) if an exception is raised in any thread.

Can also be specified by the global \$::DEBUG flag or command line option -d.

See also ::abort\_on\_exception=.

There is also an instance level method to set this for a specific thread, see abort\_on\_exception.

## abort\_on\_exception= boolean $\rightarrow$ true or $_{\odot}$ false

When set to true, all threads will abort if an exception is raised. Returns the new state.

```
Thread.abort_on_exception = true
t1 = Thread.new do
   puts "In new thread"
   raise "Exception from thread"
end
sleep(1)
puts "not reached"
```

This will produce:

```
In new thread
prog.rb:4: Exception from thread (RuntimeError)
from prog.rb:2:in `initialize'
from prog.rb:2:in `new'
from prog.rb:2
```

See also ::abort\_on\_exception.

There is also an instance level method to set this for a specific thread, see abort\_on\_exception=.

## ourrent → thread

Returns the currently executing thread.

Thread.current #=> #<Thread:0x401bdf4c run>

## exit → thread

Terminates the currently running thread and schedules another thread to be run.

If this thread is already marked to be killed, <u>::exit</u> returns the <u>Thread</u>.

If this is the main thread, or the last thread, exit the process.

## start([args]\*) {|args| block } → thread fork([args]\*) {|args| block } → thread

Basically the same as <u>::new</u>. However, if class <u>Thread</u> is subclassed, then calling start in that subclass will not invoke the subclass's initialize method.

# handle\_interrupt(hash) { ... } $\rightarrow$ result of the $_{\textcircled{}}$ block

Changes asynchronous interrupt timing.

*interrupt* means asynchronous event and corresponding procedure by <u>#raise</u>, <u>#kill</u>, signal trap (not supported yet) and main thread termination (if main thread terminates, then all other thread will be killed).

The given hash has pairs like ExceptionClass => :TimingSymbol. Where the ExceptionClass is the

interrupt handled by the given block. The TimingSymbol can be one of the following symbols:

```
:immediate
```

Invoke interrupts immediately.

:on\_blocking

Invoke interrupts while *BlockingOperation*.

:never

Never invoke all interrupts.

*BlockingOperation* means that the operation will block the calling thread, such as read and write. On CRuby implementation, *BlockingOperation* is any operation executed without GVL.

Masked asynchronous interrupts are delayed until they are enabled. This method is similar to sigprocmask(3).

## NOTE

Asynchronous interrupts are difficult to use.

If you need to communicate between threads, please consider to use another way such as Queue.

Or use them with deep understanding about this method.

## Usage

In this example, we can guard from  $\frac{\text{#raise}}{\text{exceptions.}}$ 

Using the :never TimingSymbol the <u>RuntimeError</u> exception will always be ignored in the first block of the main thread. In the second <u>::handle\_interrupt</u> block we can purposefully handle <u>RuntimeError</u> exceptions.

While we are ignoring the <u>RuntimeError</u> exception, it's safe to write our resource allocation code. Then, the ensure block is where we can safely deallocate your resources.

## **Guarding from Timeout::Error**

In the next example, we will guard from the Timeout::Error exception. This will help prevent from leaking resources when Timeout::Error exceptions occur during normal ensure clause. For this example we use the help of the standard library Timeout, from lib/timeout.rb

```
require 'timeout'
Thread.handle_interrupt(Timeout::Error => :never)
timeout(10){
    # Timeout::Error doesn't occur here
    Thread.handle_interrupt(Timeout::Error => :or
        # possible to be killed by Timeout::Error
        # while blocking operation
    }
    # Timeout::Error doesn't occur here
}
```

In the first part of the timeout block, we can rely on Timeout::Error being ignored. Then in the Timeout::Error => :on\_blocking block, any operation that will block the calling thread is susceptible to a Timeout::Error exception being raised.

#### **Stack control settings**

It's possible to stack multiple levels of <u>::handle\_interrupt</u> blocks in order to control more than one ExceptionClass and TimingSymbol at a time.



#### Inheritance with ExceptionClass

All exceptions inherited from the ExceptionClass parameter will be considered.



## $\otimes$ kill(thread) $\rightarrow$ thread

Causes the given thread to exit, see also ::exit.

count = 0	
a = Thread.new {	loop { count += 1 } }
<pre>sleep(0.1)</pre>	
Thread.kill(a)	
count	
a.alive?	

ist → array

Returns an array of <u>Thread</u> objects for all threads that are either runnable or stopped.

```
Thread.new { sleep(200) }
Thread.new { 1000000.times {|i| i*i } }
Thread.new { Thread.stop }
Thread.list.each {|t| p t}
```

This will produce:

```
#<Thread:0x401b3e84 sleep>
#<Thread:0x401b3f38 run>
#<Thread:0x401b3fb0 sleep>
#<Thread:0x401bdf4c run>
```

 $\odot$  main  $\rightarrow$  thread

Returns the main thread.

```
_{\odot} new { ... } → thread
```

- $_{\odot}$  new(\*args, &proc)  $\rightarrow$  thread
- $_{\odot}$  new(\*args) { |args| ... }  $\rightarrow$  thread

Creates a new thread executing the given block.

Any args given to ::new will be passed to the block:



A <u>ThreadError</u> exception is raised if <u>::new</u> is called without a block.

If you're going to subclass <u>Thread</u>, be sure to call super in your initialize method, otherwise a <u>ThreadError</u> will be raised.

Give the thread scheduler a hint to pass execution to another thread. A running thread may or may not switch, it depends on OS and processor.

## $\odot$ pending\_interrupt?(error = nil) $\rightarrow$ true/false

Returns whether or not the asynchronous queue is empty.

Since ::handle\_interrupt can be used to defer asynchronous events, this method can be used to determine if there are any deferred events.

If you find this method returns true, then you may finish :never blocks.

For example, the following method processes deferred asynchronous events immediately.



If error is given, then check only for error type deferred events.

## Usage





This example can also be written as the following, which you should use to avoid asynchronous interrupts.



## start([args]\*) {|args| block } → thread fork([args]\*) {|args| block } → thread

Basically the same as <u>::new</u>. However, if class <u>Thread</u> is subclassed, then calling start in that subclass will not invoke the subclass's initialize method.

## $\odot$ stop $\rightarrow$ nil

Stops execution of the current thread, putting it into a "sleep" state, and schedules execution of another thread.

```
a = Thread.new { print "a"; Thread.stop; print "c
sleep 0.1 while a.status!='sleep'
print "b"
a.run
a.join
```



## **Public Instance Methods**

## thr[sym] → obj or nil

Attribute Reference—Returns the value of a fiberlocal variable (current thread's root fiber if not explicitly inside a <u>Fiber</u>), using either a symbol or a string name. If the specified variable does not exist, returns nil.

[	
<pre>Thread.new { Thread.current["name</pre>	"] = "A" },
<pre>Thread.new { Thread.current[:name</pre>	] = "B" },
<pre>Thread.new { Thread.current["name</pre>	"] = "C" }
].each do  th	
th.join	
<pre>puts "#{th.inspect}: #{th[:name]}</pre>	u –
end	

This will produce:

```
#<Thread:0x0000002a54220 dead>: A
#<Thread:0x00000002a541a8 dead>: B
#<Thread:0x00000002a54130 dead>: C
```

Thread#[] and Thread#[]= are not thread-local but fiber-local. This confusion did not exist in Ruby 1.8 because fibers are only available since Ruby 1.9. Ruby 1.9 chooses that the methods behaves fiberlocal to save following idiom for dynamic scope.

```
def meth(newvalue)
    begin
    oldvalue = Thread.current[:name]
    Thread.current[:name] = newvalue
    yield
    ensure
    Thread.current[:name] = oldvalue
```

#### end end

The idiom may not work as dynamic scope if the methods are thread-local and a given block switches fiber.



For thread-local variables, please see <u>thread\_variable\_get</u> and <u>thread\_variable\_set</u>.

## obj → obj → obj

Attribute Assignment—Sets or creates the value of a fiber-local variable, using either a symbol or a string.

See also Thread#[].

For thread-local variables, please see thread\_variable\_set and thread\_variable\_get.

#### $\otimes$ abort\_on\_exception $\rightarrow$ true or false

Returns the status of the thread-local "abort on exception" condition for this thr.

The default is false.

See also abort\_on\_exception=.

There is also a class level method to set this for all

threads, see ::abort\_on\_exception.

# abort\_on\_exception= boolean $\rightarrow$ true or $_{\textcircled{}}$ false

When set to true, all threads (including the main program) will abort if an exception is raised in this thr.

The process will effectively exit(0).

See also abort\_on\_exception.

There is also a class level method to set this for all threads, see ::abort\_on\_exception=.

## $_{\odot}$ add\_trace\_func(proc) $\rightarrow$ proc

Adds proc as a handler for tracing.

See #set\_trace\_func and Kernel#set\_trace\_func.

## $_{\odot}$ alive? $\rightarrow$ true or false

Returns true if thr is running or sleeping.



See also stop? and status.

#### backtrace → array

Returns the current backtrace of the target thread.

## $_{\odot}$ backtrace\_locations(\*args) $\rightarrow$ array or nil

Returns the execution stack for the target thread—an
array containing backtrace location objects.

See Thread::Backtrace::Location for more information.

This method behaves similarly to

Kernel#caller\_locations except it applies to a specific thread.

## ⊚ exit → thr or nil

## ill → thr or nil

## terminate → thr or nil

Terminates thr and schedules another thread to be run.

If this thread is already marked to be killed, <u>exit</u> returns the <u>Thread</u>.

If this is the main thread, or the last thread, exits the process.

# group → thgrp or nil

Returns the <u>ThreadGroup</u> which contains the given thread, or returns nil if thr is not a member of any group.



# inspect → string

Dump the name, id, and status of *thr* to a string.

# ⊚join → thr

### $\odot$ join(limit) $\rightarrow$ thr

The calling thread will suspend execution and run this

thr.

Does not return until thr exits or until the given limit seconds have passed.

If the time limit expires, nil will be returned, otherwise thr is returned.

Any threads not joined will be killed when the main program exits.

If thr had previously raised an exception and the ::abort\_on\_exception or \$::DEBUG flags are not set, (so the exception has not yet been processed), it will be processed at this time.



The following example illustrates the limit parameter.



This will produce:

tick Waiting tick			
Waiting			
tick			

# $_{\odot}$ key?(sym) $\rightarrow$ true or false

Returns true if the given string (or symbol) exists as a fiber-local variable.



## 

Returns an array of the names of the fiber-local variables (as Symbols).

```
thr = Thread.new do
   Thread.current[:cat] = 'meow'
   Thread.current["dog"] = 'woof'
end
thr.join  #=> #<Thread:0x401b3f10 dead>
thr.keys  #=> [:dog, :cat]
```

# rightarrow exit $\rightarrow$ thr or nil

#### ill → thr or nil

### terminate → thr or nil

Terminates thr and schedules another thread to be run.

If this thread is already marked to be killed, <u>exit</u> returns the <u>Thread</u>.

If this is the main thread, or the last thread, exits the process.

# $\odot$ pending\_interrupt?(error = nil) $\rightarrow$ true/false

Returns whether or not the asynchronous queue is empty for the target thread.

If error is given, then check only for error type deferred events.

See ::pending\_interrupt? for more information.

## ority → integer i

Returns the priority of *thr*. Default is inherited from the current thread which creating the new thread, or zero for the initial main thread; higher-priority thread will run more frequently than lower-priority threads (but lower-priority threads can also run).

This is just hint for Ruby thread scheduler. It may be ignored on some platform.

```
Thread.current.priority #=> 0
```

### $_{\odot}$ priority= integer $\rightarrow$ thr

Sets the priority of *thr* to *integer*. Higher-priority threads will run more frequently than lower-priority threads (but lower-priority threads can also run).

This is just hint for Ruby thread scheduler. It may be ignored on some platform.

🚳 raise

- raise(string)
- raise(exception [, string [, array]])

Raises an exception from the given thread. The caller

does not have to be thr. See <u>Kernel#raise</u> for more information.

```
Thread.abort_on_exception = true
a = Thread.new { sleep(200) }
a.raise("Gotcha")
```

This will produce:

```
prog.rb:3: Gotcha (RuntimeError)
from prog.rb:2:in `initialize'
from prog.rb:2:in `new'
from prog.rb:2
```

#### $\otimes$ run $\rightarrow$ thr

Wakes up thr, making it eligible for scheduling.

```
a = Thread.new { puts "a"; Thread.stop; puts "c"
sleep 0.1 while a.status!='sleep'
puts "Got here"
a.run
a.join
```

This will produce:



See also the instance method wakeup.

### $_{\odot}$ safe\_level $\rightarrow$ integer

Returns the safe level in effect for *thr*. Setting threadlocal safe levels can help when implementing sandboxes which run insecure code.

```
thr = Thread.new { $SAFE = 3; sleep }
Thread.current.safe_level #=> 0
thr.safe_level #=> 3
```

## set\_trace\_func(proc) → proc set\_trace\_func(nil) → nil

Establishes *proc* on *thr* as the handler for tracing, or disables tracing if the parameter is nil.

See Kernel#set\_trace\_func.

## $_{\odot}$ status $\rightarrow$ string, false or nil

Returns the status of thr.

"sleep"

Returned if this thread is sleeping or waiting on  $\ensuremath{\mathsf{I/O}}$ 

"run"

When this thread is executing

"aborting"

If this thread is aborting

false

When this thread is terminated normally

nil

If terminated with an exception.

See also the instance methods alive? and stop?

## $rightarrow stop? \rightarrow true or false$

Returns true if thr is dead or sleeping.

```
a = Thread.new { Thread.stop }
b = Thread.current
a.stop? #=> true
b.stop? #=> false
```

See also alive? and status.

## ⊚ exit → thr or nil

#### is kill → thr or nil

#### terminate → thr or nil

Terminates thr and schedules another thread to be run.

If this thread is already marked to be killed, <u>exit</u> returns the Thread.

If this is the main thread, or the last thread, exits the process.

# $_{\odot}$ thread\_variable?(key) → true or false

Returns true if the given string (or symbol) exists as a thread-local variable.



Note that these are not fiber local variables. Please see Thread#[] and <u>#thread\_variable\_get</u> for more details.

# $_{\odot}$ thread\_variable\_get(key) $\rightarrow$ obj or nil

Returns the value of a thread local variable that has been set. Note that these are different than fiber local values. For fiber local values, please see Thread#[] and Thread#[]=.

Thread local values are carried along with threads, and do not respect fibers. For example:



The value "bar" is returned for the thread local, where nil is returned for the fiber local. The fiber is executed in the same thread, so the thread local values are available.

# thread\_variable\_set(key, value)

Sets a thread local with key to value. Note that these are local to threads, and not to fibers. Please see <u>#thread\_variable\_get</u> and Thread#[] for more information.

## $_{\odot}$ thread\_variables $\rightarrow$ array

Returns an array of the names of the thread-local variables (as Symbols).

```
thr = Thread.new do
    Thread.current.thread_variable_set(:cat, 'meow
```



Note that these are not fiber local variables. Please see Thread#[] and <u>#thread\_variable\_get</u> for more details.

# $\odot$ value $\rightarrow$ obj

Waits for thr to complete, using join, and returns its value or raises the exception which terminated the thread.



## $\otimes$ wakeup $\rightarrow$ thr

Marks a given thread as eligible for scheduling, however it may still remain blocked on I/O.

**Note:** This does not invoke the scheduler, see <u>run</u> for more information.



Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class ThreadError

Raised when an invalid operation is attempted on a thread.

For example, when no other thread has been started:

Thread.stop

This will raises the following exception:

```
ThreadError: stopping only thread note: use sleep to stop forever
```

# In Files

📄 thread.c

# Parent

StandardError

Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class ThreadGroup

ThreadGroup provides a means of keeping track of a number of threads as a group.

A given <u>Thread</u> object can only belong to one <u>ThreadGroup</u> at a time; adding a thread to a new group will remove it from any previous group.

Newly created threads belong to the same group as the thread from which they were created.

# **In Files**

lthread.c

# Parent

Object

# Constants

#### Default

The default <u>ThreadGroup</u> created when Ruby starts; all Threads belong to it by default.

# **Public Instance Methods**

# $_{\odot}$ add(thread) $\rightarrow$ thgrp

Adds the given thread to this group, removing it from any other group to which it may have previously been a member.



This will produce:



### enclose → thgrp

Prevents threads from being added to or removed from the receiving ThreadGroup.

New threads can still be started in an enclosed <u>ThreadGroup</u>.



 $_{\odot}$  enclosed?  $\rightarrow$  true or false

Returns true if the thgrp is enclosed. See also #enclose.

# osist → array

Returns an array of all existing <u>Thread</u> objects that belong to this group.



Generated by <u>RDoc</u> 3.12.2. Generated with the <u>Darkfish Rdoc Generator</u> 3.

# class Time

Time is an abstraction of dates and times. Time is stored internally as the number of seconds with fraction since the *Epoch*, January 1, 1970 00:00 UTC. Also see the library module Date. The <u>Time</u> class treats GMT (Greenwich Mean <u>Time</u>) and UTC (Coordinated Universal <u>Time</u>) as equivalent. GMT is the older way of referring to these baseline times but persists in the names of calls on POSIX systems.

All times may have fraction. Be aware of this fact when comparing times with each other – times that are apparently equal when displayed may be different when compared.

Since Ruby 1.9.2, <u>Time</u> implementation uses a signed 63 bit integer, <u>Bignum</u> or <u>Rational</u>. The integer is a number of nanoseconds since the *Epoch* which can represent 1823-11-12 to 2116-02-20. When <u>Bignum</u> or <u>Rational</u> is used (before 1823, after 2116, under nanosecond), <u>Time</u> works slower as when integer is used.

# **Examples**

All of these examples were done using the EST timezone which is GMT-5.

# **Creating a new <b>Time** instance

You can create a new instance of <u>Time</u> with <u>::new</u>. This will use the current system time. <u>::now</u> is an alias for this. You can also pass parts of the time to <u>::new</u> such as year, month, minute, etc. When you want to construct a time this way you must pass at least a year. If you pass the year with nothing else time will default to January 1 of that year at 00:00:00 with the current system timezone. Here are some examples:

Time.new(2002)								
Time.new(2002,	10)							
Time.new(2002,	10,	31)						
Time.new(2002,	10,	31,	2,	2,	2,	"+02:0	90")	
4								Þ

You can also use gm, local and <u>utc</u> to infer GMT, local and UTC timezones instead of using the current system setting.

You can also create a new time using <u>::at</u> which takes the number of seconds (or fraction of seconds) since the <u>Unix Epoch</u>.



# Working with an instance of **Time**

Once you have an instance of <u>Time</u> there is a multitude of things you can do with it. Below are some examples. For all of the following examples, we will work on the assumption that you have done the following:



Was that a monday?

t.monday? #=> false

What year was that again?

t.year **#=> 1993** 

Was is daylight savings at the time?

t.dst? #=> false

What's the day a year later?



How many seconds was that since the Unix Epoch?

#### t.to\_i #=> 730522800

You can also do standard functions like compare two times.



# In Files

📄 time.c

# Parent

Object

# **Included Modules**

Comparable

# **Public Class Methods**

 $_{\odot}$  at(time)  $\rightarrow$  time

- $_{\odot}$  at(seconds\_with\_frac)  $\rightarrow$  time
  - at(seconds, microseconds\_with\_frac)  $\rightarrow$

# 🚳 time

Creates a new <u>Time</u> object with the value given by time, the given number of seconds\_with\_frac, Or seconds and microseconds\_with\_frac Since the Epoch. seconds\_with\_frac and microseconds\_with\_frac can be an <u>Integer</u>, <u>Float</u>, Rational, or other Numeric. non-portable feature

allows the offset to be negative on some systems.

If a numeric argument is given, the result is in local time.

Time.at(0)	1969-12
Time.at(Time.at(0))	<b>1969-12</b>
Time.at(946702800)	1999-12
Time.at(-284061600)	<b>1960-12</b>
Time.at(946684800.2).usec	200000
Time.at(946684800, 123456.789).nsec	12345678
4	•

 $_{\odot}$  utc(year)  $\rightarrow$  time

- $_{\odot}$  utc(year, month)  $\rightarrow$  time
- $_{\odot}$  utc(year, month, day)  $\rightarrow$  time
- $_{\odot}$  utc(year, month, day, hour)  $\rightarrow$  time
- rightarrow utc(year, month, day, hour, min) → time utc(year, month, day, hour, min,
- sec\_with\_frac) → time utc(year, month, day, hour, min, sec,
- $\otimes$  usec\_with\_frac)  $\rightarrow$  time
  - utc(sec, min, hour, day, month, year, dummy,
- og dummy, dummy, dummy) → time
- gm(year) → time
- $_{\odot}$  gm(year, month)  $\rightarrow$  time
- $_{\odot}$  gm(year, month, day)  $\rightarrow$  time
- $_{\odot}$  gm(year, month, day, hour)  $\rightarrow$  time

 $_{\odot}$  gm(year, month, day, hour, min)  $\rightarrow$  time gm(year, month, day, hour, min,  $rightarrow sec_with_frac) \rightarrow time$ gm(year, month, day, hour, min, sec,  $rightarrow usec with frac) \rightarrow time$ gm(sec, min, hour, day, month, year, dummy, rightarrow dummy, dummy)  $\rightarrow$  time Creates a Time object based on given values, interpreted as UTC (GMT). The year must be specified. Other values default to the minimum value for that field (and may be nil or omitted). Months may be specified by numbers from 1 to 12, or by the three-letter English month names. Hours are specified on a 24-hour clock (0..23). Raises an ArgumentError if any values are out of range. Will also accept ten arguments in the order output by #to a.

sec\_with\_frac and usec\_with\_frac can have a
fractional part.



 $\odot$  local(year)  $\rightarrow$  time

- $\odot$  local(year, month)  $\rightarrow$  time
- $_{\odot}$  local(year, month, day)  $\rightarrow$  time
- $_{\odot}$  local(year, month, day, hour)  $\rightarrow$  time
- rightarrow local(year, month, day, hour, min) → time local(year, month, day, hour, min,
- sec\_with\_frac) → time local(year, month, day, hour, min, sec,
- $\otimes$  usec\_with\_frac)  $\rightarrow$  time

local(sec, min, hour, day, month, year,

- ombody dummy, dummy, isdst, dummy) → time
- $_{\odot}$  mktime(year)  $\rightarrow$  time
- $\otimes$  mktime(year, month)  $\rightarrow$  time
- $\otimes$  mktime(year, month, day)  $\rightarrow$  time
- $\otimes$  mktime(year, month, day, hour)  $\rightarrow$  time
- mktime(year, month, day, hour, min) → time mktime(year, month, day, hour, min,
- sec\_with\_frac) → time mktime(year, month, day, hour, min, sec,

```
rightarrow usec with frac) \rightarrow time
```

mktime(sec, min, hour, day, month, year,  $rac{1}{2}$  dummy, dummy, isdst, dummy)  $\rightarrow$  time

Same as <u>::gm</u>, but interprets the values in the local time zone.

Time.local(2000,"jan",1,20,15,1) #=> 2000-

```
\odot local(year) \rightarrow time
```

4

- rightarrow local(year, month)  $\rightarrow$  time
- $\odot$  local(year, month, day)  $\rightarrow$  time
- $\odot$  local(year, month, day, hour)  $\rightarrow$  time
- Solution Strain Str
- $\odot$  sec\_with\_frac)  $\rightarrow$  time

local(year, month, day, hour, min, sec,

 $\otimes$  usec\_with\_frac)  $\rightarrow$  time

local(sec, min, hour, day, month, year,

- omega dummy, dummy, isdst, dummy) → time
- $\otimes$  mktime(year)  $\rightarrow$  time
- $_{\odot}$  mktime(year, month)  $\rightarrow$  time

- $_{\odot}$  mktime(year, month, day)  $\rightarrow$  time
- $_{\odot}$  mktime(year, month, day, hour)  $\rightarrow$  time
- Sometime (year, month, day, hour, min) → time mktime (year, month, day, hour, min,
- sec\_with\_frac) → time
   mktime(year, month, day, hour, min, sec,
   usec with frac) → time

mktime(sec, min, hour, day, month, year,  $rac{1}{2}$  dummy, dummy, isdst, dummy)  $\rightarrow$  time

Same as <u>::gm</u>, but interprets the values in the local time zone.



#### onew → time

# new(year, month=nil, day=nil, hour=nil, min=nil, sec=nil, utc\_offset=nil) → time

Returns a <u>Time</u> object.

It is initialized to the current system time if no argument is given.

**Note:** The new object will use the resolution available on your system clock, and may include fractional seconds.

If one or more arguments specified, the time is initialized to the specified time.

sec may have fraction if it is a rational.

utc\_offset is the offset from UTC. It can be a string such as "+09:00" or a number of seconds such as 32400.



a == b #=>			
"%.6f" % a.to_f			
"%.6f" % b.to_f			
Time.new(2008,6,21, 1	3,30,0, "+0	9:00") #=>	
, , , , , ,		,	
# A trip for RubyConf			
t1 = Time.new(2007,11	,1,15,25,0,	"+09:00")	
t2 = Time.new(2007, 11)	,1,12, 5,0,	"-05:00")	
t3 = Time.new(2007, 11)	,1,13,25,0,	"-05:00")	
t4 = Time.new(2007,11	,1,16,53,0,	"-04:00")	
t5 = Time.new(2007,11	,5, 9,24,0,	"-05:00")	
t6 = Time.new(2007,11	,5,11,21,0,	"-05:00")	
t7 = Time.new(2007,11	,5,13,45,0,	"-05:00")	
t8 = Time.new(2007,11	,6,17,10,0,	"+09:00")	
p((t2-t1)/3600.0)			
p((t4-t3)/3600.0)			
p((t6-t5)/3600.0)			
p((t8-t7)/3600.0)			
			•

### $\otimes$ now $\rightarrow$ time

Creates a new <u>Time</u> object for the current time. This is same as ::new without arguments.



- $_{\odot}$  utc(year)  $\rightarrow$  time
- $\otimes$  utc(year, month)  $\rightarrow$  time
- $_{\odot}$  utc(year, month, day)  $\rightarrow$  time
- $_{\odot}$  utc(year, month, day, hour)  $\rightarrow$  time
- outc(year, month, day, hour, min) → time utc(year, month, day, hour, min,
- rightarrowsec\_with\_frac) → time utc(year, month, day, hour, min, sec,
- $\otimes$  usec\_with\_frac)  $\rightarrow$  time
  - utc(sec, min, hour, day, month, year, dummy,

- omega dummy, dummy, dummy) → time
- ⊕ gm(year) → time
- $_{\odot}$  gm(year, month)  $\rightarrow$  time
- $_{\odot}$  gm(year, month, day)  $\rightarrow$  time
- $_{\odot}$  gm(year, month, day, hour)  $\rightarrow$  time
- gm(year, month, day, hour, min) → time gm(year, month, day, hour, min,
- sec\_with\_frac) → time gm(year, month, day, hour, min, sec,
- with\_frac) → time
   gm(sec, min, hour, day, month, year, dummy,
   a dummy, dummy, dummy) → time

Creates a Time object based on given values, interpreted as UTC (GMT). The year must be specified. Other values default to the minimum value for that field (and may be nil or omitted). Months may be specified by numbers from 1 to 12, or by the three-letter English month names. Hours are specified on a 24-hour clock (0..23). Raises an <u>ArgumentError</u> if any values are out of range. Will also accept ten arguments in the order output by #to\_a.

sec\_with\_frac and usec\_with\_frac can have a
fractional part.



# **Public Instance Methods**

### on time + numeric → time

Addition — Adds some number of seconds (possibly

fractional) to *time* and returns that value as a new Time object.



# $rightarrow time - other_time → float$ <math>rightarrow time - numeric → time

Difference — Returns a new <u>Time</u> object that represents the difference between *time* and other\_time, or subtracts the given number of seconds in numeric from *time*.



# time <=> other\_time → -1, 0, +1 or nil

Comparison—Compares time with other\_time.

-1, 0, +1 or nil depending on whether time is less than, equal to, or greater than other\_time.

nil is returned if the two values are incomparable.

```
t = Time.now  #=> 2007-11-19 08:12:12 -0600
t2 = t + 2592000  #=> 2007-12-19 08:12:12 -0600
t <=> t2  #=> -1
t2 <=> t  #=> 1
t = Time.now  #=> 2007-11-19 08:13:38 -0600
t2 = t + 0.1  #=> 2007-11-19 08:13:38 -0600
t.nsec  #=> 98222990
t2.nsec  #=> 98222990
t2.nsec  #=> 198222999
t <=> t2  #=> -1
t2 <=> t  #=> 1
t <=> t  #=> 1
```

# [▲] F $\odot$ asctime $\rightarrow$ string $\odot$ ctime $\rightarrow$ string Returns a canonical string representation of time. Time.now.asctime 4 $\otimes$ asctime $\rightarrow$ string $\odot$ ctime $\rightarrow$ string Returns a canonical string representation of time. Time.now.asctime 4 a day → fixnum omday → fixnum Returns the day of the month (1..n) for time. t = Time.now t.day t.mday

# $_{\odot}$ isdst $\rightarrow$ true or false

### obst? → true or false

Returns true if *time* occurs during Daylight Saving Time in its time zone.

Time.local(2000,	1,	1).zone	
Time.local(2000,	1,	1).isdst	
Time.local(2000,	1,	<b>1).</b> dst?	
Time.local(2000,	7,	1).zone	
Time.local(2000,	7,	1).isdst	
Time.local(2000,	7,	1).dst?	

<pre>Time.local(2000,</pre>	1,	1).zone	
<pre>Time.local(2000,</pre>	1,	1).isdst	
<pre>Time.local(2000,</pre>	1,	<b>1).dst</b> ?	
<pre>Time.local(2000,</pre>	7,	1).zone	
<pre>Time.local(2000,</pre>	7,	1).isdst	
Time.local(2000,	7,	1).dst?	

# @ eql?(other\_time)

Returns true if *time* and other\_time are both <u>Time</u> objects with the same seconds and fractional seconds.

# $_{\odot}$ friday? → true or false

Returns true if *time* represents Friday.

t = Time.local(1987, 12, 1 t.friday?	18)	
		•

# ⊕ getgm → new\_time

### getutc → new\_time

Returns a new Time object representing time in UTC.



## getlocal → new\_time

## $\odot$ getlocal(utc\_offset) $\rightarrow$ new\_time

Returns a new <u>Time</u> object representing *time* in local time (using the local time zone in effect for this

process).

If utc\_offset is given, it is used instead of the local time.

<pre>l = t.getlocal  #=&gt; 2000-01-01 1 l.utc?  #=&gt; false t == 1  #=&gt; true j = t.getlocal("+09:00")  #=&gt; 2000-01-02 0 j.utc?  #=&gt; false t == j  #=&gt; true</pre>	t = Time.utc(2000,1,1,20,15,1) t.utc?			
<pre>j = t.getlocal("+09:00") #=&gt; 2000-01-02 0 j.utc? #=&gt; false t == j #=&gt; true</pre>	l = t.getlocal l.utc? t == l			
	j = t.getlocal("+09:00") j.utc? t == j	#=> #=> #=>	2000-01-02 false true	0

# ⊚ getgm → new\_time ⊚ getutc → new\_time

Returns a new Time object representing time in UTC.



# $\odot$ utc? $\rightarrow$ true or false

# $\odot$ gmt? $\rightarrow$ true or false

Returns true if time represents a time in UTC (GMT).

<pre>t = Time.now t.utc? t = Time.gm(2000,"jan",1,20,15,1) t.utc?</pre>	
<pre>t = Time.now t.gmt? t = Time.gm(2000,1,1,20,15,1) t.gmt?</pre>	
4	•

offset → fixnum

# gmtoff → fixnum

4

# offset → fixnum

Returns the offset in seconds between the timezone of *time* and UTC.



# gmtime → time

## $\otimes$ utc $\rightarrow$ time

Converts time to UTC (GMT), modifying the receiver.

<pre>t = Time.now t.gmt? t.gmtime t.gmt?</pre>		
<pre>t = Time.now t.utc? t.utc t.utc?</pre>		

# gmt\_offset → fixnum

### ⊚gmtoff → fixnum

### offset → fixnum

Returns the offset in seconds between the timezone of *time* and UTC.



## 

4

Returns a hash code for this Time object.

See also Object#hash.

### 

Returns the hour of the day (0..23) for time.

t = Time.now #=> 2007-11-19 08:26:20 -0600 t.hour #=> 8

۲

# inspect → string

### $\odot$ to\_s $\rightarrow$ string

Returns a string representing *time*. Equivalent to calling strftime with the appropriate format string.



# isdst → true or false dst? → true or false

Returns true if *time* occurs during Daylight Saving Time in its time zone.

# CST6CDT:			
lime.local(2000,	1,	1).zone	
Time.local(2000,	1,	1).isdst	
Time.local(2000,	1,	1).dst?	
Time.local(2000,	7,	1).zone	
Time.local(2000,	7,	1).isdst	

Time.local(2000,	7,	1).dst?	#=>	true
Asia/Tokyo:	1	1) zone		
Time.local(2000,	1,	1).isdst		
Time.local(2000, Time.local(2000,	1, 7,	1).dst? 1).zone		
<pre>Time.local(2000, Time.local(2000,</pre>	7, 7,	1).isdst 1).dst?		

## $_{\odot}$ localtime → time $_{\odot}$ localtime(utc\_offset) → time

Converts *time* to local time (using the local time zone in effect for this process) modifying the receiver.

If utc\_offset is given, it is used instead of the local time.

t = Time.utc(2000, t.utc?	"jan",	1,	20,	15,	1)	
t.localtime t.utc?						
t.localtime("+09:00 t.utc?	9")					

# o day → fixnum

# $_{\odot}$ mday $\rightarrow$ fixnum

Returns the day of the month (1..n) for time.



## imin → fixnum

Returns the minute of the hour (0..59) for time.



# mon → fixnum month → fixnum

Returns the month of the year (1..12) for time.

t = Time.now		
t.mon		
t.month		

# $_{\odot}$ monday? $\rightarrow$ true or false

Returns true if time represents Monday.



# mon → fixnum

# month → fixnum

Returns the month of the year (1..12) for time.



# $\otimes$ nsec $\rightarrow$ int

### ostv\_nsec → int

Returns the number of nanoseconds for time.



The lowest digits of  $\underline{to_f}$  and  $\underline{nsec}$  are different

because IEEE 754 double is not accurate enough to represent the exact number of nanoseconds since the Epoch.

The more accurate value is returned by nsec.

# opround([ndigits]) → new\_time

Rounds sub seconds to a given precision in decimal digits (0 digits by default). It returns a new <u>Time</u> object. ndigits should be zero or positive integer.

```
require 'time'
t = Time.utc(2010,3,30, 5,43,"25.123456789".to_r
p t.iso8601(10)
p t.round.iso8601(10)
p t.round(0).iso8601(10)
p t.round(1).iso8601(10)
p t.round(2).iso8601(10)
p t.round(3).iso8601(10)
p t.round(4).iso8601(10)
p t.round(5).iso8601(10)
p t.round(6).iso8601(10)
p t.round(7).iso8601(10)
p t.round(8).iso8601(10)
p t.round(9).iso8601(10)
p t.round(10).iso8601(10)
t = Time.utc(1999,12,31, 23,59,59)
p((t + 0.4).round.iso8601(3))
p((t + 0.49).round.iso8601(3))
p((t + 0.5).round.iso8601(3))
p((t + 1.4).round.iso8601(3))
p((t + 1.49).round.iso8601(3))
p((t + 1.5).round.iso8601(3))
t = Time.utc(1999,12,31, 23,59,59)
p(t + 0.123456789).round(4).iso8601(6)
```

 $_{\odot}$  saturday?  $\rightarrow$  true or false

Returns true if time represents Saturday.



### Sec → fixnum

Returns the second of the minute (0..60) for time.

**Note:** Seconds range from zero to 60 to allow the system to inject leap seconds. See en.wikipedia.org/wiki/Leap\_second for further details.



# $_{\odot}$ strftime( string ) $\rightarrow$ string

Formats *time* according to the directives in the given format string.

The directives begin with a percent (%) character. Any text not listed as a directive will be passed through to the output string.

The directive consists of a percent (%) character, zero or more flags, optional minimum field width, optional modifier and a conversion specifier as follows:

#### %<flags><width><modifier><conversion>

Flags:

-	don't pad a numerical output
_	use spaces for padding
0	use zeros for padding
Λ	upcase the result string
#	change case
:	use colons for %z

The minimum field width specifies the minimum width.

The modifiers are "E" and "O". They are ignored.

Format directives:

```
Date (Year, Month, Day):
 %Y - Year with century if provided, will pad re
          -0001, 0000, 1995, 2009, 14292, etc.
 %C - year / 100 (rounded down such as 20 in 200
 %y - year % 100 (00..99)
 %m - Month of the year, zero-padded (01..12)
          %_m blank-padded ( 1..12)
          %-m no-padded (1..12)
 %B - The full month name (``January'')
          %^B uppercased (``JANUARY'')
 %b - The abbreviated month name (``Jan'')
          %^b uppercased (``JAN'')
 %h - Equivalent to %b
 %d - Day of the month, zero-padded (01..31)
          %-d no-padded (1..31)
 %e - Day of the month, blank-padded (1..31)
 %j - Day of the year (001..366)
Time (Hour, Minute, Second, Subsecond):
 %H - Hour of the day, 24-hour clock, zero-padde
 %k - Hour of the day, 24-hour clock, blank-pade
 %I - Hour of the day, 12-hour clock, zero-padde
 <u>%l - Hour of the day, 12-hour clock, blank-pade</u>
 %P - Meridian indicator, lowercase (``am'' or
 %p - Meridian indicator, uppercase (``AM'' or
 %M - Minute of the hour (00..59)
 %S - Second of the minute (00..60)
 %L - Millisecond of the second (000..999)
       The digits under millisecond are truncated
 %N - Fractional seconds digits, default is 9 di
               millisecond (3 digits)
          %3N
          %6N
               microsecond (6 digits)
               nanosecond (9 digits)
          %9N
          %12N picosecond (12 digits)
```

%15N femtosecond (15 digits) %18N attosecond (18 digits) %21N zeptosecond (21 digits) %24N yoctosecond (24 digits) The digits under the specified length are carry up. Time zone: %z - Time zone as hour and minute offset from I %:z - hour and minute offset from UTC %::z - hour, minute and second offset %Z - Abbreviated time zone name or similar info Weekday: %A - The full weekday name (``Sunday'') %^A uppercased (``SUNDAY'') %a - The abbreviated name (``Sun'') %^a uppercased (``SUN'') %u - Day of the week (Monday is 1, 1..7) %w - Day of the week (Sunday is 0, 0..6) ISO 8601 week-based year and week number: The first week of YYYY starts with a Monday and The days in the year before the first week are in the previous year. %G - The week-based year %g - The last 2 digits of the week-based year %V - Week number of the week-based year (01..53 Week number: The first week of YYYY that starts with a Sunday or %W). The days in the year before the first wee %U - Week number of the year. The week starts w %W - Week number of the year. The week starts w Seconds since the Epoch: %s - Number of seconds since 1970-01-01 00:00:0 Literal string: %n - Newline character (\n) <u>%t - Tab character (\t)</u> %% - Literal ``%'' character Combination: %c - date and time (%a %b %e %T %Y) %D - Date (%m/%d/%y) %F - The ISO 8601 date format (%Y-%m-%d)


This method is similar to strftime() function defined in ISO C and POSIX.

While all directives are locale independent since Ruby 1.9, %Z is platform dependent. So, the result may differ even if the same format string is used in other systems such as C.

%z is recommended over %Z. %Z doesn't identify the timezone. For example, "CST" is used at America/Chicago (-06:00), America/Havana (-05:00), Asia/Harbin (+08:00), Australia/Darwin (+09:30) and Australia/Adelaide (+10:30). Also, %Z is highly dependent on the operating system. For example, it may generate a non ASCII string on Japanese Windows. i.e. the result can be different to "JST". So the numeric time zone offset, %z, is recommended.

**Examples:** 

t t	<pre>: = Time.new(2007,11,19,8,37,48 strftime("Printed on %m/%d/%)</pre>	3,"-06:00 (")	)") #=> 200 "Printed
t		// #=>	
٩Ť			•

Various ISO 8601 formats:

%Y%m%d	=>	20071119	Cal
%F	=>	2007-11-19	Cal
%Y-%m	=>	2007-11	Cal
%Y	=>	2007	Cal
%C	=>	20	Cal
%Y%j	=>	2007323	Ord
%Y-%j	=>	2007-323	Ord
%GW%V%u	=>	2007W471	Wee
%G-W%V-%u	=>	2007-W47-1	Wee

%GW%V	=>	2007W47	Wee
%G-W%V	=>	2007-W47	Wee
%H%M%S	=>	083748	Loc
%Т	=>	08:37:48	Loc
%H%M	=>	0837	Loc
%H:%M	=>	08:37	Loc
%Н	=>	08	Loc
%H%M%S,%L	=>	083748,000	Loc
%T,%L	=>	08:37:48,000	Loc
%H%M%S.%L	=>	083748.000	Loc
%T.%L	=>	08:37:48.000	Loc
%H%M%S%z	=>	083748-0600	Loc
%T%:z	=>	08:37:48-06:00	Loc
%Y%m%dT%H%M%S%z	=>	20071119T083748-0600	Dat
%FT%T%:z	=>	2007-11-19T08:37:48-06:00	Dat
%Y%jT%H%M%S%z	=>	2007323T083748-0600	Dat
%Y-%jT%T%:z	=>	2007-323T08:37:48-06:00	Dat
%GW%V%uT%H%M%S%z	=>	2007W471T083748-0600	Dat
%G-W%V-%uT%T%:z	=>	2007-W47-1T08:37:48-06:00	Dat
%Y%m%dT%H%M	=>	20071119T0837	Cal
%FT%R	=>	2007-11-19T08:37	Cal
%Y%jT%H%MZ	=>	2007323T0837Z	Ord
%Y-%jT%RZ	=>	2007-323T08:37Z	Ord
%GW%V%uT%H%M%z	=>	2007W471T0837-0600	Wee
%G-W%V-%uT%R%:z	=>	2007-W47-1T08:37-06:00	Wee
•			

#### Subsec → number

Returns the fraction for *time*.

The return value can be a rational number.



The lowest digits of <u>to\_f</u> and <u>subsec</u> are different because IEEE 754 double is not accurate enough to represent the rational number.

The more accurate value is returned by subsec.

#### osucc → new\_time

Returns a new <u>Time</u> object, one second later than *time*. <u>#succ</u> is obsolete since 1.9.2 for time is not a discrete value.



Use instead time + 1

t + 1		
<b>[4</b> ]		•

#### $_{\odot}$ sunday? $\rightarrow$ true or false

Returns true if time represents Sunday.

t = Time.local(1990, t.sunday?	4,	1)		
				Þ

#### $_{\odot}$ thursday? → true or false

Returns true if *time* represents Thursday.



#### oto\_a → array

Returns a ten-element array of values for time:



See the individual methods for an explanation of the valid ranges of each value. The ten elements can be passed directly to ::utc or ::local to create a new Time

object.



#### $\odot$ to\_f $\rightarrow$ float

Returns the value of *time* as a floating point number of seconds since the Epoch.



Note that IEEE 754 double is not accurate enough to represent the number of nanoseconds since the Epoch.

# 

#### $\otimes$ tv\_sec $\rightarrow$ int

Returns the value of *time* as an integer number of seconds since the Epoch.



#### oto\_r → a\_rational

Returns the value of *time* as a rational number of seconds since the Epoch.



This methods is intended to be used to get an accurate value representing the nanoseconds since the Epoch. You can use this method to convert *time* 

to another Epoch.

# inspect → string

#### os to\_s → string

Returns a string representing *time*. Equivalent to calling strftime with the appropriate format string.



## $_{\odot}$ tuesday? $\rightarrow$ true or false

Returns true if time represents Tuesday.



### $\otimes$ nsec $\rightarrow$ int

#### ostv\_nsec → int

Returns the number of nanoseconds for time.



The lowest digits of <u>to\_f</u> and <u>nsec</u> are different because IEEE 754 double is not accurate enough to represent the exact number of nanoseconds since the Epoch.

The more accurate value is returned by <u>nsec</u>.

# $\odot$ to\_i $\rightarrow$ int

#### $\otimes$ tv\_sec $\rightarrow$ int

Returns the value of *time* as an integer number of seconds since the Epoch.



#### $\otimes$ usec $\rightarrow$ int

#### $\odot$ tv\_usec $\rightarrow$ int

Returns the number of microseconds for time.



#### $\odot$ usec $\rightarrow$ int

#### ostv\_usec → int

Returns the number of microseconds for time.



#### $\odot$ gmtime $\rightarrow$ time

#### $\otimes$ utc $\rightarrow$ time

Converts time to UTC (GMT), modifying the receiver.

<pre>t = Time.now t.gmt? t.amtime</pre>		
t.gmt?		
t = Time.now		

t.utc?	#=>	false	
t.utc			
t.utc?			

#### rightarrow utc? → true or false rightarrow gmt? → true or false

Returns true if time represents a time in UTC (GMT).

<pre>t = Time.now t.utc? t = Time.gm(2000,"jan",1,20,15,1) t.utc?</pre>	
<pre>t = Time.now t.gmt? t = Time.gm(2000,1,1,20,15,1) t.gmt?</pre>	
4	•

# gmt\_offset → fixnum gmtoff → fixnum

## outc\_offset → fixnum

Returns the offset in seconds between the timezone of *time* and UTC.



### wday → fixnum

Returns an integer representing the day of the week, 0..6, with Sunday == 0.



t.monday?	#=> false
t.tuesday?	
t.wednesday?	
t.thursday?	
t.friday?	
t.saturday?	

#### $_{\odot}$ wednesday? $\rightarrow$ true or false

Returns true if time represents Wednesday.



### yday → fixnum

Returns an integer representing the day of the year, 1..366.



#### o year → fixnum

Returns the year for *time* (including the century).



#### one → string

Returns the name of the time zone used for *time*. As of Ruby 1.8, returns "UTC" rather than "GMT" for UTC times.

```
t = Time.gm(2000, "jan", 1, 20, 15, 1)
t.zone #=> "UTC"
t = Time.local(2000, "jan", 1, 20, 15, 1)
t.zone #=> "CST"
```

# class TracePoint

A class that provides the functionality of Kernel#set\_trace\_func in a nice Object-Oriented API.

# Example

We can use <u>TracePoint</u> to gather information specifically for exceptions:



## **Events**

If you don't specify the type of events you want to listen for, <u>TracePoint</u> will include all available events.

**Note** do not depend on current event set, as this list is subject to change. Instead, it is recommended you specify the type of events you want to use.

To filter what is traced, you can pass any of the following as events:

:line

execute code on a new line

:class

start a class or module definition

:end

finish a class or module definition

:call

call a Ruby method

:return

return from a Ruby method

:c\_call

call a C-language routine

:c\_return

return from a C-language routine

:raise

raise an exception

:b\_call

event hook at block entry

:b\_return

event hook at block ending

:thread\_begin

event hook at thread beginning

:thread\_end

event hook at thread ending

# **In Files**

vm\_trace.c

## Parent

Object

# **Public Class Methods**

### mew(\*events) { |obj| block } → obj

Returns a new <u>TracePoint</u> object, not enabled by default.

Next, in order to activate the trace, you must use **#enable** 



When you want to deactivate the trace, you must use **#disable** 

```
trace.disable
See Events at TracePoint for possible events and
```

more information.

A block must be given, otherwise a <u>ThreadError</u> is raised.

If the trace method isn't included in the given events filter, a **RuntimeError** is raised.



If the trace method is called outside block, a RuntimeError is raised.



Access from other threads is also forbidden.

### ⊚ stat → obj

Returns internal information of TracePoint.

The contents of the returned value are implementation specific. It may be changed in future.

This method is only for debugging TracePoint itself.

### $_{\odot}$ trace(\*events) { |obj| block } $\rightarrow$ obj

A convenience method for <u>::new</u>, that activates the trace automatically.



# **Public Instance Methods**

## binding()

Return the generated binding object from event

## defined\_class()

Return class or module of the method being called.

```
class C; def foo; end; end
trace = TracePoint.new(:call) do |tp|
   p tp.defined_class #=> C
```

```
end.enable do
C.new.foo
end
```

If method is defined by a module, then that module is returned.

```
module M; def foo; end; end
class C; include M; end;
trace = TracePoint.new(:call) do |tp|
   p tp.defined_class #=> M
end.enable do
   C.new.foo
end
```

**Note:** defined\_class returns singleton class.

6th block parameter of Kernel#set\_trace\_func passes original class of attached by singleton class.

This is a difference between Kernel#set\_trace\_func and TracePoint.

```
class C; def self.foo; end; end
trace = TracePoint.new(:call) do |tp|
   p tp.defined_class #=> #<Class:C>
end.enable do
   C.foo
end
```

## $\odot$ disable $\rightarrow$ true or false

 $\odot$  disable { block }  $\rightarrow$  obj

Deactivates the trace

Return true if trace was enabled. Return false if trace was disabled.



If a block is given, the trace will only be disable within the scope of the block.



Note: You cannot access event hooks within the block.

trace.disable { p tp.lineno }
#=> RuntimeError: access from outside

### $_{\odot}$ enable $\rightarrow$ true or false

#### $\odot$ enable { block } $\rightarrow$ obj

Activates the trace

Return true if trace was enabled. Return false if trace was disabled.



If a block is given, the trace will only be enabled within the scope of the block.



trace.enabled?
#=> false

end

Note: You cannot access event hooks within the block.

trace.enable { p tp.lineno }
#=> RuntimeError: access from outside

#### $_{\odot}$ enabled? $\rightarrow$ true or false

The current status of the trace

## 🚳 event()

Type of event

See Events at TracePoint for more information.

#### inspect → string

Return a string containing a human-readable TracePoint status.

### 🚳 lineno()

Line number of the event

### method\_id()

Return the name of the method being called

### 🚳 path()

Path of the file being run

### raised\_exception()

Value from exception raised on the :raise event

#### return\_value()

Return value from :return, c\_return, and b\_return event

#### 🚳 self()

Return the trace object during event

Same as **#binding**:

trace.binding.eval('self')

# class TrueClass

The global value true is the only instance of class TrueClass and represents a logically true value in boolean expressions. The class provides operators allowing true to be used in logical expressions.

# In Files

object.c

## Parent

Object

# **Public Instance Methods**

### true & obj → true or false

And—Returns false if *obj* is nil Or false, true otherwise.

## ⊚ true ^ obj → !obj

Exclusive Or—Returns true if *obj* is nil Or false, false otherwise.

## 🍙 inspect()

Alias for: to\_s

#### s → "true"

The string representation of true is "true".

Also aliased as: inspect

#### obj → true

Or—Returns true. As *obj* is an argument to a method call, it is always evaluated; there is no short-circuit evaluation in this case.

```
true | puts("or")
true || puts("logical or")
```

produces:

or

```
Generated by <u>RDoc</u> 3.12.2.
Generated with the <u>Darkfish Rdoc Generator</u> 3.
```

# class TypeError

Raised when encountering an object that is not of the expected type.



raises the exception:

TypeError: no implicit conversion of String in

In Files

Parent

StandardError

# class UnboundMethod

Ruby supports two forms of objectified methods. <u>Class</u> Method is used to represent methods that are associated with a particular object: these method objects are bound to that object. Bound method objects for an object can be created using Object#method.

Ruby also supports unbound methods; methods objects that are not associated with a particular object. These can be created either by calling Module#instance\_method Or by calling unbind On a bound method object. The result of both of these is an UnboundMethod Object.

Unbound methods can only be called after they are bound to an object. That object must be be a kind\_of? the method's original class.

```
class Square
  def area
    @side * @side
  end
  def initialize(side)
    @side = side
  end
end
area_un = Square.instance_method(:area)
s = Square.new(12)
area = area_un.bind(s)
area.call #=> 144
```

Unbound methods are a reference to the method at the time it was objectified: subsequent changes to the underlying class will not affect the unbound method.



# In Files

proc.c

## Parent

**Object** 

# **Public Instance Methods**

eql?(other\_meth) → true or false
 meth == other\_meth → true or false
 Two method objects are equal if they are bound to

the same object and refer to the same method definition and their owners are the same class or module.

#### arity → fixnum

Returns an indication of the number of arguments accepted by a method. Returns a nonnegative integer for methods that take a fixed number of arguments. For Ruby methods that take a variable number of arguments, returns -n-1, where n is the number of required arguments. For methods written in C, returns -1 if the call takes a variable number of arguments.

```
class C
  def one;
              end
  def two(a); end
  def three(*a);
                  end
  def four(a, b); end
  def five(a, b, *c);
                         end
  def six(a, b, *c, &d); end
end
c = C.new
c.method(:one).arity
c.method(:two).arity
c.method(:three).arity
c.method(:four).arity
c.method(:five).arity
c.method(:six).arity
"cat".method(:size).arity
"cat".method(:replace).arity
"cat".method(:squeeze).arity
"cat".method(:count).arity
```

#### bind(obj) → method

Bind *umeth* to *obj*. If Klass was the class from which *umeth* was obtained, obj.kind\_of?(Klass) must be true.

```
class A
  def test
   puts "In test, class = #{self.class}"
  end
end
class B < A
end
class C < B
end
um = B.instance_method(:test)
bm = um.bind(C.new)
bm.call
bm = um.bind(B.new)
bm.call
bm = um.bind(A.new)
bm.call
```

produces:

```
In test, class = C
In test, class = B
prog.rb:16:in %xbind': bind argument must be an :
from prog.rb:16
```

#### osignment of the second se

Returns a clone of this method.



in eql?(other\_meth) → true or false
 in meth == other\_meth → true or false
 Two method objects are equal if they are bound to

the same object and refer to the same method definition and their owners are the same class or module.

#### $\otimes$ hash $\rightarrow$ integer

Returns a hash value corresponding to the method object.

See also Object#hash.

## ⊚to\_s → string

#### inspect → string

Returns the name of the underlying method.



#### on name → symbol

Returns the name of the method.

#### $\odot$ original\_name $\rightarrow$ symbol

Returns the original name of the method.

#### owner → class\_or\_module

Returns the class or module that defines the method.

#### 

Returns the parameter information of this method.

#### $\odot$ source\_location $\rightarrow$ [String, Fixnum]

Returns the Ruby source filename and line number containing this method or nil if this method was not

defined in Ruby (i.e. native)

### super\_method()

Returns a <u>Method</u> of superclass, which would be called when super is used.

# $\odot$ to\_s $\rightarrow$ string

#### inspect → string

Returns the name of the underlying method.

	<pre>cat".method(:count).inspect</pre>		
4			= )• î

# class UncaughtThrowError

Raised when throw is called with a *tag* which does not have corresponding catch block.

throw "foo", "bar"

raises the exception:

UncaughtThrowError: uncaught throw "foo"

In Files

📄 vm\_eval.c

Parent

ArgError

## **Public Class Methods**

#### 🚳 new(\*args)

Document-class: UncaughtThrowError

Raised when throw is called with a *tag* which does not have corresponding catch block.

throw "foo", "bar"

raises the exception:

UncaughtThrowError: uncaught throw "foo"

# **Public Instance Methods**

rightarrow tag 
ightarrow object which was called for.

#### ● **to\_s** → **string** Returns formatted message with the inspected tag.

 $\Rightarrow$  value  $\rightarrow$  obj Return the return value which was called for.

# class ZeroDivisionError

Raised when attempting to divide an integer by 0.



Note that only division by an exact 0 will raise the exception:

```
42 / 0.0 #=> Float::INFINITY
42 / -0.0 #=> -Float::INFINITY
0 / 0.0 #=> NAN
```

# **In Files**

📄 numeric.c

## Parent

StandardError

# class fatal

fatal is an <u>Exception</u> that is raised when ruby has encountered a fatal error and must exit. You are not able to rescue fatal.

# **In Files**

error.c

## Parent

Exception

# Ruby 2.2.4 Core API Reference API Reference

This is the API documentation for 'Ruby 2.2.4 Core API Reference API Reference'.