# Ruby 2.2.4 Core API Reference API Reference 

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

## Classes/Modules

${ }^{\circ} \mathrm{I}$
IO::EAGAINWaitReadable
IO::EAGAINWaitWritable
IO::EINPROGRESSWaitReadable
IO::EINPROGRESSWaitWritable
IO::EWOULDBLOCKWaitReadable
IO::EWOULDBLOCKWaitWritable
IO::WaitReadable
10::WaitWritable
${ }^{5}$ Encoding
Encoding::CompatibilityError
Encoding::Converter
Encoding::ConverterNotFoundError
Encoding:InvalidByteSequenceError
Encoding::UndefinedConversionError
Process
Process::GID
Process::Status
Process::Sys
Process::UID
Process::Waiter

```
Enumerator
Enumerator::Generator
Enumerator::Lazy
Enumerator::Yielder
File
File::Constants
File::Stat
RubyVM
RubyVM::Env
RubyVM::InstructionSequence
Complex
Complex::compatible
GC
GC:: Profiler
Math
Math::DomainError
ObjectSpace
ObjectSpace::WeakMap
Rational
Rational::compatible
ArgumentError
Array
BasicObject
```

```
Bignum
Binding
Class
(1) Comparable
Continuation
Data
Dir
ENV
EOFError
EncodingError
Enumerable
Errno
Exception
FalseClass
Fiber
FiberError
FileTest
Fixnum
\({ }^{\sigma}\) Float
FloatDomainError
Hash
IOError
IndexError
```

```
Integer
Interrupt
Kernel
KeyError
LoadError
LocalJumpError
Marshal
MatchData
Method
Module
Mutex
NameError
NilClass
NoMemoryError
NoMethodError
NotImplementedError
Numeric
Object
Proc
Random
Range
RangeError
Regexp
```

RegexpError
Ripper
RuntimeError
ScriptError
SecurityError
Signal
SignalException
StandardError
Stoplteration
String
Struct
Symbol
SyntaxError
SystemCallError
${ }^{\sigma}$ SystemExit
SystemStackError
Thread
ThreadError
ThreadGroup
Time
TracePoint
TrueClass
TypeError

UnboundMethod
UncaughtThrowError
ZeroDivisionError
fatal

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## class ArgumentError

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

Ex: passing the wrong number of arguments
$[1,2,3] . f i r s t(4,5)$
raises the exception:


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 RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 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 Integer, a String and a Float:

```
ary = [1, "two", 3.0]
```

An array can also be created by explicitly calling $\because: \because n e w$ with zero, one (the initial size of the Array) or two arguments (the initial size and a default object).

```
ary = Array.new
Array.new(3)
Array.new(3, 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:

Array.new(4) \{ Hash.new \}

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


An array can also be created by using the Array() method, provided by Kernel, which tries to call to_ary, then to_a on its argument.

Array(\{:a => "a", :b => "b"\}) 4 1

## Example Usage

In addition to the methods it mixes in through the Enumerable module, the Array 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.

```
arr = [1, 2, 3, 4, 5, 6]
arr[2]
arr[100]
arr[-3]
arr[2, 3]
arr[1..4]
arr[1..-3]
```

Another way to access a particular array element is by using the at method

```
arr.at(0)
```

The slice 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 fetch.


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

```
arr.first
arr.last
```

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

```
arr.take(3)
```

drop does the opposite of take, by returning the elements after n elements have been dropped:
arr.drop(3)

## Obtaining Information about an Array

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


To check whether an array contains any elements at all
browsers.empty?
To check whether a particular item is included in the array
browsers.include?('Konqueror')

## Adding Items to Arrays

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

```
arr = [1, 2, 3, 4]
arr.push(5)
arr << 6
```

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

```
arr.unshift(0)
```

With insert you can add a new element to an array at any position.


Using the insert 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
arr
```

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

```
arr.shift
arr
```

To delete an element at a particular index:

```
arr.delete_at(2)
arr
```

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

```
arr = [1, 2, 2, 3]
arr.delete(2)
arr
```

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

```
arr = ['foo', 0, nil, 'bar', 7, 'baz', nil]
arr.compact ##> ['foo', 0, 'bar', 7, 'baz'
```



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

It has the non-destructive uniq, and destructive method 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 each, all elements in the Array instance are yielded to the supplied block in sequence.

Note that this operation leaves the array unchanged.

```
arr = [1, 2, 3, 4, 5]
arr.each { |a| print a -= 10, " " }
```

Another sometimes useful iterator is reverse_each which will iterate over the elements in the array in reverse order.

```
words = %w[first second third fourth fifth si,
str = ""
words.reverse_each { |word| str += "#{word}
p str

The map 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| 2*a }

```
arr


\section*{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
\begin{tabular}{ll}
\begin{tabular}{l} 
arr \(=[1,2,3,4,5,6]\) \\
arr.select \(\{|a| a>3\}\) \\
arr.reject \(\{|a| a<3\}\) \\
arr.drop_while \(\{|a| a<4\}\) \\
arr
\end{tabular} & \(: \Rightarrow[4,5,6]\) \\
\hline
\end{tabular}

\section*{Destructive Selection}
select! and reject! are the corresponding destructive methods to select and reject

Similar to select vs. reject, delete_if and keep_if have the exact opposite result when supplied with the same block:
```

arr.delete_if { |a| a<4 }
arr
arr = [1, 2, 3, 4, 5, 6]
arr.keep_if { |a| a < 4 }

```

\section*{In Files}
- array.c
- pack.c

\section*{Parent}

\section*{Object}

\section*{Included Modules}
- Enumerable

\section*{Public Class Methods}

\section*{[](*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 \#to_ary 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 }

```

\section*{Common gotchas}

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

a = Array.new(2, Hash.new)
a[0]['cat'] = 'feline'
a
a[1]['cat'] = 'Felix'
a

```

Since all the Array 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

```

\section*{try_convert(obj) \(\rightarrow\) 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.
```

Array.try_convert([1])
Array.try_convert("1") \#=> ni.1
if tmp = Array.try_convert(arg)

```
```

elsif tmp = String.try_convert(arg)
end

```

\section*{Public Instance Methods}

\author{
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 hash and eql? methods for efficiency.


See also \#uniq.
ary * int \(\rightarrow\) new_ary
ary * str \(\rightarrow\) 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 \(\rightarrow\) 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.

\section*{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 hash and eql? methods for efficiency.


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

\section*{ary << obj \(\rightarrow\) 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.

\section*{[1, 2 ] << "c" << "d" << [ 3, 4 ]}

\section*{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.


\section*{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] \(\rightarrow\) obj or nil
ary[start, length] \(\rightarrow\) new_ary or nil
ary[range] \(\rightarrow\) new_ary or nil
slice(index) \(\rightarrow\) obj or nil
slice(start, length) \(\rightarrow\) new_ary or nil
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.

\(\mathrm{a}[4 . .7]\)
\(\mathrm{a}[6 . .10]\)
\(\mathrm{a}[-3,3]\)
\#special cases
\(\mathrm{a}[5]\)
\(\mathrm{C}[6,1]\)
\(\mathrm{a}[5,1]\)
\(\mathrm{a}[5,10]\)
ary[index] = obj \(\rightarrow\) obj
ary[start, length] = obj or other_ary or nil \(\rightarrow\) obj or other_ary or nil
ary[range] = obj or other_ary or nil \(\rightarrow\) obj or 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[0, 0] = [ 1, 2 ]
a[3, 0] = "B"

```
any? [\{ |obj| block \}] \(\rightarrow\) true or false
See also Enumerable\#any?

\section*{assoc(obj) \(\rightarrow\) 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


\section*{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\#П.
```

a = [ "a", "b", "c", "d", "e" ]
a.at(0)
a.at(-1)

```
which meets the given condition in \(\mathrm{O}(\log \mathrm{n})\) 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 ( \(0<=\mathrm{i}<=\) 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 ( \(0<=\mathrm{i}<=\mathrm{j}<=\) ary.size) so that:
@ the block returns a positive number for ary if 0 \(<=\mathrm{k}<\mathrm{i}\),
© the block returns zero for ary if \(\mathrm{i}<=\mathrm{k}<\mathrm{j}\), and
® the block returns a negative number for ary if \(j\) <= k < ary.size.

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.
```

ary = [0, 4, 7, 10, 12]
ary.bsearch {|x| 1 - x / 4 }
ary.bsearch {|x| 4 - x / 2 }

```

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.

\section*{clear \(\rightarrow\) ary}

Removes all elements from self.
```

a = [ "a", "b", "c", "d", "e" ]
a.clear

```
collect \{ |item| block \} \(\rightarrow\) new_ary
map \{ |item| block \} \(\rightarrow\) new_ary
collect \(\rightarrow\) Enumerator
map \(\rightarrow\) 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.

collect! \{|item| block \} \(\rightarrow\) ary
map! \{|item| block \} \(\rightarrow\) ary
collect! \(\rightarrow\) Enumerator map! \(\rightarrow\) 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.
```

a = [ "a", "b", "c", "d" ]
a.map! {|x| x + "!" }
a
a.collect!.with_index {|x, i| x[0...i] }
a

```

\section*{combination(n) \{ |c| block \(\} \rightarrow\) ary} combination(n) \(\rightarrow\) 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:

compact \(\rightarrow\) new_ary
Returns a copy of self with all nil elements removed.
```

[ "a", nil, "b", nil, "c", nil ].compact

```

\section*{compact! \(\rightarrow\) ary or nil}

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

concat(other_ary) \(\rightarrow\) ary
Appends the elements of other_ary to self.


See also Array\#+.
count \(\rightarrow\) int
count(obj) \(\rightarrow\) int
count \{ |item| block \} \(\rightarrow\) 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.
```

ary = [1, 2, 4, 2]
ary.count
ary.count(2)
ary.count { |x| x%2 == 0 }

```

\section*{cycle(n=nil) \{ |obj| block \} \(\rightarrow\) nil}
cycle(n=nil) \(\rightarrow\) 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 Enumerator is returned instead.
```

a = ["a", "b", "c"]
a.cycle { |x| puts x }
a.cycle(2) { |x| puts x }

```

\section*{delete(obj) \(\rightarrow\) item or nil} delete(obj) \{ block \} \(\rightarrow\) 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!)
```

a = [ "a", "b", "b", "b", "c" ]
a.delete("b")
a
a.delete("z")
a.delete("z") { "not found" } \#=> "not found

```

\section*{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!
```

a = ["ant", "bat", "cat", "dog"]
a.delete_at(2)
a
a.delete_at(99)

```
delete_if \(\{\) |item| block \(\} \rightarrow\) ary
delete_if \(\rightarrow\) 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 Enumerator is returned instead.
```

scores = [ 97, 42, 75 ]
scores.delete_if {|score| score < 80 }

```

\section*{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
```

a = [1, 2, 3, 4, 5, 0]
a.drop(3)

```

\section*{drop_while \(\{\) |arr| block \} \(\rightarrow\) new_ary}
© drop_while \(\rightarrow\) 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.

See also \#take_while
```

a = [1, 2, 3, 4, 5, 0]
a.drop_while {|i| i < 3 }

```
each \{ |item| block \} \(\rightarrow\) ary
each \(\rightarrow\) 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.
```

a = [ "a", "b", "c" ]

```
produces:
```

a -- b -- c --

```

Same as \#each, but passes the index of the element instead of the element itself.

An Enumerator is returned if no block is given.
```

a = [ "a", "b", "c" ]
a.each_index {|x| print x, " -- " }

```
produces:
```

0 -- 1 -- 2 --

```
empty? \(\rightarrow\) true or false
Returns true if self contains no elements.
[].empty?

\section*{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) \(\rightarrow\) obj fetch(index, default) \(\rightarrow\) obj
fetch(index) \{ |index| block \} \(\rightarrow\) 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.

fill(obj) \(\rightarrow\) ary
fill(obj, start [, length]) \(\rightarrow\) ary
fill(obj, range ) \(\rightarrow\) ary
fill \{ |index| block \} \(\rightarrow\) ary
fill(start [, length] ) \{ |index| block \} \(\rightarrow\) ary 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) \(\rightarrow\) int or nil
find_index \{ |item| block \} \(\rightarrow\) int or nil
find_index \(\rightarrow\) Enumerator index(obj) \(\rightarrow\) int or nil
index \{ |item| block \} \(\rightarrow\) int or nil index \(\rightarrow\) 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 \(\rightarrow\) obj or nil
first(n) \(\rightarrow\) 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.
```

a = [ "q", "r", "s", "t" ]
a.first
a.first(2)

```
flatten \(\rightarrow\) new_ary
flatten(level) \(\rightarrow\) 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.


\section*{flatten! \(\rightarrow\) ary or nil}
flatten!(level) \(\rightarrow\) 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.
```

a = [ 1, 2, [3, [4, 5] ] ]
a.flatten!
a.flatten!
a
a = [ 1, 2, [3, [4, 5] ] ]
a.flatten!(1)

```

\section*{frozen? \(\rightarrow\) true or false}

Return true if this array is frozen (or temporarily frozen while being sorted). See also Object\#frozen?
hash \(\rightarrow\) fixnum
Compute a hash-code for this array.
Two arrays with the same content will have the same
hash code (and will compare using eql?).
See also Object\#hash.

\section*{include?(object) \(\rightarrow\) 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")
a.include?("z")
\#\#=
find_index(obj) \(\rightarrow\) int or nil
find_index \{ |item| block \} \(\rightarrow\) int or nil
find_index \(\rightarrow\) Enumerator
index(obj) \(\rightarrow\) int or nil
index \{ |item| block \} \(\rightarrow\) int or nil index \(\rightarrow\) 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.


\section*{replace(other_ary) \(\rightarrow\) ary} initialize_copy(other_ary) \(\rightarrow\) ary
Replaces the contents of self with the contents of other_ary, truncating or expanding if necessary.

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.

inspect \(\rightarrow\) string
to_s \(\rightarrow\) string
Creates a string representation of self.


Also aliased as: to_s
join(separator=\$,) \(\rightarrow\) 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.
[ "a", "b", "c" ].join
[ "a", "b", "c" \(]. j o i n("-") ~\)

\section*{keep_if \{ |item| block \} \(\rightarrow\) ary}
keep_if \(\rightarrow\) Enumerator
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]/ }

```
last \(\rightarrow\) obj or nil
last(n) \(\rightarrow\) 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.
```

a = [ "w", "x", "y", "z" ]
a.last
a.last(2)

```

\section*{length \(\rightarrow\) int}

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

[].length

```

Also aliased as: size
collect \{ |item| block \} \(\rightarrow\) new_ary map \{ |item| block \} \(\rightarrow\) new_ary collect \(\rightarrow\) Enumerator
map \(\rightarrow\) 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.

collect! \{|item| block \} \(\rightarrow\) ary
map! \{|item| block \} \(\rightarrow\) ary
collect! \(\rightarrow\) Enumerator
map! \(\rightarrow\) 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.

\footnotetext{
a = [ "a", "b", "c", "d" ]
}
```

a.map! {|x| x + "!" }
a
a.collect!.with_index {|x, i| x[0...i] }
a

```

\section*{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.
\begin{tabular}{|c|c|c|}
\hline Integer & Array & \\
\hline Directive & Element & Meaning \\
\hline C & Integer & 8-bit unsigned \\
\hline S & Integer & 16-bit unsigned, \\
\hline L & Integer & 32-bit unsigned, \\
\hline Q & Integer & 64-bit unsigned, \\
\hline C & Integer & 8-bit signed (si \\
\hline s & Integer & 16-bit signed, \\
\hline
\end{tabular}
\begin{tabular}{|c|c|c|}
\hline 1 & Integer & 32-bit signed, native er \\
\hline q & Integer & 64-bit signed, native er \\
\hline & & \\
\hline S_, S! & Integer & unsigned short, native \\
\hline I, \(\mathrm{I}_{-}\)I! & Integer & unsigned int, native end \\
\hline L_, L! & Integer & unsigned long, native er \\
\hline \(Q_{-,} Q^{\text {! }}\) & Integer & unsigned long long, nat: \\
\hline & & if the platform has no \\
\hline & & (Q_ and Q! is available \\
\hline & & \\
\hline S_, S! & Integer & signed short, native enc \\
\hline i, i_, i! & Integer & signed int, native endia \\
\hline \(l_{-\prime} \mathrm{l}\) ! & Integer & signed long, native end- \\
\hline \(\mathrm{q}_{-,} \mathrm{q}!\) & Integer & signed long long, native \\
\hline & & if the platform has no \\
\hline & & (q_ and q! is available \\
\hline & & \\
\hline S> L> Q> & Integer & same as the directives \\
\hline s> l> q> & & big endian \\
\hline S! > I! \({ }^{\text {l }}\) & & (available since Ruby 1. \\
\hline L!> Q!> & & "S>" is same as "n" \\
\hline S!> i! > & & "L>" is same as "N" \\
\hline l!> q! > & & \\
\hline & & \\
\hline \(\mathrm{S}<\mathrm{L}<\mathrm{Q}<\) & Integer & same as the directives \\
\hline \(\mathrm{S}<\mathrm{l}<\mathrm{q}<\) & & little endian \\
\hline S! < I! < & & (available since Ruby 1. \\
\hline \(\mathrm{L}!<\mathrm{Q}!<\) & & "S<" is same as "v" \\
\hline S! < i! < & & "L<" is same as "V" \\
\hline \(1!<\mathrm{q}!<\) & & \\
\hline & & \\
\hline n & Integer & 16-bit unsigned, networt \\
\hline N & Integer & 32-bit unsigned, networt \\
\hline v & Integer & 16-bit unsigned, VAX (li \\
\hline V & Integer & 32-bit unsigned, VAX (li \\
\hline & & \\
\hline U & Integer & UTF-8 character \\
\hline w & Integer & BER-compressed integer \\
\hline Float & & \\
\hline Directive & & Meaning \\
\hline D, d & Float & double-precision, native \\
\hline F, f & Float & single-precision, nativ¢ \\
\hline E & Float & double-precision, little \\
\hline e & Float & single-precision, little \\
\hline G & Float & double-precision, networ \\
\hline
\end{tabular}
\begin{tabular}{|c|c|c|}
\hline g & Float & single-precision, networ \\
\hline String & & \\
\hline Directive & & Meaning \\
\hline A & String & arbitrary binary string \\
\hline a & String & arbitrary binary string \\
\hline Z & String & same as ``a'', except th \\
\hline B & String & bit string (MSB first) \\
\hline b & String & bit string (LSB first) \\
\hline H & String & hex string (high nibble \\
\hline h & String & hex string (low nibble \\
\hline u & String & UU-encoded string \\
\hline M & String & quoted printable, MIME \\
\hline m & String & base64 encoded string ( (if count is 0, no line \\
\hline P & String & pointer to a structure \\
\hline p & String & pointer to a null-termir \\
\hline Misc. & & \\
\hline Directive & & Meaning \\
\hline @ & & moves to absolute posiṫ \\
\hline X & --- & back up a byte \\
\hline X & --- & null byte \\
\hline
\end{tabular}

\section*{permutation \{ |p| block \} \(\rightarrow\) ary} permutation \(\rightarrow\) Enumerator permutation(n) \{ |p| block \} \(\rightarrow\) ary permutation(n) \(\rightarrow\) 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.

\section*{Examples:}


\section*{pop \(\rightarrow\) obj or nil}
pop(n) \(\rightarrow\) new_ary
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.
```

a = [ "a", "b", "c", "d" ]
a.pop
a.pop(2)
a

```

\section*{product(other_ary, ...) \(\rightarrow\) new_ary} product(other_ary, ...) \{ |p| block \} \(\rightarrow\) 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.
```

[1,2].product([3,4],[5,6])
[1,2].product()
[1,2].product([])

```
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 \#pop for the opposite effect.
```

a = [ "a", "b", "c" ]
a.push("d", "e", "f")
[1, 2, 3,].push(4).push(5)

```

\section*{rassoc(obj) \(\rightarrow\) 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.


\section*{reject \(\{|i t e m|\) block \} \(\rightarrow\) new_ary}
reject \(\rightarrow\) Enumerator
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.

\section*{reject! \{ |item| block \} \(\rightarrow\) ary or nil reject! \(\rightarrow\) Enumerator}

Equivalent to \#delete_if, 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 Enumerator is returned instead.

\section*{repeated_combination(n) \{ |c| block \} \(\rightarrow\) ary repeated_combination(n) \(\rightarrow\) 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.

\section*{Examples:}

```

a.repeated_combination(0).to_a

```
repeated_permutation(n) \{ |p| block \} \(\rightarrow\) ary repeated_permutation(n) \(\rightarrow\) 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 Enumerator is returned instead.

\section*{Examples:}
```

a = [1, 2]
a.repeated_permutation(1).to_a
a.repeated_permutation(2).to_a
a.repeated_permutation(3).to_a
a.repeated_permutation(0).to_a

```
replace(other_ary) \(\rightarrow\) ary
initialize_copy(other_ary) \(\rightarrow\) ary
Replaces the contents of self with the contents of other_ary, truncating or expanding if necessary.

reverse \(\rightarrow\) new_ary
Returns a new array containing self's elements in
reverse order.
```

[ "a", "b", "c" ].reverse
[ 1 ].reverse

```
reverse! \(\rightarrow\) ary
Reverses self in place.
```

a = [ "a", "b", "c" ]
a.reverse!
a

```
(3) reverse_each \{ |item| block \} \(\rightarrow\) ary
reverse_each \(\rightarrow\) Enumerator
Same as \#each, but traverses self in reverse order.
```

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

```
produces:
```

c b a

```

\section*{rindex(obj) \(\rightarrow\) int or nil}
rindex \{ |item| block \} \(\rightarrow\) int or nil
sindex \(\rightarrow\) 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 \#index.
If neither block nor argument is given, an Enumerator
is returned instead.
```

a = [ "a", "b", "b", "b", "c" ]
a.rindex("b")
a.rindex("z")
a.rindex { |x| x == "b" }

```

\section*{rotate(count=1) \(\rightarrow\) 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.


\section*{rotate!(count=1) \(\rightarrow\) 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.
```

a = [ "a", "b", "c", "d" ]
a.rotate!
a
a.rotate!(2)
a.rotate!(-3)

```
sample \(\rightarrow\) obj
sample(random: rng) \(\rightarrow\) obj
sample(n) \(\rightarrow\) new_ary
sample(n, random: rng) \(\rightarrow\) 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.
```

a = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
a.sample
a.sample(4)

```

\section*{select \{ |item| block \} \(\rightarrow\) new_ary} select \(\rightarrow\) 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! \(\{\mid\) item| block \(\} \rightarrow\) ary or nil
select! \(\rightarrow\) 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 Enumerator is returned instead.
shift \(\rightarrow\) obj or nil
shift(n) \(\rightarrow\) new_ary
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.
\begin{tabular}{|c|}
\hline ```
args = [ "-m", "-q", "filename" ]
args.shift
args
``` \\
\hline  \\
\hline
\end{tabular}
shuffle \(\rightarrow\) new_ary
shuffle(random: rng) \(\rightarrow\) new_ary
Returns a new array with elements of self shuffled.
\[
\begin{aligned}
& \mathrm{a}=[1,2,3] \\
& \text { a. shuffle }
\end{aligned}
\]

The optional rng argument will be used as the random number generator.
a.shuffle(random: Random.new(1))
shuffle! \(\rightarrow\) ary
shuffle!(random: rng) \(\rightarrow\) ary
Shuffles elements in self in place.
```

a = [1, 2, 3 ]
a.shuffle!
a

```

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

size()
Alias for: length
ary[index] \(\rightarrow\) obj or nil
ary[start, length] \(\rightarrow\) new_ary or nil ary[range] \(\rightarrow\) new_ary or nil
slice(index) \(\rightarrow\) obj or nil
slice(start, length) \(\rightarrow\) new_ary or nil
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.


\section*{slice!(index) \(\rightarrow\) obj or nil}
slice!(start, length) \(\rightarrow\) new_ary or nil
slice!(range) \(\rightarrow\) 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.

8) sort \(\rightarrow\) new_ary
sort \{ |a, b| block \} \(\rightarrow\) 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.


\section*{sort! \(\rightarrow\) ary}
sort! \{ |a, b| block \} \(\rightarrow\) ary
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 \(\mathrm{b}, 0\) when a and b are equivalent, or +1 if \(b\) follows \(a\).

See also Enumerable\#sort_by.

sort_by! \{ |obj| block \} \(\rightarrow\) ary

\section*{sort_by! \(\rightarrow\) 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.
take(n) \(\rightarrow\) new_ary
Returns first n elements from the array.
If a negative number is given, raises an
ArgumentError.
See also \#drop
```

a = [1, 2, 3, 4, 5, 0]
a.take(3)

```
take_while \{ |arr| block \} \(\rightarrow\) new_ary take_while \(\rightarrow\) 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 Enumerator is returned instead.

See also \#drop_while
```

a = [1, 2, 3, 4, 5, 0]
a.take_while { |i| i< < }

```

\section*{to_a \(\rightarrow\) ary}

Returns self.
If called on a subclass of Array, converts the receiver to an Array object.
to_ary \(\rightarrow\) ary
Returns self.
to_h \(\rightarrow\) hash
Returns the result of interpreting ary as an array of [key, value] pairs.
[[:foo, :bar], [1, 2]].to_h
to_s()
Alias for: inspect
transpose \(\rightarrow\) new_ary
Assumes that self is an array of arrays and
transposes the rows and columns.
```

a = [[1,2], [3,4], [5,6]]
a.transpose

```

If the length of the subarrays don't match, an
IndexError is raised.
uniq \(\rightarrow\) new_ary
uniq \(\{\) |item| ... \} \(\rightarrow\) 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 hash and eql? methods for efficiency.
a = [ "a", "a", "b", "b", "c" ]
```

a.uniq
b = [["student","sam"], ["student","george"], ["
b.uniq { |s| s.first } \# => [["student", "sam"]

```

\section*{uniq! \(\rightarrow\) ary or nil}

\section*{uniq! \(\{\) |item| ... \} \(\rightarrow\) 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 hash and eql? methods for efficiency.
Returns nil if no changes are made (that is, no duplicates are found).


\section*{unshift(obj, ...) \(\rightarrow\) ary}

Prepends objects to the front of self, moving other elements upwards. See also \#shift for the opposite effect.
```

a = [ "b", "c", "d" ]
a.unshift("a")
a.unshift(1, 2)

```
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.

sip(arg, ...) \(\rightarrow\) new_ary
zip(arg, ...) \{ |arr| block \} \(\rightarrow\) 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.


\section*{ary | other_ary \(\rightarrow\) 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 hash and eql? methods for efficiency.


See also \#uniq.

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{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 BasicObject for other users an appropriately named subclass of BasicObject should be created instead of directly modifying BasicObject:

\section*{class MyObjectSystem < BasicObject end}

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 BasicObject. 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
```

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_priva
DELEGATE.include?(name) or super
end
end

```

Access to classes and modules from the Ruby standard library can be obtained in a BasicObject 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

```

\section*{In Files}
© class.c
- gc.c
\({ }^{-}\)object.c
- vm_eval.c

\section*{Parent}

\section*{Public Class Methods}
new()
Not documented

\section*{Public Instance Methods}

\section*{!obj \(\rightarrow\) true or false}

Boolean negate.

\section*{obj != other \(\rightarrow\) true or false}

Returns true if two objects are not-equal, otherwise false.
obj \(==\) other \(\rightarrow\) true or false
equal?(other) \(\rightarrow\) true or false
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

```
obj == other
obj.equal? other
obj. equal? obj
The eql? method returns true if obj and other refer to the same hash key. This is used by Hash 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
1.eql? 1.0

```
© __id__ \(\rightarrow\) integer
object_id \(\rightarrow\) 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...]) \(\rightarrow\) obj
5 __send__(symbol [, args...]) \(\rightarrow\) obj send(string [, args...]) \(\rightarrow\) obj
\& __send__(string [, args...]) \(\rightarrow\) obj
Invokes the method identified by symbol, passing it any arguments specified. You can use \(\qquad\) send \(\qquad\) 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.

obj \(==\) other \(\rightarrow\) true or false
equal?(other) \(\rightarrow\) true or false
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
obj == other
obj.equal? other
obj.equal? obj

```

The eql? method returns true if obj and other refer to the same hash key. This is used by Hash 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
1.eql? 1.0

```
instance_eval(string [, filename [, lineno]] )
\(\rightarrow\) obj
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}."

```
```

    end
    end
k = KlassWithSecret.new
k.instance_eval { @secret }
k.instance_eval { the_secret }
k.instance_eval {|obj| obj == self } \#\#\# true

```
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.
```

class KlassWithSecret
def initialize
@secret = 99
end
end
k = KlassWithSecret.new
k.instance_exec(5) {|x| @secret+x }

```

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{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 infinitelength bitstring with 2's complement representation.

While Fixnum values are immediate, Bignum objects are not-assignment and parameter passing work with references to objects, not the objects themselves.

\section*{In Files}
bignum.c

\section*{Parent}

\section*{Integer}

\section*{Constants}

\section*{GMP_VERSION}

The version of loaded GMP.

\section*{Public Instance Methods}
big \% other \(\rightarrow\) Numeric
modulo(other) \(\rightarrow\) Numeric
Returns big modulo other. See Numeric\#divmod for more information.
\(\rightarrow\) big \& numeric \(\rightarrow\) integer
Performs bitwise and between big and numeric.
big * other \(\rightarrow\) Numeric
Multiplies big and other, returning the result.
big ** exponent \(\rightarrow\) 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
\begin{tabular}{lll}
123456789 & \(* *\) & 2 \\
123456789 & \(* *\) & 1.2 \\
123456789 & \(* *\) & -2
\end{tabular}
big + other \(\rightarrow\) Numeric
Adds big and other, returning the result.
big - other \(\rightarrow\) Numeric

Subtracts other from big, returning the result.
-big \(\rightarrow\) integer
Unary minus (returns an integer whose value is 0-big)
big / other \(\rightarrow\) Numeric
Performs division: the class of the resulting object depends on the class of numeric and on the magnitude of the result.

\section*{big < real \(\rightarrow\) true or false}

Returns true if the value of big is less than that of real.
big << numeric \(\rightarrow\) integer
Shifts big left numeric positions (right if numeric is negative).

\section*{big <= real \(\rightarrow\) true or false}

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

\section*{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.
\[
\text { big }==\text { obj } \rightarrow \text { true or false }
\]

Returns true only if obj has the same value as big. Contrast this with Bignum\#eql?, which requires obj to be a Bignum.
\[
68719476736==68719476736.0
\]

\section*{big \(==\) obj \(\rightarrow\) 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

```

\section*{big > real \(\rightarrow\) true or false}

Returns true if the value of big is greater than that of real.
big >= real \(\rightarrow\) true or false
Returns true if the value of big is greater than or equal to that of real.
big >> numeric \(\rightarrow\) integer
Shifts big right numeric positions (left if numeric is negative).
\(\operatorname{big}[\mathrm{n}] \rightarrow \mathbf{0 , 1}\)
Bit Reference—Returns the nth bit in the (assumed) binary representation of big, where big is the least significant bit.
```

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

```
```

print a[n]
end

```
produces:

0001011101101000001110000111100101001111000101116
big ^ numeric \(\rightarrow\) integer
Performs bitwise +exclusive or+ between big and numeric.

\section*{abs \(\rightarrow\) aBignum}
magnitude \(\rightarrow\) aBignum
Returns the absolute value of big.


\section*{bit_length \(\rightarrow\) 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 \(2 * * \mathrm{n}\) is \(\mathrm{n}+1\).) 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**10000-1).bit_length
10001
(-2**10000).bit_length
10000
(-2**10000+1).bit_length \#\#> 10000
(-2**1000-1).bit_length
1001
(-2**1000).bit_length
1000
(-2**1000+1).bit_length
1000
(2**1000-1).bit_length

```
(2**1000).bit_length
(2**1000+1).bit_length
(2**10000-1).bit_length
10000
(2**10000).bit_length
10001
(2**10000+1).bit_length
10001
```

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
```


## coerce(numeric) $\rightarrow$ 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 TypeError is raised if the numeric is not a Fixnum or Bignum type.


## div(other) $\rightarrow$ integer

Performs integer division: returns integer value.
divmod(numeric) $\rightarrow$ array
See Numeric\#divmod.

## eql?(obj) $\rightarrow$ 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.
68719476736.eql?(68719476736.0)

## even? $\rightarrow$ true or false

Returns true if big is an even number.
fdiv(numeric) $\rightarrow$ float
Returns the floating point result of dividing big by numeric.

```
-1234567890987654321.fdiv(13731)
-1234567890987654321.fdiv(13731.24)
```

hash $\rightarrow$ fixnum
Compute a hash based on the value of big.
See also Object\#hash.
inspect(p1 = v1)
Alias for: to_s
abs $\rightarrow$ aBignum
magnitude $\rightarrow$ aBignum
Returns the absolute value of big.

big \% other $\rightarrow$ Numeric
modulo(other) $\rightarrow$ Numeric
Returns big modulo other. See Numeric\#divmod for
more information.

## odd? $\rightarrow$ true or false

Returns true if big is an odd number.

## remainder(numeric) $\rightarrow$ number

Returns the remainder after dividing big by numeric.

```
-1234567890987654321. remainder(13731)
-1234567890987654321. remainder(13731.24)
```


## size $\rightarrow$ integer

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

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


## to_f $\rightarrow$ float

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

## to_s(base=10) $\rightarrow$ string

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

```
12345654321.to_s
12345654321.to_s(2)
12345654321.to_s(8)
12345654321.to_s(16)
78546939656932.to_s(36)
12345654321
1011011111111011011.
"133766736061"
"2dfdbbc31"
"ruioyrules"
```

Also aliased as: inspect
big | numeric $\rightarrow$ integer
Performs bitwise or between big and numeric.
~big $\rightarrow$ integer
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.


Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## 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. Binding 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)
eval("@secret")
```


## Binding objects have no class-specific methods.

## In Files

proc.c

## Parent

## Object

## Public Instance Methods

## eval(string [, filename [,lineno]]) $\rightarrow$ 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")
```

local_variable_defined?(symbol) $\rightarrow$ obj

Returns a true if a local variable symbol exists.

```
def foo
    a = 1
    binding.local_variable_defined?(:a)
    binding.local_variable_defined?(:b)
end
```

This method is short version of the following code.

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


local_variable_get(symbol) $\rightarrow$ obj
Returns a value of local variable symbol.

```
def foo
    a = 1
    binding.local_variable_get(:a)
    binding.local_variable_get(:b) #=> NameError
end
```

This method is short version of the following code.

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

```
local_variable_set(symbol, obj) -> obj
```

Set local variable named symbol as obj.

```
def foo
    a = 1
    b = binding
    b.local_variable_set(:a, 2) # set existing loct
    b.local_variable_set(:b, 3) # create new local.
    b.local_variable_get(:a)
    b.local_variable_get(:b)
    p a
    p b #=> NameError
end
```

This method is a similar behavior of the following code

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

if obj can be dumped in Ruby code.

## variables

```
def foo
    a = 1
    2.times do |n|
        binding.local_variables
    end
end
```

This method is short version of the following code.
binding.eval("local_variables")
receiver $\rightarrow$ object
Returns the bound receiver of the binding object.

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## class Class

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

Typically, you create a new class by using:

```
class Name
# some code describing the class behavior
end
```

When a new class is created, an object of type Class 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

```
new(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() $\rightarrow$ 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.

```
klass = Class.new do
    def initialize(*args)
        @initialized = true
    end
    def initialized?
        @initialized || false
    end
end
klass.allocate.initialized? #=> false
```

new(args, ...) $\rightarrow$ 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 $\rightarrow$ a_super_class or nil

Returns the superclass of class, or nil.

```
File.superclass
#=> IO
IO.superclass
Object.superclass
#=> Object
#=> BasicObject
class Foo; end
class Bar < Foo; end
Bar.superclass
### =00
```

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 <=> anOther.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

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

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

## 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.

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

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

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

Compares two objects based on the receiver's <=> method, returning true if it returns 0 or 1 .
between?(min, max) $\rightarrow$ true or false
Returns false if obj <=> min is less than zero or if anObject <=> max is greater than zero, true otherwise.

```
3.between?(1, 5)
6.between?(1, 5)
'cat'.between?('ant', 'dog')
'gnu'.between?('ant', 'dog')
```

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## 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, $\because:$ rect, $\because:$ polar or \#to_c method.

```
Complex(1)
Complex(2, 3)
Complex.polar(2, 3)
3.to_c
4-
```

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

```
Complex(0.3)
Complex('0.3-0.5i')
Complex('2/3+3/4i')
Complex('1@2')
0.3.to_c
    '0.3-0.5i'.to_c
'2/3+3/4i'.to_c
'1@2'.to_c

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

Complex(1, 1) / 2
Complex(1, 1) / 2.0

```

\section*{In Files}
complex.c

\section*{Parent}

\section*{Numeric}

\section*{Constants}

\section*{I}

The imaginary unit.

\section*{Public Class Methods}

\section*{polar(abs[, arg]) \(\rightarrow\) complex}

Returns a complex object which denotes the given
polar form.
```

Complex.polar(3, 0)
Complex.polar(3, Math::PI/2)
Complex.polar(3, Math::PI)
Complex.polar(3, -Math::PI/2)
rect(real[, imag]) $\rightarrow$ complex
rectangular(real[, imag]) $\rightarrow$ complex
Returns a complex object which denotes the given

## rectangular form.

Complex.rectangular(1, 2)
sect(real[, imag]) $\rightarrow$ complex
rectangular(real[, imag]) $\rightarrow$ complex
Returns a complex object which denotes the given rectangular form.

Complex.rectangular(1, 2)

## Public Instance Methods

cmp * numeric $\rightarrow$ complex
Performs multiplication.

| Complex (2, 3) | * Complex(2, 3) | $\#$ |
| :--- | :--- | :--- |
| Complex(900) | $*$ Complex(1) | $\#=(9+12 i)$ |
| Complex $(-2,9)$ | $*$ Complex $(-9,2)$ | $\#$ |
| Complex $(9,8)$ | $* 4$ | $(0-85 i)$ |
| Complex $(20,9)$ | $* 9.8$ | $\#$ |

```
cmp ** numeric }->\mathrm{ complex
```

Performs exponentiation.

| Complex('i') ** 2 |
| :--- |
| Complex(-8) ** Rational(1, 3) $\# \Rightarrow(-1+0 \mathrm{i})$ |

## cmp + numeric $\rightarrow$ complex

Performs addition.

```
Complex(2, 3) + Complex(2, 3)
Complex(900) + Complex(1)
```

```
Complex(-2, 9) + Complex(-9, 2)
Complex(9, 8) + 4
Complex(20, 9) + 9.8
```


## cmp - numeric $\rightarrow$ complex

Performs subtraction.

```
Complex(2, 3) - Complex(2, 3)
Complex(900) - Complex(1) ##> (899+0i)
Complex(-2, 9) - Complex(-9, 2)
Complex(9, 8) - 4
Complex(20, 9) - 9.8 ##> (1.0.2+9i)
```

-cmp $\rightarrow$ complex
Returns negation of the value.

```
-Complex(1, 2)
```


## cmp / numeric $\rightarrow$ complex

## quo(numeric) $\rightarrow$ complex

Performs division.


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

Returns true if cmp equals object numerically.

```
Complex(2, 3) == Complex(2, 3) ##> true
Complex(5) == 5
Complex(0) == 0.0
Complex('1/3') == 0.33
Complex('1/2') == '1/2'
#=> true
#=> true
#=> false
#=> false
```

abs $\rightarrow$ real
(8) magnitude $\rightarrow$ real

Returns the absolute part of its polar form.

```
Complex(-1).abs
Complex(3.0, -4.0).abs
```

(8) abs2 $\rightarrow$ real

Returns square of the absolute value.
Complex (-1).abs2
Complex(3.0, -4.0).abs2
arg $\rightarrow$ float
angle $\rightarrow$ float
© phase $\rightarrow$ float
Returns the angle part of its polar form.

```
Complex.polar(3, Math::PI/2).arg
```


## arg $\rightarrow$ float

angle $\rightarrow$ float
phase $\rightarrow$ float
Returns the angle part of its polar form.

conj $\rightarrow$ complex
conjugate $\rightarrow$ complex
Returns the complex conjugate.

Complex(1, 2).conjugate
conj $\rightarrow$ complex
conjugate $\rightarrow$ complex
Returns the complex conjugate.
Complex(1, 2).conjugate
denominator $\rightarrow$ integer
Returns the denominator (Icm of both denominator real and imag).

See numerator.
fdiv(numeric) $\rightarrow$ complex
Performs division as each part is a float, never returns a float.

imag $\rightarrow$ real
imaginary $\rightarrow$ real
Returns the imaginary part.

```
Complex(7).imaginary
Complex(9, -4).imaginary
```

imag $\rightarrow$ real
imaginary $\rightarrow$ real
Returns the imaginary part.

```
Complex(7).imaginary
Complex(9, -4).imaginary
```


## inspect $\rightarrow$ string

Returns the value as a string for inspection.

```
Complex(2).inspect
Complex('-8/6').inspect
Complex('1/2i').inspect
Complex(0, Float::INFINITY).inspect
Complex(Float::NAN, Float::NAN).inspect
```



## abs $\rightarrow$ real

magnitude $\rightarrow$ real
Returns the absolute part of its polar form.

```
Complex(-1).abs
Complex(3.0, -4.0).abs #=> 5.0
```


## numerator $\rightarrow$ numeric

Returns the numerator.


See denominator.
arg $\rightarrow$ float
angle $\rightarrow$ float

## phase $\rightarrow$ float

Returns the angle part of its polar form.

polar $\rightarrow$ array
Returns an array; [cmp.abs, cmp.arg].

cmp / numeric $\rightarrow$ complex

## quo(numeric) $\rightarrow$ complex

Performs division.

|  | Complex(2, 3) | / Complex(2, 3 ) |  | \#=> | $((1 / 1)+(0 / 1$ |
| :---: | :---: | :---: | :---: | :---: | :---: |
|  | Complex(900) | / Complex(1) |  | \#=> | $((900 / 1)+(0$ |
|  | Complex(-2, 9) | / Complex(-9, |  | \#=> | $(136 / 85)-(7$ |
|  | Complex(9, 8) | / 4 |  | \#=> | $(/ 9 / 4)+(2 / 1$ |
|  | Complex(20, 9) | / 9.8 |  | \#=> | (2.04081632 |
|  | 4 |  |  |  |  |

## rationalize([eps]) $\rightarrow$ rational

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

```
Complex(1.0/3, 0).rationalize
Complex(1, 0.0).rationalize
Complex(1, 2).rationalize
RanceError
```

See to_r.
real $\rightarrow$ real
Returns the real part.
Complex(7). real
real? $\rightarrow$ false
Returns false.
rect $\rightarrow$ array
rectangular $\rightarrow$ array
Returns an array; [cmp.real, cmp.imag].
Complex(1, 2).rectangular
rect $\rightarrow$ array
rectangular $\rightarrow$ array
Returns an array; [cmp.real, cmp.imag].
Complex (1, 2).rectangular

$$
\text { to_c } \rightarrow \text { self }
$$

Returns self.

```
Complex(2).to_c
Complex(-8, 6).to_c
```


## to_f $\rightarrow$ float

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

```
Complex(1, 0).to_f
Complex(1, 0.0).to_f
Complex(1, 2).to_f
```

to_i $\rightarrow$ integer

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

```
Complex(1, 0).to_i
Complex(1, 0.0).to_i
Complex(1, 2).to_i
```


## to_r $\rightarrow$ rational

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

```
Complex(1, 0).to_r
Complex(1, 0.0).to_r
Complex(1, 2).to_r
```

See rationalize.

## to_s $\rightarrow$ string

Returns the value as a string.

| Complex(2).to_s | "> "2+0i" |
| :---: | :---: |
| Complex('-8/6').to_s | "-4/3+ |
| Complex('1/2i').to_s | \#=> "0+1/2 |
| Complex(0, Float::INFINITY).to_s | $>\mathrm{O} 0+\mathrm{Inf}$ |
| Complex(Float::NAN, Float::NAN).to_s | => "NaN+N |
| 4 |  |

## conj $\rightarrow$ complex

conjugate $\rightarrow$ complex
Returns the complex conjugate.
Complex(1, 2).conjugate

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

# class <br> Complex::compatible 

## In Files

complex.c

## Parent

Object
Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## 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:

Freddie
Herbie
Ron
Max

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: 10 11 12 13 14
3: 15 16
```


## In Files

cont.c

## Parent

## Object

## Public Instance Methods

call(args, ...)
(8) 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.


Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## 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

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## 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

## Dir[ string [, string ...]] $\rightarrow$ array

Equivalent to calling Dir.glob([string, ...],0).
chdir( [ string] ) $\rightarrow 0$

## chdir( [ string] ) \{| path | block \} $\rightarrow$ 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
```

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.

```
delete( string ) }->
rmdir( string ) }->
unlink( string ) }->
```

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

```
entries( dirname ) \(\rightarrow\) array
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.

exist?(file_name) $\rightarrow$ true or false
Returns true if the named file is a directory, false otherwise.
exists?(file_name) $\rightarrow$ true or false
Deprecated method. Don't use.

```
foreach( dirname ) {| filename | block } -> nil
foreach( dirname, encoding: enc ) {| filename
|lock } -> nil
foreach( dirname ) }->\mathrm{ an_enumerator
```


## foreach( dirname, encoding: enc ) $\rightarrow$ 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.

Dir.foreach("testdir") \{|x| puts "Got \#\{x\}" \}
produces:

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

getwd $\rightarrow$ string
pwd $\rightarrow$ string
Returns the path to the current working directory of this process as a string.

```
Dir.chdir("/tmp")
Dir.getwd
Dir.pwd
```


## glob( pattern, [flags] ) $\rightarrow$ matches

glob( pattern, [flags] ) \{ |filename| block \} $\rightarrow$ nil
Expands pattern, which is an Array of patterns or a pattern String, 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 File.fnmatch 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
c*
Matches all files beginning with c

* C

Matches all files ending with 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 Regexp, 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.
$\backslash$
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:



## home() $\rightarrow$ "/home/me"

home("root") $\rightarrow$ "/root"
Returns the home directory of the current user or the named user if given.

## mkdir( string [, integer] ) $\rightarrow 0$

Makes a new directory named by string, with
permissions specified by the optional parameter anInteger. 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.

## Dir.mkdir(File.join(Dir.home, ".foo"), 0700)

## new( string ) $\rightarrow$ aDir

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.
open( string ) $\rightarrow$ aDir
open( string, encoding: enc ) $\rightarrow$ aDir
open( string ) \{| aDir | block \} $\rightarrow$ anObject open( string, encoding: enc ) \{| aDir | block \} $\rightarrow$ 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 aDir as a parameter. The directory is closed at the end of the block, and Dir: : open returns the value of the block.

## getwd $\rightarrow$ string

## pwd $\rightarrow$ string

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

```
Dir.chdir("/tmp")
Dir.getwd
Dir.pwd
```

delete( string ) $\rightarrow 0$
rmdir(string ) $\rightarrow 0$
unlink( string ) $\rightarrow 0$
Deletes the named directory. Raises a subclass of SystemCallerror if the directory isn't empty.
delete( string ) $\rightarrow 0$
rmdir( string ) $\rightarrow 0$
unlink( string ) $\rightarrow 0$
Deletes the named directory. Raises a subclass of systemCallerror if the directory isn't empty.

## Public Instance Methods

close $\rightarrow$ nil
Closes the directory stream. Any further attempts to access dir will raise an IoError.

```
d = Dir.new("testdir")
d.close
```

each \{ |filename| block \} $\rightarrow$ dir
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.

```
d = Dir.new("testdir")
d.each {|x| puts "Got #{x}" }
```

produces:

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


## fileno $\rightarrow$ integer

Returns the file descriptor used in dir.

```
d = Dir.new("..")
d.fileno
```

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

## inspect $\rightarrow$ string

Return a string describing this Dir object.
path $\rightarrow$ string or nil
to_path $\rightarrow$ string or nil
Returns the path parameter passed to dir's constructor.

```
d = Dir.new("..")
d.path
```


## pos $\rightarrow$ integer <br> tell $\rightarrow$ integer

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

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


## pos = integer $\rightarrow$ integer

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

```
d = Dir.new("testdir")
#=> #<Dir:0x401b3c40>
d.read
i = d.pos
d.read
d.pos = i
d.read
```


## read $\rightarrow$ string or nil

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

```
d = Dir.new("testdir")
d.read
d.read
d.read ##> "config.h"
```


## rewind $\rightarrow$ dir

Repositions dir to the first entry.

```
d = Dir.new("testdir")
d.read
d.rewind ##> #<Dir:0x401.b3fb0>
d.read
```


## seek( integer ) $\rightarrow$ dir

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

```
d = Dir.new("testdir")
d.read
i = d.tell
d.read
d.seek(i)
d.read
```

pos $\rightarrow$ integer
tell $\rightarrow$ integer

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

```
d = Dir.new("testdir")
d.tell
d.read
d.tell
```


## path $\rightarrow$ string or nil

to_path $\rightarrow$ string or nil
Returns the path parameter passed to dir's constructor.

```
d = Dir.new("..")
d.path
```

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## class ENV

ENV is a hash-like accessor for environment variables.

## In Files

hash.c

## Parent

## Object

## Public Class Methods

## ENV[name] $\rightarrow$ value

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

ENV[name] = value
store(name, value) $\rightarrow$ value
Sets the environment variable name to value. If the value given is nil the environment variable is deleted.
environment variable with name or nil if the name cannot be found.

## clear

Removes every environment variable.

```
delete(name) -> value
delete(name) { |name| } }->\mathrm{ 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.

## delete_if $\{$ |name, value| \} $\rightarrow$ Hash delete_if $\rightarrow$ Enumerator

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

If no block is given an enumerator is returned instead.
each $\{$ |name, value| \} $\rightarrow$ Hash
each $\rightarrow$ Enumerator
each_pair \{ |name, value| \} $\rightarrow$ Hash
each_pair $\rightarrow$ Enumerator
Yields each environment variable name and value.
If no block is given an Enumerator is returned.
each_key $\{$ |name| \} $\rightarrow$ Hash
each_key $\rightarrow$ Enumerator
Yields each environment variable name.

An Enumerator is returned if no block is given.

each \{ |name, value| \} $\rightarrow$ Hash<br>each $\rightarrow$ Enumerator<br>each_pair \{ |name, value| \} $\rightarrow$ Hash<br>each_pair $\rightarrow$ Enumerator<br>Yields each environment variable name and value.<br>If no block is given an Enumerator is returned.

each_value $\{$ |value| $\} \rightarrow$ Hash
each_value $\rightarrow$ Enumerator
Yields each environment variable value.
An Enumerator is returned if no block was given.
empty? $\rightarrow$ true or false
Returns true when there are no environment variables

## fetch(name) $\rightarrow$ value

fetch(name, default) $\rightarrow$ value
fetch(name) \{ |missing_name| ... \} $\rightarrow$ value
Retrieves the environment variable name.
If the given name does not exist and neither default nor a block a provided an IndexError 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.
key?(name) $\rightarrow$ true or false

$$
\begin{aligned}
& \text { include? }(\text { name }) \rightarrow \text { true or false } \\
& \text { has_key? }(\text { name }) \rightarrow \text { true or false } \\
& \text { member?(name) } \rightarrow \text { true or false }
\end{aligned}
$$

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

## value?(value) $\rightarrow$ true or false <br> has_value?(value) $\rightarrow$ true or false

Returns true if there is an environment variable with the given value.
key?(name) $\rightarrow$ true or false include?(name) $\rightarrow$ true or false
has_key?(name) $\rightarrow$ true or false
member?(name) $\rightarrow$ true or false
Returns true if there is an environment variable with the given name.
index(value) $\rightarrow$ key
Deprecated method that is equivalent to ::key
inspect $\rightarrow$ string
Returns the contents of the environment as a String.
invert $\rightarrow$ Hash
Returns a new hash created by using environment variable names as values and values as names.
keep_if $\{$ |name, value| $\} \rightarrow$ Hash

## keep_if $\rightarrow$ Enumerator

Deletes every environment variable where the block evaluates to false.

Returns an enumerator if no block was given.

## key(value) $\rightarrow$ name

Returns the name of the environment variable with value. If the value is not found nil is returned.
key?(name) $\rightarrow$ true or false include?(name) $\rightarrow$ true or false has_key? (name) $\rightarrow$ true or false member?(name) $\rightarrow$ true or false
Returns true if there is an environment variable with the given name.

## keys $\rightarrow$ Array

Returns every environment variable name in an Array
length
size
Returns the number of environment variables.
key?(name) $\rightarrow$ true or false include?(name) $\rightarrow$ true or false has_key?(name) $\rightarrow$ true or false member?(name) $\rightarrow$ true or false
Returns true if there is an environment variable with the given name.

## rassoc(value)

Returns an Array 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| \} $\rightarrow$ Hash reject $\rightarrow$ Enumerator

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

## reject! \{ |name, value| \} $\rightarrow$ ENV or nil reject! $\rightarrow$ Enumerator

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

Returns an Enumerator if no block was given.

## replace(hash) $\rightarrow$ env

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

## select \{ |name, value| \} $\rightarrow$ Hash select $\rightarrow$ 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| \} $\rightarrow$ ENV or nil <br> select! $\rightarrow$ Enumerator

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

## shift $\rightarrow$ Array or nil

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

## length

## size

Returns the number of environment variables.

## ENV[name] = value

store(name, value) $\rightarrow$ value
Sets the environment variable name to value. If the value given is nil the environment variable is deleted.

## to_a $\rightarrow$ Array

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

## ENV.to_a

to_hash $\rightarrow$ hash
to_h $\rightarrow$ hash
Creates a hash with a copy of the environment variables.
to_hash $\rightarrow$ hash
to_h $\rightarrow$ hash
Creates a hash with a copy of the environment variables.
to_s $\rightarrow$ "ENV"
Returns "ENV"
update(hash) $\rightarrow$ Hash
update(hash) \{ |name, old_value, new_value| \} $\rightarrow$ 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.

## value?(value) $\rightarrow$ true or false

has_value?(value) $\rightarrow$ true or false
Returns true if there is an environment variable with the given value.
values $\rightarrow$ Array
Returns every environment variable value as an Array
values_at(name, ...) $\rightarrow$ Array
Returns an array containing the environment variable values associated with the given names. See also : : select.

Generated by RDoc 3.12.2. Generated with the Darkfish Rdoc Generator 3.

## class EOFError

Raised by some IO operations when reaching the end of file. Many IO 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 RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## class Encoding

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

Encoding: :ISO_8859_1.name

Encoding : : ISO_8859_1.names

Ruby methods dealing with encodings return or accept Encoding instances as arguments (when a method accepts an Encoding instance as an argument, it can be passed an Encoding 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 ASCII8BIT characters with other ASCII compatible characters.

## Changing an encoding

The associated Encoding of a String can be changed in two different ways.

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

```
string
string.encoding
string.force_encoding(Encoding::UTF_8)
```

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 String\#encode for the various forms of transcoding, and the Encoding::Converter class for additional control over the transcoding process.

```
string
string.encoding
string = string.encode!(Encoding::ISO_8859_1)
string.encoding
```



## Script encoding

All Ruby script code has an associated Encoding which any String 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:


The $\qquad$ ENCODING $\qquad$ keyword returns the script encoding of the file which the keyword is written:

ENCODING
ruby - k 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 USASCII strings or regular expressions.

## Locale encoding

The default encoding of the environment. Usually derived from locale.
see Encoding.locale_charmap, ::find('locale')

## 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 :::default_external=, 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 IO object is not the default external encoding, you can reset its external encoding with IO\#set_encoding or set it at IO object
creation (see IO.new 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 IO object it is transcoded from the internal encoding to the external encoding of the IO object.

The internal encoding of an IO object can be set with IO\#set_encoding or at IO object creation (see IO.new 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 . ::default_internal returns the current internal encoding.


The default internal encoding may also be set through :: default_internal=, 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:ISO-8859-1") do |io
    io.write(string)
end
puts "raw text:"
p File.binread("transcoded.txt")
puts
open("transcoded.txt", "r:ISO-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" $\rightarrow$ "orig2", ...\}
Returns the hash of available encoding alias and original encoding name.


## 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

## default_external $\rightarrow$ enc

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 ::default_external for information on how the default external encoding is used.

## default_internal $\rightarrow$ 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
- Values from ENV
- Values in ARGV including \$PROGRAM_NAME

Additionally String\#encode and String\#encode! 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.

## default_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.

## find(string) $\rightarrow$ enc

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

## Encoding.find("US-ASCII")

$\cdot$
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 ArgumentError 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.

```
list \(\rightarrow\) [enc1, enc2, ...]
```

Returns the list of loaded encodings.


## name_list $\rightarrow$ ["enc1", "enc2", ...]

Returns the list of available encoding names.

```
Encoding.name_list
    "IS0-8859-1", "Shift_JIS", "EUC-JP",
    "Windows-31J",
    "BINARY", "CP932", "eucJP"]
```


## Public Instance Methods

## ascii_compatible? $\rightarrow$ true or false

Returns whether ASCII-compatible or not.
Encoding: :UTF_8.ascii_compatible?
true
Encoding: :UTF_16BE.ascii_compatible? false

## dummy? $\rightarrow$ 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.
Encoding: :ISO_2022_JP.dummy?
Encoding: :UTF_8.dummy?
inspect $\rightarrow$ string
Returns a string which represents the encoding for programmers.

```
Encoding::UTF_8.inspect
Encoding::ISO_2022_JP.inspect
```

name $\rightarrow$ string
to_s $\rightarrow$ string
Returns the name of the encoding.

```
Encoding::UTF_8.name
```

sinames $\rightarrow$ array
Returns the list of name and aliases of the encoding.

replicate(name) $\rightarrow$ 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.
name $\rightarrow$ string
to_s $\rightarrow$ string
Returns the name of the encoding.

```
Encoding: :UTF_8.name
```

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

# class <br> Encoding::CompatibilityErro 

Raised by Encoding and String methods when the source encoding is incompatible with the target encoding.

## In Files

encoding.c

## Parent

EncodingError
Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## class Encoding::Converter

## In Files

$\doteq$ 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

## Encoding::Converter.asciicompat_encoding(str $\rightarrow$ encoding or nil <br> Encoding::Converter.asciicompat_encoding(en $\rightarrow$ encoding or nil

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.asciicompat_encoding("ISO-202
Encoding : :Converter.asciicompat_encoding("UTF-16:
Encoding : Converter.asciicompat_encoding("UTF-8")
$\xrightarrow{\square}$

## Encoding::Converter.new(source_encoding, destination_encoding) <br> Encoding::Converter.new(source_encoding, destination_encoding, opt) <br> Encoding::Converter.new(convpath)

## possible options elements:

```
hash form:
    :invalid => nil
    :invalid => :replace
    :undef => nil
    :undef => :replace
    :replace => string
    :newline => :universal # decorator for con
    :newline => :crlf
    :newline => :cr
    :universal_newline => true # decorator for con
    :crlf_newline => true # decorator for con
    :cr_newline => true # decorator for con
    :xml => :text
    :xml => :attr
integer form:
    Encoding::Converter::INVALID_REPLACE
    Encoding::Converter::UNDEF_REPLACE
    Encoding::Converter::UNDEF_HEX_CHARREF
    Encoding::Converter::UNIVERSAL_NEWLINE_DECORATd
    Encoding::Converter::CRLF_NEWLINE_DECORATOR
    Encoding::Converter::CR_NEWLINE_DECORATOR
    Encoding::Converter::XML_TEXT_DECORATOR
    Encoding::Converter::XML_ATTR_CONTENT_DECORATOF
    Encoding: : Converter : : XML_ATTR_QUOTE_DECORATOR
```

Source_encoding and \#destination_encoding should be a string or Encoding 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 \#source_encoding is not defined in destination_encoding. This is a default behavior.

## :undef => :replace

Replace undefined character in \#destination_encoding 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.

- '\&' -> '\&'
- '<' -> '\<'
- '>' -> '\>'
- undefined characters in \#destination_encoding -> 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.

- '\&' -> '\&'
- '<' -> '\<'
- '>' -> '\>'
- '"' -> '\"'
- undefined characters in \#destination_encoding ->
hexadecimal CharRef such as \&xHH;


## Examples:



## Encoding::Converter.search_convpath(source_ destination_encoding) $\rightarrow$ ary <br> Encoding::Converter.search_convpath(source destination_encoding, opt) $\rightarrow$ ary

Returns a conversion path.

```
p Encoding::Converter.search_convpath("ISO-8859-
#=> [[#<Encoding:IS0-8859-1>, #<Encoding:UTF-8>]
    [#<Encoding:UTF-8>, #<Encoding:EUC-JP>]]
p Encoding::Converter.search_convpath("ISO-8859-1
or
p Encoding::Converter.search_convpath("ISO-8859-1
#=> [[#<Encoding:IS0-8859-1>, #<Encoding:UTF-8>]
    [#<Encoding:UTF-8>, #<Encoding:EUC-JP>],
    "universal_newline"]
p Encoding::Converter.search_convpath("ISO-8859-1
```



## Public Instance Methods

## ec == other $\rightarrow$ true or false

## convert(source_string) $\rightarrow$ 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.

```
ec = Encoding::Converter.new("utf-8", "euc-jp")
puts ec.convert("\u3042").dump
puts ec.finish.dump
ec = Encoding::Converter.new("euc-jp", "utf-8")
puts ec.convert("\xA4").dump
puts ec.convert("\XA2").dump
puts ec.finish.dump
ec = Encoding::Converter.new("utf-8", "iso-2022-
puts ec.convert("\xE3").dump
puts ec.convert("\x81").dump
puts ec.convert("\x82").dump
puts ec.finish.dump
```

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.

## 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:ISO-8859-1), \# [Encoding:UTF-8](Encoding:UTF-8)] means a converter from ISO-8859-1 to UTF-8. "crlf_newline" means newline converter from LF to CRLF.
destination_encoding $\rightarrow$ encoding
Returns the destination encoding as an Encoding object.

## finish $\rightarrow$ string

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

insert_output(string) $\rightarrow$ 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.


## inspect $\rightarrow$ string

Returns a printable version of ec
$\square$

## last_error $\rightarrow$ 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.
ec = Encoding: :Converter.new("utf-8", "iso-8859-1
p ec.primitive_convert(src="\xf1abcd", dst="")
p ec.last_error $\quad \# \quad$ "<Encoding:InvalidByte
p ec.primitive_convert(src, dst, nil, 1)
p ec.last_error
primitive_convert(source_buffer, destination_buffer) $\rightarrow$ symbol primitive_convert(source_buffer, destination_buffer, destination_byteoffset)
$\rightarrow$ symbol
primitive_convert(source_buffer, destination_buffer, destination_byteoffset, destination_bytesize) $\rightarrow$ symbol primitive_convert(source_buffer, destination_buffer, destination_byteoffset, destination_bytesize, opt) $\rightarrow$ symbol possible opt elements:

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]
ec = Encoding::Converter.new("UTF-8", "UTF-16BE"
ret = ec.primitive_convert(src="pi", dst="", nil,
p [ret, src, dst]
ret = ec.primitive_convert(src, dst="", nil, 1)
```

```
p [ret, src, dst]
ret = ec.primitive_convert(src, dst="", nil, 1)
p [ret, src, dst] ##> [:destination buffer_ful].
ret = ec.primitive_convert(src, dst="", nil, 1)
p [ret, src, dst]
```


## primitive_errinfo $\rightarrow$ array

\#primitive_errinfo returns important information regarding the last error as a 5-element array:

[result, enc1, enc2, error_bytes, readagain_bytes

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:

\# Vxff is invalid as EUC-JP.
ec = Encoding: :Converter.new("EUC-JP", "Shift_JIS
ec.primitive_convert(src="\xff", dst="", nil, 10)
p ec.primitive_errinfo
\#=> [:invalid byte sequence, "EUC-JP", "UTF-8",
\# HIRAGANA LETTER A (Vxa4\xa2 in EUC-JP) is not
\# Since this error is occur in UTF-8 to IS0-8859,
\# error bytes is HIRAGANA LETTERA in UTF-8 (VxE
ec = Encoding: :Converter. new("EUC-JP", "ISO-8859.

```
ec.primitive_convert(src="\xa4\xa2", dst="", nil,
p ec.primitive_errinfo
    partial character is invalid
ec = Encoding::Converter.new("EUC-JP", "ISO-8859
ec.primitive_convert(src="\xa4", dst="", nil, 10
p ec.primitive_errinfo
    Encoding:: Converter::PARTIAL_INPUT prevents in
    partial characters
ec = Encoding::Converter.new("EUC-JP", "ISO-8859
ec.primitive_convert(src="\xa4", dst="", nil, 10,
p ec.primitive_errinfo
#=> [:source_buffer_empty, nil, nil, nil, nil]
    \xd8\x00\x00@ is invalid as UTF-16BE because
    no low surrogate after high surrogate (\xd8\x0
    It is detected by 3rd byte (\00) which is part
    So the high surrogate (\xd8\x00) is discarded
    the 3rd byte is read again later
    Since the bvte is buffered in ec
ec = Encoding::Converter.new("UTF-16BE", "UTF-8"
ec.primitive_convert(src="\xd8\x00\x00@", dst=""
p ec.primitive_errinfo
p src
    Similar to UTF-16BE, \x00\xd8@\x00 is invalid
    The problem is detected by 4th byte.
ec = Encoding::Converter.new("UTF-16LE", "UTF-8"
ec.primitive_convert(src="\x00\xd8@\x00", dst="",
p ec.primitive_errinfo
p src
```


## putback(p1 = v1)

call-seq

```
ec.putback
-> string
ec.putback(max_numbytes)
-> 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.

```
ec = Encoding::Converter.new("utf-16le", "iso-885
src = "\x00\xd8\x61\x00"
dst = ""
p ec.primitive_convert(src, dst)
p ec.primitive_errinfo
p ec.putback
p ec.putback


\section*{replacement \(\rightarrow\) string}

\section*{Returns the replacement string.}
```

ec = Encoding::Converter.new("euc-jp", "us-ascii
p ec.replacement
ec = Encoding::Converter.new("euc-jp", "utf-8")
p ec.replacement

## replacement = string

Sets the replacement string.

source_encoding $\rightarrow$ encoding

Returns the source encoding as an Encoding object.

Generated by RDoc 3.12.2. Generated with the Darkfish Rdoc Generator 3.

# class <br> 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 RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

# class <br> Encoding::InvalidByteSeque 

Raised by Encoding and String 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

destination_encoding $\rightarrow$ string
Returns the destination encoding as an encoding object.
destination_encoding_name $\rightarrow$ string
Returns the destination encoding name as a string.
error_bytes $\rightarrow$ string
Returns the discarded bytes when
Encoding::InvalidByteSequenceError occurs.

```
ec = Encoding::Converter.new("EUC-JP", "ISO-8859
begin
    ec.convert("abc\xA1\xFFdef")
rescue Encoding::InvalidByteSequenceError
    p $! #=> #<Encoding::InvalidByteSequenceE
    puts $!.error_bytes.dump
    puts $!.readagain_bytes.dump
end
|
```


## incomplete_input? $\rightarrow$ true or false

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

```
ec = Encoding::Converter.new("EUC-JP", "ISO-8859
begin
    ec.convert("abc\xA1z")
rescue Encoding::InvalidByteSequenceError
    p $! #=> #<Encoding::InvalidByteSequenceE
    p $!.incomplete_input? ##> false
end
begin
    ec.convert("abc\xA1")
    ec.finish
rescue Encoding::InvalidByteSequenceError
    p $! #=> #<Encoding::InvalidByteSequenceE
    p $!.incomplete_input? #=> true
end
|
```


## readagain_bytes $\rightarrow$ string

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

## source_encoding $\rightarrow$ 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 $\rightarrow$ string
Returns the source encoding name as a string.

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## class <br> Encoding::UndefinedConver

Raised by Encoding and String methods when a transcoding operation fails.

## In Files

encoding.c

## Parent

rb_eEncodingError

## Public Instance Methods

destination_encoding $\rightarrow$ string
Returns the destination encoding as an encoding object.
destination_encoding_name $\rightarrow$ string
Returns the destination encoding name as a string.
error_char $\rightarrow$ string
Returns the one-character string which cause
Encoding::UndefinedConversionError.
ec = Encoding: : Converter.new("ISO-8859-1", "EUCbegin

```
    ec.convert("\xa0")
rescue Encoding::UndefinedConversionError
    puts $!.error_char.dump
    p $!.error_char.encoding
end
```


## source_encoding $\rightarrow$ 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 $\rightarrow$ string

Returns the source encoding name as a string.

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

# class EncodingError 

EncodingError is the base class for encoding errors.

## In Files

$\equiv$ error.c

## Parent

## StandardError

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 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

## all? [\{ |obj| block \} ] $\rightarrow$ 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 all? to return true when none of the collection members are false or nil.


## any? [\{ |obj| block \}] $\rightarrow$ true or false

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 any? to return true if at least one of the collection members is not false or nil.

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

chunk $\{|e l t| \ldots\} \rightarrow$ an_enumerator chunk(initial_state) \{ |elt, state| ... \} $\rightarrow$ 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:

```
File.foreach("README").chunk { |line|
    /\A\s*\z/ !~ line || nil
}.each { |_, lines|
    pp lines
}
```

:_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:

```
pattern = /http/
open(filename) { |f|
    f.chunk { |line| line =~ pattern ? :_alone : t|
        pp lines
    }
}
```

collect $\{$ |obj| block $\} \rightarrow$ array
map \{ |obj| block \} $\rightarrow$ array
collect $\rightarrow$ an_enumerator
map $\rightarrow$ 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.

flat_map \{ |obj| block \} $\rightarrow$ array
collect_concat \{ |obj| block \} $\rightarrow$ array
flat_map $\rightarrow$ an_enumerator
collect_concat $\rightarrow$ 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.

```
[1, 2, 3, 4].flat_map { |e| [e, -e] }
[[1, 2], [3, 4]].flat_map { |e| e + [100] }
```

count $\rightarrow$ intcount(item) $\rightarrow$ int
count \{ |obj| block \} $\rightarrow$ 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.

```
ary = [1, 2, 4, 2]
ary.count
ary.count(2)
ary.count{ |x| x%2==0 }
```

cycle(n=nil) \{ |obj| block \} $\rightarrow$ nil
cycle(n=nil) $\rightarrow$ an_enumerator
Calls block for each element of enum repeatedly $n$ times or forever if none or nil is given. If a nonpositive number is given or the collection is empty,
does nothing. Returns nil if the loop has finished without getting interrupted.
\#cycle 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.

```
a = ["a", "b", "c"]
a.cycle { |x| puts x }
a.cycle(2) { |x| puts x }
```

detect(ifnone $=$ nil) $\{$ |obj| block $\} \rightarrow$ obj or nil find(ifnone $=$ nil) $\{$ |obj| block $\} \rightarrow$ obj or nil detect(ifnone = nil) $\rightarrow$ an_enumerator 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.

drop(n) $\rightarrow$ array
Drops first n elements from enum, and returns rest elements in an array.

```
a = [1, 2, 3, 4, 5, 0]
a.drop(3)
```


## drop_while \{ |arr| block \} $\rightarrow$ array drop_while $\rightarrow$ 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 }
```

each_cons(n) \{ ... $\} \rightarrow$ nil
each_cons(n) $\rightarrow$ an_enumerator
Iterates the given block for each array of consecutive <n> elements. If no block is given, returns an enumerator.
e.g.:

```
(1..10).each_cons(3) { |a| p a }
[1, 2, 3]
[2, 3, 4]
[3, 4, 5]
[4, 5, 6]
[5, 6, 7]
[6, 7, 8]
[7, 8, 9]
[8, 9, 10]
```

each_entry \{ |obj| block \} $\rightarrow$ enum
each_entry $\rightarrow$ 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
```

each_slice(n) $\{\ldots\} \rightarrow$ nil
each_slice(n) $\rightarrow$ an_enumerator

Iterates the given block for each slice of <n> elements. If no block is given, returns an enumerator.

```
(1..10).each_slice(3) { |a| p a }
# outputs below
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
[10]
```

each_with_index(*args) \{ |obj, i| block \} $\rightarrow$ 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.

```
hash = Hash.new
%w(cat dog wombat).each_with_index { |item, inde>
    hash[item] = index
}
hash
dog"=>1, "wombat
```

each_with_object(obj) \{ |(*args), memo_obj| ... \} $\rightarrow$ obj
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.

to_a(*args) $\rightarrow$ array
entries(*args) $\rightarrow$ array
Returns an array containing the items in enum.

detect(ifnone = nil) \{ |obj| block \} $\rightarrow$ obj or nil
find(ifnone = nil) \{ |obj| block \} $\rightarrow$ obj or nil
detect(ifnone $=$ nil) $\rightarrow$ an_enumerator
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.


## find_all \{ |obj| block \} $\rightarrow$ array

select \{ |obj| block \} $\rightarrow$ array
find_all $\rightarrow$ an_enumerator
select $\rightarrow$ 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) $\rightarrow$ int or nil
find_index \{ |obj| block \} $\rightarrow$ int or nil
find_index $\rightarrow$ 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.

first $\rightarrow$ obj or nil
first(n) $\rightarrow$ 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.

```
%w[foo bar baz].first
%w[foo bar baz].first(2)
%w[foo bar baz].first(10)
[].first
```

flat_map \{ |obj| block \} $\rightarrow$ array collect_concat \{ |obj| block \} $\rightarrow$ array
flat_map $\rightarrow$ an_enumerator collect_concat $\rightarrow$ 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.

```
[1, 2, 3, 4].flat_map { |e| [e, -e] }
[[1, 2], [3, 4]].flat_map { |e| e + [100] }
```

grep(pattern) $\rightarrow$ array grep(pattern) \{ |obj| block \} $\rightarrow$ 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.

group_by \{ |obj| block \} $\rightarrow$ a_hash group_by $\rightarrow$ 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.

include?(obj) $\rightarrow$ true or false
member?(obj) $\rightarrow$ true or false
Returns true if any member of enum equals obj.
Equality is tested using ==.

> IO.constants.include? :SEEK_SET
> IO.constants.include? :SEEK_NO_FURTHER
inject(initial, sym) $\rightarrow$ obj
inject(sym) $\rightarrow$ obj
inject(initial) \{ |memo, obj| block \} $\rightarrow$ obj
inject \{ |memo, obj| block \} $\rightarrow$ obj
reduce(initial, sym) $\rightarrow$ obj
reduce(sym) $\rightarrow$ obj
8)
reduce(initial) \{ |memo, obj| block \} $\rightarrow$ obj 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 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 $\rightarrow$ lazy_enumerator

Returns a lazy enumerator, whose methods map/collect, flat_map/collect_concat, select/find_all, reject, grep, zip, take, \#take_while, drop, and \#drop_while 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:

collect \{ |obj| block \} $\rightarrow$ array
map \{ |obj| block \} $\rightarrow$ array
collect $\rightarrow$ an_enumerator
map $\rightarrow$ 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.

max $\rightarrow$ obj
$\max \{|a, b|$ block $\} \rightarrow$ obj
$\max (\mathrm{n}) \rightarrow$ obj
$\max (\mathrm{n})\{|\mathrm{a}, \mathrm{b}|$ block $\} \rightarrow$ 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 $<=>b$.

```
a = %w(albatross dog horse)
a.max
a.max { |a, b| a.length <=> b.length }

If the n argument is given, maximum n elements are returned as an array.

max_by \{|obj| block \} \(\rightarrow\) obj
max_by \(\rightarrow\) an_enumerator
max_by(n) \{|obj| block \} \(\rightarrow\) obj
max_by(n) \(\rightarrow\) 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 }

```

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 }

```
enum.max_by(n) can be used to implement weighted random sampling. Following example implements and use Enumerable\#wsample.


\title{
include?(obj) \(\rightarrow\) true or false \\ member?(obj) \(\rightarrow\) true or false
}

Returns true if any member of enum equals obj.
Equality is tested using ==.
IO.constants.include? : SEEK_SET
IO.constants.include? :SEEK_NO_FURTHER
min \(\rightarrow\) obj
\(\min \{|\mathrm{a}, \mathrm{b}|\) block \(\} \rightarrow\) obj
\(\min (n) \rightarrow\) array
\(\min (\mathrm{n})\{|\mathrm{a}, \mathrm{b}|\) block \(\} \rightarrow\) 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.


If the n argument is given, minimum n elements are returned as an array.

min_by \{|obj| block \} \(\rightarrow\) obj
min_by \(\rightarrow\) an_enumerator
min_by(n) \{|obj| block \} \(\rightarrow\) array
min_by(n) \(\rightarrow\) 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.
```

a = %w(albatross dog horse)
a.min_by { |x| x.length }

```

If the n argument is given, minimum n elements are returned as an array.
```

a = %w[albatross dog horse]
p a.min_by(2) {|x| x.length }

```
minmax \(\rightarrow\) [min, max]
\(\operatorname{minmax}\{|a, b|\) block \(\} \rightarrow[\min , \max ]\)

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 \(<=>b\).
```

a = %w(albatross dog horse)
a.minmax
a.minmax { |a, b| a.length <=> b.length }

```
minmax_by \{ |obj| block \} \(\rightarrow\) [min, max]
minmax_by \(\rightarrow\) 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.
```

a = %w(albatross dog horse)

```


\section*{none? [\{ |obj| block \}] \(\rightarrow\) 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.


\section*{one? [\{ |obj| block \}] \(\rightarrow\) 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, false_array ]

\section*{partition \(\rightarrow\) 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.

inject(initial, sym) \(\rightarrow\) obj
inject(sym) \(\rightarrow\) obj
inject(initial) \{ |memo, obj| block \} \(\rightarrow\) obj
inject \{ |memo, obj| block \} \(\rightarrow\) obj
reduce(initial, sym) \(\rightarrow\) obj
reduce(sym) \(\rightarrow\) obj
reduce(initial) \{ |memo, obj| block \} \(\rightarrow\) obj
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 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.
```

(5..10) .reduce( : + )

```
(5..10).inject \(\{\mid\) sum, \(n \mid\) sum \(+n\}\)
(5..10).reduce(1, :*)
\(\#\) Same using a biock
(5..10).inject(1) \{ |product, \(\mathrm{n} \mid\) product * n \}
\# find the longest word
longest \(=\) \%w cat sheep bear \}.inject do |memo,
memo.length \(>\) word.length ? memo : word
end
longest

\section*{reject \{ |obj| block \} \(\rightarrow\) array}
reject \(\rightarrow\) an_enumerator
Returns an array for all elements of enum for which the given block returns false.

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

(1..10).reject { |i| i % 3 == 0 }
[1, 2, 3, 4, 5].reject { |num| num.even? }

```

See also \#find_all.

\section*{reverse_each(*args) \{ |item| block \} \(\rightarrow\) enum reverse_each(*args) \(\rightarrow\) an_enumerator}

Builds a temporary array and traverses that array in reverse order.

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

    (1..3).reverse_each { |v| p v }
    produces:
3
2
1

```
```

find_all { |obj| block } -> array
select { |obj| block } -> array
find_all }->\mathrm{ an_enumerator
select }->\mathrm{ 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.
```

(1..10).find_all { |i| i % 3 == 0 }

```
\([1,2,3,4,5]\). select \(\{\) |num| num.even? \}

See also \#reject.
slice_after(pattern) \(\rightarrow\) an_enumerator slice_after \{ |elt| bool \} \(\rightarrow\) 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:


Other methods of the Enumerator 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) \(\rightarrow\) an_enumerator slice_before \(\{\) |elt| bool \} \(\rightarrow\) an_enumerator slice_before(initial_state) \{ |elt, state| bool \} (7) \(\rightarrow\) 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 Enumerator class and Enumerable module, such as map, etc., are also usable.

For example, iteration over ChangeLog entries can
be implemented as follows:
```

    -twerate over ChangeLog entries
    open("ChangeLog") { |f|
f.slice_before(/\A\S/).each { |e| pp e }
}

## same as above. block is used instead of patte

open("ChangeLog") { |f|
f.slice_before { |line| /\A\S/ === line }.each
}

```
"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:
```

a = [0, 2, 3, 4, 6, 7, 9]
prev = a[0]
p a.slice_before { |e|
prev, prev2 = e, prev
prev2 + 1 != e
}.map { |es|
es.length <= 2 ? es.join(",") : "\#{es.first}-\#
}.join(",")
\#=> "0,2-4,6,7,9

```

However local variables should be used carefully if the result enumerator is enumerated twice or more. The local variables should be initialized for each

\section*{enumeration. Enumerator.new can be used to do it.}
```

    Word wrapping. This assumes all characters hav
    def wordwrap(words, maxwidth)
Enumerator.new {|y|
\# cols is initialized in Enumerator.new.
cols = 0
words.slice_before { |w|
cols += 1 if cols != 0
cols += w.length
if maxwidth < cols
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(" ") } \# first enul
puts "-"*10
enum.each { |ws| puts ws.join(" ") } \# second ent
puts "-"*10

```
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
    }
    }

# split mails in mbox (slice before Unix From lis

open("mbox") { |f|
f.slice_before(emp: true) { |line, h|
prevemp = h[:emp]
h[:emp] = line == "\n"
prevemp \&\& line.start_with?("From ")
}.each { |mail|
mail.pop if mail.last == "\n"
pp mail
}
}

```
slice_when \(\left\{\left|e l t \_b e f o r e, ~ e l t \_a f t e r\right| ~ b o o l ~\right\} ~ \rightarrow ~\) 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 Enumerator class and
Enumerable 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:
\[
\begin{aligned}
& a=[0,9,2,2,3,2,7,5,9,5] \\
& p \text { a.slice_when }\{|i, j| i>j\} . t o \_a
\end{aligned}
\]

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 \#chunk to ignore empty lines.)


\section*{sort \(\rightarrow\) array}
sort \{ |a, b| block \} \(\rightarrow\) 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 \} \(\rightarrow\) array
sort_by \(\rightarrow\) 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.


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.
```

files = Dir["*"]
sorted = files.sort { |a, b| File.new(a).mtime
sorted \#=> ["mon", "tues", "wed", "thurs"]

```


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.
```

files = Dir["*"]
sorted = files.sort { |a, b|
test(M, a) <=> test(M, b)
}
sorted \#\#> ["mon", "tures|

```

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.
```

sorted = Dir["*"].collect { |f|
[test(M, f), f]
}.sort.collect { |f| f[1] }
sorted

```

This is exactly what sort_by does internally.
```

sorted = Dir["*"].sort_by { |f| test(M, f) }
sorted

```

\section*{take(n) \(\rightarrow\) array}

Returns first n elements from enum.
```

a = [1, 2, 3, 4, 5, 0]
a.take(3)
a.take(30)

```

\section*{take_while \{ |arr| block \} \(\rightarrow\) array}
take_while \(\rightarrow\) an_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 enumerator is returned instead.
```

a = [1, 2, 3, 4, 5, 0]
a.take_while { |i| i < 3 }

```
to_a(*args) \(\rightarrow\) array

\section*{entries(*args) \(\rightarrow\) array}

Returns an array containing the items in enum.

> (1. . 7).to_a
> \{ 'a'=>1, 'b'=>2, 'c'=>3 \}.to_a
> require 'prime'
> Prime.entries 10

\section*{to_h(*args) \(\rightarrow\) hash}

Returns the result of interpreting enum as a list of [key, value] pairs.
\%[hello world].each_with_index.to_h
sip(arg, ...) \(\rightarrow\) an_array_of_array zip(arg, ...) \{ |arr| block \} \(\rightarrow\) 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 RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{class Enumerator}

A class which allows both internal and external iteration.

An Enumerator 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,
puts "\#{obj}: \#{item}"
end
foo: one
foo: two
foo: three
enum_with_ooj = enumerator.each_with_oobject("
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:
```

puts %w[foo bar baz].map.with_index { |w, i|
|

```

An Enumerator can also be used as an external iterator. For example, \#next returns the next value of the iterator or raises Stoplteration if the Enumerator is at the end.
```

e = [1,2,3].each
puts e.next
puts e.next
puts e.next
puts e.next
raises Stopiteration

```
4.

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

```


\section*{In Files}
enumerator.c

\section*{Parent}

\section*{Object}

\section*{Included Modules}

Enumerable

\section*{Public Class Methods}

\author{
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 \(+\ll+\) ):


The optional parameter can be used to specify how to calculate the size in a lazy fashion (see \#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.


\section*{Public Instance Methods}each \(\{\mid\) elm| block \(\} \rightarrow\) obj
each \(\rightarrow\) enum
each(*appending_args) \{ |elm| block \} \(\rightarrow\) obj
each(*appending_args) \(\rightarrow\) an_enumerator
Iterates over the block according to how this
Enumerator was constructed. If no block and no arguments are given, returns self.

\section*{Examples}

each_with_index \(\{\mid(* a r g s)\), idx| ... \}
each_with_index
Same as \#with_index, i.e. there is no starting offset.

If no block is given, a new Enumerator is returned that includes the index.

\section*{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.

\section*{Example}
```

to_three = Enumerator.new do |y|
3.times do |x|
y << x
end
end
to_three_with_string = to_three.with_object("foo'
to_three_with_string.each do |x,string|
puts "\#{string}: \#{x}"
end

```
feed obj \(\rightarrow\) 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.
```


# Following example shows that "next" returns the

# values passed to "feed" are collected as an ar

# obtained by StopIteration\#result.

e = [1,2,3].map
p e.next
e.feed "a"
p e.next
e.feed "b"
p e.next
e.feed "c"
begin
e.next
rescue StopIteration
p \$!.result \#=> ["a", "b", "c"]
end
o = Object.new
def o.each
x = yield \# (2) blocks
p x \# (5) => "foom
x = yield \# (6) blocks
p x \# (8) => ni.l
x = yield \# (9) blocks
p x \# not reached w/o another e,
end
e = o.to_enum
e.next
e.feed "foo"
\# (3)
e.next \# (4)
e.next \# (7)
\# (10)
4.

```

\section*{inspect \(\rightarrow\) string}

Creates a printable version of \(e\).

\section*{next \(\rightarrow\) object}

Returns the next object in the enumerator, and move the internal position forward. When the position reached at the end, Stoplteration is raised.

\section*{Example}
```

a = [1,2,3]
e = a.to_enum
p e.next
p e.next \#\#> 2
p e.next \#\#> 3
p e.next \#raises StopIteration

```

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.

\section*{next_values \(\rightarrow\) array}

Returns the next object as an array in the enumerator, and move the internal position forward. When the position reached at the end, Stoplteration is raised.

This method can be used to distinguish yield and yield nil.

\section*{Example}
```

o = Object.new
def o.each
yield
yield 1
yield 1, 2
yield nil
yield [1, 2]
end
e = o.to_enum
p e.next_values
p e.next_values
p e.next_values
p e.next_values
p e.next_values
e = o.to_enum
p e.next

```
```

p e.next
p e.next
p e.next
p e.next

```
\#\# yield args
next_values
yield [] nil.
yield 1
yield 1, 2
yield nil.
yield [1, 2]

\section*{[1]}

[nil] nil
[1, 2 il
Note that next_values does not affect other nonexternal enumeration methods unless underlying iteration method itself has side-effect, e.g. IO\#each_line.

\section*{peek \(\rightarrow\) object}

Returns the next object in the enumerator, but doesn't move the internal position forward. If the position is already at the end, Stoplteration is raised.

\section*{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 \#\#> 3
p e.peek \#raises StopIteration

```

\section*{peek_values \(\rightarrow\) array}

Returns the next object as an array, similar to \#next_values, but doesn't move the internal position forward. If the position is already at the end, Stoplteration is raised.

\section*{Example}
```

o = Object.new
def o.each
yield
yield 1
yield 1, 2
end
e = o.to_enum
p e.peek_values
e.next
p e.peek_values
p e.peek_values
e.next
p e.peek_values
e.next
p e.peek_values \# raises StopIteration

```

\section*{rewind \(\rightarrow\) e}

Rewinds the enumeration sequence to the beginning.
If the enclosed object responds to a "rewind" method, it is called.
size \(\rightarrow\) 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.

\section*{Example}
```

to_three = Enumerator.new do |y|
3.times do |x|
y << x
end
end
to_three_with_string = to_three.with_object("foo'
to_three_with_string.each do |x,string|
puts "\#{string}: \#{x}"
end

```

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\title{
class \\ Enumerator::Generator
}

Lazy

\section*{In Files}
enumerator.c

\section*{Parent}

\section*{Object}

\section*{Included Modules}
- Enumerable

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{class Enumerator::Lazy}

Lazy

\section*{In Files}
enumerator.c

\section*{Parent}

\section*{Enumerator}

\section*{Public Class Methods}
new(obj, size=nil) \{ |yielder, *values| ... \}
Creates a new Lazy 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

```


\section*{Public Instance Methods}
chunk(*args)
collect()

\section*{collect_concat \{ |obj| block \} \(\rightarrow\)}

\section*{a_lazy_enumerator}
flat_map \{ |obj| block \} \(\rightarrow\) 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

```
drop(p1)
drop_while()
to_enum(method = :each, *args) \(\rightarrow\) lazy_enum
enum_for(method = :each, *args) \(\rightarrow\) lazy_enum
to_enum(method = :each, *args) \{|*args|
block\} \(\rightarrow\) lazy_enum
enum_for(method = :each, *args)\{|*args|
block\} \(\rightarrow\) lazy_enum
Similar to Kernel\#to_enum, except it returns a lazy
enumerator. This makes it easy to define Enumerable 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 \} \(\rightarrow\)
a_lazy_enumerator
flat_map \{ |obj| block \} \(\rightarrow\) 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

\section*{grep(p1)}
lazy()
map()
reject()
select()
slice_after(*args)
slice_before(*args)
slice_when(*args)

\section*{take(p1)}take_while()
to_enum(method = :each, *args) \(\rightarrow\)
lazy_enum
enum_for(method = :each, *args) \(\rightarrow\)
lazy_enum
to_enum(method = :each, *args) \{|*args|
block\} \(\rightarrow\) lazy_enum
enum_for(method = :each, *args)\{|*args|
block\} \(\rightarrow\) lazy_enum
Similar to Kernel\#to_enum, except it returns a lazy
enumerator. This makes it easy to define Enumerable
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 RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator
3.

\title{
class Enumerator::Yielder Lazy
}

\section*{In Files}
\(\doteq\) enumerator.c

\section*{Parent}

Object
Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{module Errno}

Ruby exception objects are subclasses of Exception. However, operating systems typically report errors using plain integers. Module Errno 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.

Errno: :EACCES: :Errno
Errno:: EAGAIN: :Errno
Errno::EINTR::Errno

The full list of operating system errors on your particular platform are available as the constants of Errno.


\section*{In Files}
- error.c

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{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 StandardError or RuntimeError, 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 StandardError or RuntimeError 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
- ScriptError
- LoadError
- NotImplementedError
- SyntaxError
- SecurityError
- SignalException
- Interrupt
- StandardError - default for rescue
- ArgumentError
- UncaughtThrowError
- EncodingError
- FiberError
- IOError
- EOFError
- IndexError
- KeyError
- Stoplteration
- LocalJumpError
- NameError
- NoMethodError
- RangeError
- FloatDomainError
- RegexpError
- RuntimeError - default for raise
- SystemCallError
- Errno::*
- ThreadError
- TypeError
- ZeroDivisionError
- SystemExit
- SystemStackError
- fatal - impossible to rescue

\section*{In Files}
error.c

\section*{Parent}

Object

\section*{Public Class Methods}

\section*{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.
new(msg = nil) \(\rightarrow\) exception
Construct a new Exception object, optionally passing in a message.

\section*{Public Instance Methods}

\section*{exc == obj \(\rightarrow\) 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.

\section*{backtrace \(\rightarrow\) 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

```

\section*{backtrace_locations \(\rightarrow\) array}

Returns any backtrace associated with the exception.
This method is similar to \#backtrace, but the
backtrace is an array of
Thread: :Backtrace: Location.
Now, this method is not affected by \#set_backtrace.

\section*{cause \(\rightarrow\) 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.

\section*{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.

\section*{inspect \(\rightarrow\) string}

Return this exception's class name and message
message \(\rightarrow\) 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.

\section*{set_backtrace(backtrace) \(\rightarrow\) array}

Sets the backtrace information associated with exc.
The backtrace must be an array of String objects or a single String in the format described in \#backtrace.

\section*{to_s \(\rightarrow\) string}

Returns exception's message (or the name of the exception if no message is set).

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{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.

\section*{In Files}
- object.c

\section*{Parent}

\section*{Object}

\section*{Public Instance Methods}
false \& obj \(\rightarrow\) false
nil \& obj \(\rightarrow\) false
And-Returns false. obj is always evaluated as it is the argument to a method call-there is no shortcircuit evaluation in this case.
false \({ }^{\wedge}\) obj \(\rightarrow\) true or false
nil ^ obj \(\rightarrow\) true or false
Exclusive Or—lf obj is nil or false, returns false;
otherwise, returns true.

\section*{inspect()}

Alias for: to_s
to_s \(\rightarrow\) "false"
'nuf said...
Also aliased as: inspect
false | obj \(\rightarrow\) true or false
nil | obj \(\rightarrow\) true or false
Or-Returns false if obj is nil or false; true otherwise.

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{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

\section*{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

\section*{In Files}
cont.c

\section*{Parent}

\section*{Object}

\section*{Public Class Methods}

\section*{current() \(\rightarrow\) 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.
yield(args, ...) \(\rightarrow\) 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.

\section*{Public Instance Methods}

\section*{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.

\section*{resume(args, ...) \(\rightarrow\) 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

\section*{transfer(args, ...) \(\rightarrow\) 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"

```

\begin{abstract}
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
\end{abstract}

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{class FiberError}

Raised when an invalid operation is attempted on a Fiber, in particular when attempting to call/resume a dead fiber, attempting to yield from the root fiber, or calling a fiber across threads.
fiber = Fiber.new\{\}
fiber. resume
fiber. resume

\section*{In Files}
cont.c

\section*{Parent}

\section*{StandardError}

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{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 File can be found in File::Constants.

\section*{In Files}
\(\square\) dir.c
\(\equiv\) file.c
\(\square\) io.c

\section*{Parent}

IO

\section*{Constants}

\section*{ALT_SEPARATOR}
platform specific alternative separator

\section*{PATH_SEPARATOR}
path list separator

\section*{SEPARATOR}
separates directory parts in path

\section*{Separator}
separates directory parts in path

\section*{Public Class Methods}

\section*{absolute_path(file_name [, dir_string] ) \(\rightarrow\) 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.


\section*{atime(file_name) \(\rightarrow\) time}

Returns the last access time for the named file as a Time object).
file_name can be an IO object.


\section*{basename(file_name [, suffix] ) \(\rightarrow\) 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.
File.basename("/home/gumby/work/ruby.rb")
File.basename("/home/gumby/work/ruby.rb", ".rb")
File.basename("/home/gumby/work/ruby.rb", ".*")
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).

blockdev?(file_name) \(\rightarrow\) true or false
Returns true if the named file is a block device.
file_name can be an IO object.
chardev?(file_name) \(\rightarrow\) true or false
Returns true if the named file is a character device.
file_name can be an IO object.
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")

\section*{chown(owner_int, group_int, file_name,... )}

\section*{\(\rightarrow\) 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")

```
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).
```

File.ctime("testfile")

```
delete(file_name, ...) \(\rightarrow\) integer
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.

\section*{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?(".")
dirname(file_name) \(\rightarrow\) 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.


\section*{executable?(file_name) \(\rightarrow\) true or false}

Returns true if the named file is executable by the effective user id of this process.

\section*{executable_real?(file_name) \(\rightarrow\) true or false}

Returns true if the named file is executable by the real user id of this process.

\section*{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.
exists?(file_name) \(\rightarrow\) true or false
Deprecated method. Don't use.
expand_path(file_name [, dir_string] ) \(\rightarrow\) 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 " \(\sim\) ", which expands to the process owner's home directory (the environment variable номе 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 \(\qquad\) FILE \(\qquad\) , that is bin/, then go to the parent, the root of the project and appends lib/mygem.rb.

\section*{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.
```

File.extname("test.rb")
File.extname("a/b/d/test.rb")
File.extname("foo.")
File.extname("test")
File.extname(".profile")
File.extname(".profile.sh")

```

\section*{file?(file) \(\rightarrow\) 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.

\section*{fnmatch( pattern, path, [flags] ) \(\rightarrow\) (true or 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:

\section*{*}

Matches any file. Can be restricted by other values in the glob. Equivalent to / .* /x in regexp.
*
Matches all files regular files
c*
Matches all files beginning with 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.

\section*{[set]}

Matches any one character in set. Behaves exactly like character sets in Regexp, including set negation ([^a-z]).
\(\backslash\)
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.

\section*{Examples:}
\begin{tabular}{lll} 
File.fnmatch('cat', & 'cat') & \(\#=>\) true \\
File.fnmatch('cat', & 'category') & \(\#=>\) fals \\
File.fnmatch('c\{at, ub\}s', 'cats') & \\
File.fnmatch('c\{at, ub\}s', 'cats', File: :FNM_EXTGl \\
& \\
File.fnmatch('c?t', & 'cat') & \(\#=>\) true \\
File.fnmatch('c??t', & 'cat') & \(\#=>\) fals \\
File.fnmatch('c*', & 'cats') & \(\# \Rightarrow>\) true
\end{tabular}
```

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' \# you don't have to do
File.fnmatch(rbfiles, 'main.rb')
File.fnmatch(rbfiles, './main.rb')
File.fnmatch(rbfiles, 'lib/song.rb')
File.fnmatch('**.rb', 'main.rb')
File.fnmatch('**.rb', './main.rb')
File.fnmatch('**.rb', 'lib/song.rb')
File.fnmatch('*', 'dave/.profile')
pattern = '*' '/' '*'
File.fnmatch(pattern, 'dave/.profile', File::FNM
File.fnmatch(pattern, 'dave/.profile', File::FNM_
pattern = '**' '/' 'foo'
File.fnmatch(pattern, 'a/b/c/foo', File::FNM_PATH
File.fnmatch(pattern, '/a/b/c/foo', File::FNM_PA
File.fnmatch(pattern, 'c:/a/b/c/foo', File::FNM
File.fnmatch(pattern, 'a/.b/c/foo', File::FNM_PAT
File.fnmatch(pattern, 'a/.b/c/foo', File::FNM_PA
4

```
fnmatch( pattern, path, [flags] ) \(\rightarrow\) (true or false)
```

fnmatch?( pattern, path, [flags]) }->\mathrm{ (true or

```

\section*{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
c*
Matches all files beginning with c
*C
Matches all files ending with 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 Regexp, including set negation ([^a-z]).
\(\backslash\)
Escapes the next metacharacter.
\{a,b\}

\section*{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.

\section*{Examples:}
```

File.fnmatch('cat', 'cat')
File.fnmatch('cat'
File.fnmatch('c{at,ub}s', 'cats')
File.fnmatch('c{at,ub}s', 'cats', File::FNM_EXTG|
File.fnmatch('c?t', 'cat')
File.fnmatch('c??t', 'cat')
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_DOTMATC
File.fnmatch('.*', '.profile')
rbfiles = '**' '/' '*.rb' \# you don't have to do
File.fnmatch(rbfiles, 'main.rb')
File.fnmatch(rbfiles, './main.rb')
File.fnmatch(rbfiles, 'lib/song.rb')
File.fnmatch('**.rb', 'main.rb')
File.fnmatch('**.rb', './main.rb')
File.fnmatch('**.rb', 'lib/song.rb')
File.fnmatch('*', 'dave/.profile')

```
```

pattern = '*' '/' '*'
File.fnmatch(pattern, 'dave/.profile', File::FNM
File.fnmatch(pattern, 'dave/.profile', File::FNM
pattern = '**' '/' 'foo'
File.fnmatch(pattern, 'a/b/c/foo', File::FNM_PAT
File.fnmatch(pattern, '/a/b/c/foo', File::FNM_PA
File.fnmatch(pattern, 'c:/a/b/c/foo', File::FNM
File.fnmatch(pattern, 'a/.b/c/foo', File::FNM_PA
File.fnmatch(pattern, 'a/.b/c/foo', File::FNM_PA

```

\section*{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".

File.ftype("testfile")
File.ftype("/dev/tty")
File.ftype("/tmp/.X11-unix/X0")

\section*{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.

\section*{identical?(file_1, file_2) \(\rightarrow\) 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")
p File.identical?("a", "./a")
File.link("a", "b")

```
```

p File.identical?("a", "b")
File.symlink("a", "c")
p File.identical?("a", "c")
open("d", "w") {}
p File.identical?("a", "d")

```

\section*{join(string, ...) \(\rightarrow\) string}

Returns a new string formed by joining the strings using File: : SEPARATOR.


\section*{Ichmod(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.

\section*{Ichown(owner_int, group_int, file_name,..) \(\rightarrow\) 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.

\section*{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.
\(\square\)

Istat(file_name) \(\rightarrow\) stat
Same as File: :stat, but does not follow the last symbolic link. Instead, reports on the link itself.
```

File.symlink("testfile", "link2test")
File.stat("testfile").size
File.lstat("link2test").size
File.stat("link2test").size

```

\section*{mtime(file_name) \(\rightarrow\) time}

Returns the modification time for the named file as a Time object.
file_name can be an IO object.

new(filename, mode='r' [, opt]) \(\rightarrow\) file new(filename [, mode [, perm]] [, opt]) \(\rightarrow\) 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.

\section*{Examples}
```

f = File.new("testfile", "r")
f = File.new("newfile", "w+")
f = File.new("newfile", File::CREAT|File::TRUNC|F

```
open(filename, mode="r" [, opt]) \(\rightarrow\) file open(filename [, mode [, perm]] [, opt]) \(\rightarrow\) file open(filename, mode="r" [, opt]) \{|file| block \} \(\rightarrow\) obj open(filename [, mode [, perm]] [, opt]) \{|file| block \} \(\rightarrow\) obj
With no associated block, File.open is a synonym for \(\because\) 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 ::new for further discussion.

See IO.new for a description of the mode and opt parameters.

\section*{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.

\section*{path(path) \(\rightarrow\) string}

Returns the string representation of the path

> File. path("/dev/null")
> File.path(Pathname.new("/tmp"))

Returns true if the named file is a pipe.
file_name can be an IO object.
readable?(file_name) \(\rightarrow\) true or false
Returns true if the named file is readable by the effective user id of this process.

\section*{readable_real?(file_name) \(\rightarrow\) true or false} Returns true if the named file is readable by the real user id of this process.

\section*{readlink(link_name) \(\rightarrow\) file_name}

Returns the name of the file referenced by the given link. Not available on all platforms.

realdirpath(pathname [, dir_string]) \(\rightarrow\) 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.

\section*{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.

\section*{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")

\section*{setgid?(file_name) \(\rightarrow\) true or false}

Returns true if the named file has the setgid bit set.

\section*{setuid?(file_name) \(\rightarrow\) true or false}

Returns true if the named file has the setuid bit set.

\section*{size(file_name) \(\rightarrow\) integer}

Returns the size of file_name.
file_name can be an IO object.

\section*{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.

\section*{socket?(file_name) \(\rightarrow\) true or false}

Returns true if the named file is a socket.
file_name can be an IO object.
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.

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.
symlink(old_name, new_name) \(\rightarrow \mathbf{0}\)
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")

Returns true if the named file is a symbolic link.

\section*{truncate(file_name, integer) \(\rightarrow 0\)}

Truncates the file file_name to be at most integer bytes long. Not available on all platforms.
```

f = File.new("out", "w")
f.write("1234567890")
f.close
File.truncate("out", 5)
File.size("out")

```

\section*{umask() \(\rightarrow\) integer}

\section*{umask(integer) \(\rightarrow\) 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.

File. umask(0006)
File.umask
delete(file_name, ...) \(\rightarrow\) integer 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.
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.

\section*{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.
```

File.world_readable?("/etc/passwd")
m = File.world_readable?("/etc/passwd")
sprintf("%o", m)

```

\section*{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.
```

File.world_writable?("/tmp")
m = File.world_writable?("/tmp")
sprintf("%o", m)

```

\section*{writable?(file_name) \(\rightarrow\) true or false}

Returns true if the named file is writable by the effective user id of this process.

\section*{writable_real?(file_name) \(\rightarrow\) true or false}

Returns true if the named file is writable by the real user id of this process.

\section*{zero?(file_name) \(\rightarrow\) true or false}

Returns true if the named file exists and has a zero size.
file_name can be an IO object.

\section*{Public Instance Methods}
atime \(\rightarrow\) time
Returns the last access time (a Time object)
```

for <i>file</i>, or epoch if <i>file</> has not
File.new("testfile").atime

```
birthtime \(\rightarrow\) time
Returns the birth time for file.
Note that on Windows (NTFS), returns creation time
(birth time).

chmod(mode_int) \(\rightarrow \mathbf{0}\)
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)

```
chown(owner_int, group_int ) \(\rightarrow \mathbf{0}\)
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)

```

\section*{ctime \(\rightarrow\) time}

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).
```

File.new("testfile").ctime

```

\section*{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):
\begin{tabular}{|c|c|}
\hline LOCK_EX & Exclusive lock. Only one process may exclusive lock for a given file at \\
\hline LOCK_NB & Don't block when locking. May be com with other lock options using logica \\
\hline LOCK_SH & Shared lock. Multiple processes may \\
\hline
\end{tabular}


\section*{Example:}
```

    # update a counter using write lock
    
# don't use "w" because it truncates the file be

File.open("counter", File::RDWR|File::CREAT, 0644
f.flock(File::LOCK_EX)
value = f.read.to_i + 1
f.rewind
f.write("\#{value}\n")
f.flush
f.truncate(f.pos)
}

# read the counter using read lock

File.open("counter", "r") {|f|
f.flock(File::LOCK_SH)
p f.read
}

```

\section*{Istat \(\rightarrow\) stat}

Same as Io\#stat, but does not follow the last
symbolic link. Instead, reports on the link itself.
```

File.symlink("testfile", "link2test")
File.stat("testfile").size
f = File.new("link2test")
f.lstat.size
f.stat.size
\#=> 8
\#=> 66

```

\section*{mtime \(\rightarrow\) time}

Returns the modification time for file.


\section*{path \(\rightarrow\) filename}
to_path \(\rightarrow\) filename
Returns the pathname used to create file as a string.
Does not normalize the name.
```

File.new("testfile").path
File.new("/tmp/../tmp/xxx", "w").path

```
size \(\rightarrow\) integer
Returns the size of file in bytes.
```

File.new("testfile").size

```

\section*{path \(\rightarrow\) filename}
to_path \(\rightarrow\) filename
Returns the pathname used to create file as a string.
Does not normalize the name.
```

File.new("testfile").path
File.new("/tmp/../tmp/xxx", "w").path

```
truncate(integer) \(\rightarrow 0\)
Truncates file to at most integer bytes. The file must be opened for writing. Not available on all platforms.
```

f = File.new("out", "w")
f.syswrite("1234567890")
f.truncate(5)
f.close()
File.size("out")

```

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{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.

\section*{In Files}
dir.c

\section*{Constants}

\section*{APPEND}
append on each write

\section*{BINARY}
disable line code conversion

\section*{CREAT}
create file if it does not exist

\section*{DIRECT}

Try to minimize cache effects of the I/O to and from this file.

\section*{DSYNC}
any write operation perform synchronously except some meta data

\section*{EXCL}
error if CREAT and the file exists

\section*{LOCK_EX}
exclusive lock. see File\#flock

\section*{LOCK_NB}
non-blocking lock. used with LOCK_SH or LOCK_EX.
see File\#flock

\section*{LOCK_SH}
shared lock. see File\#flock

\section*{LOCK_UN}
unlock. see File\#flock

\section*{NOATIME}
do not change atime

\section*{NOCTTY}
not to make opened IO the controlling terminal device

\section*{NOFOLLOW}
do not follow symlinks

\section*{NONBLOCK}
do not block on open or for data to become available

\section*{NULL}

Name of the null device

\section*{RDONLY}
open for reading only
RDWR
open for reading and writing

\section*{RSYNC}
any read operation perform synchronously. used with SYNC or DSYNC.

SYNC
any write operation perform synchronously

\section*{TRUNC}
truncate size to 0

\section*{WRONLY}
open for writing only

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{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.

\section*{In Files}
dir.c

\section*{Parent}

\section*{Object}

\section*{Included Modules}
- Comparable

\section*{Public Class Methods}

File::Stat.new(file_name) \(\rightarrow\) stat
Create a File::Stat object for the given file name (raising an exception if the file doesn't exist).

\section*{Public Instance Methods}

\section*{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
```

f1 = File.new("f1", "w")
sleep 1
f2 = File.new("f2", "w")
f1.stat <=> f2.stat

```
atime \(\rightarrow\) time
Returns the last access time for this file as an object of class Time.


\section*{birthtime \(\rightarrow\) 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
File.stat("testfile").mtime
File.stat("testfile").ctime
2014-02-24
2014
File.stat("testfile").atime
2014-02-24

```


\section*{blksize \(\rightarrow\) 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

```

\section*{blockdev? \(\rightarrow\) 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?
> File.stat("/dev/hda1").blockdev?

\section*{blocks \(\rightarrow\) 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

```

\section*{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.
```

File.stat("/dev/tty").chardev?

```
ctime \(\rightarrow\) aTime
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).

dev \(\rightarrow\) fixnum
Returns an integer representing the device on which stat resides.
```

File.stat("testfile").dev

```
dev_major \(\rightarrow\) fixnum
Returns the major part of File_Stat\#dev or nil.
```

File.stat("/dev/fd1").dev_major
File.stat("/dev/tty").dev_major

```

\section*{dev_minor \(\rightarrow\) fixnum}

Returns the minor part of File_Stat\#dev or nil.
File.stat("/dev/fd1").dev_minor
File.stat("/dev/tty").dev_minor
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?(".")

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?

\section*{executable_real? \(\rightarrow\) true or false}

Same as executable?, but tests using the real owner of the process.

\section*{file? \(\rightarrow\) true or false}

Returns true if stat is a regular file (not a device file, pipe, socket, etc.).
```

File.stat("testfile").file?

```

\section*{ftype \(\rightarrow\) string}

Identifies the type of stat. The return string is one of: "file", "directory", "characterSpecial",
"blockSpecial", "fifo", "link", "socket", or "unknown".

gid \(\rightarrow\) fixnum
Returns the numeric group id of the owner of stat.
```

File.stat("testfile").gid

```

\section*{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

```

\section*{ino \(\rightarrow\) fixnum}

Returns the inode number for stat.
```

File.stat("testfile").ino \#\#> 1083669

```

\section*{inspect \(\rightarrow\) string}

Produce a nicely formatted description of stat.


\section*{mode \(\rightarrow\) fixnum}

Returns an integer representing the permission bits of stat. The meaning of the bits is platform dependent; on Unix systems, see stat(2).
```

File.chmod(0644, "testfile")
s = File.stat("testfile")
sprintf("%o", s.mode)

```

\section*{mtime \(\rightarrow\) aTime}

Returns the modification time of stat.


\section*{nlink \(\rightarrow\) fixnum}

Returns the number of hard links to stat.
```

File.stat("testfile").nlink
File.link("testfile", "testfile.bak")
File.stat("testfile").nlink

```

\section*{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?
File.stat("/etc/passwd"). owned?
pipe? \(\rightarrow\) true or false
Returns true if the operating system supports pipes and stat is a pipe; false otherwise.

\section*{rdev \(\rightarrow\) 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.

File.stat("/dev/fd1").rdev
File.stat("/dev/tty").rdev
rdev_major \(\rightarrow\) fixnum
Returns the major part of File_Stat\#rdev or nil.
```

File.stat("/dev/fd1").rdev_major
File.stat("/dev/tty").rdev_major

```
rdev_minor \(\rightarrow\) fixnum

Returns the minor part of File_Stat\#rdev or nil.

File.stat("/dev/fd1").rdev_minor
File.stat("/dev/tty").rdev_minor

\section*{readable? \(\rightarrow\) true or false}

Returns true if stat is readable by the effective user id of this process.
```

File.stat("testfile").readable?

```

\section*{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?

```

\section*{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?

```

\section*{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?
size \(\rightarrow\) fixnum
Returns the size of stat in bytes.

File.stat("testfile").size
size \(\rightarrow\) integer
Returns the size of stat in bytes.
File.stat("testfile").size

\section*{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?

\section*{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?

```

\section*{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") File.stat("alink").symlink?
File.lstat("alink").symlink?
uid \(\rightarrow\) fixnum

Returns the numeric user id of the owner of stat.
File.stat("testfile").uid

\section*{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).
```

m = File.stat("/etc/passwd").world_readable?
sprintf("%o", m)

```

\section*{world_writable? \(\rightarrow\) 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).
```

m = File.stat("/tmp").world_writable?
sprintf("%o", m)

```

\section*{writable? \(\rightarrow\) true or false}

Returns true if stat is writable by the effective user id of this process.

File.stat("testfile").writable?

\section*{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?
zero? \(\rightarrow\) true or false
Returns true if stat is a zero-length file; false otherwise.

File.stat("testfile").zero?

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{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).

\section*{In Files}
- file.c

\section*{Public Instance Methods}
blockdev?(file_name) \(\rightarrow\) true or false
Returns true if the named file is a block device.
file_name can be an IO object.
chardev?(file_name) \(\rightarrow\) true or false
Returns true if the named file is a character device.
file_name can be an IO object.
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.

\section*{File.directory?(".")}

\section*{executable?(file_name) \(\rightarrow\) true or false}

Returns true if the named file is executable by the effective user id of this process.
executable_real?(file_name) \(\rightarrow\) true or false
Returns true if the named file is executable by the real user id of this process.

\section*{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.

\section*{exists?(file_name) \(\rightarrow\) true or false}

Deprecated method. Don't use.

\section*{file?(file) \(\rightarrow\) 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.

\section*{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) \(\rightarrow\) 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")
p File.identical?("a", "./a")
File.link("a", "b")
p File.identical?("a", "b")
File.symlink("a", "c")
p File.identical?("a", "c")
open("d", "w") {}
p File.identical?("a", "d")

```

\section*{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.
pipe?(file_name) \(\rightarrow\) true or false
Returns true if the named file is a pipe.
file_name can be an IO object.
readable?(file_name) \(\rightarrow\) true or false
Returns true if the named file is readable by the effective user id of this process.

Returns true if the named file is readable by the real user id of this process.
setgid?(file_name) \(\rightarrow\) true or false
Returns true if the named file has the setgid bit set.
setuid?(file_name) \(\rightarrow\) true or false
Returns true if the named file has the setuid bit set.
size(file_name) \(\rightarrow\) 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.
socket?(file_name) \(\rightarrow\) true or false
Returns true if the named file is a socket.
file_name can be an IO object.
sticky?(file_name) \(\rightarrow\) true or false
Returns true if the named file has the sticky bit set.

\section*{symlink?(file_name) \(\rightarrow\) true or false}

Returns true if the named file is a symbolic link.

\section*{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.
```

File.world_readable?("/etc/passwd")
m = File.world_readable?("/etc/passwd")
sprintf("%o", m)

```

\section*{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.
```

File.world_writable?("/tmp")
m = File.world_writable?("/tmp")
sprintf("%o", m)

```

\section*{writable?(file_name) \(\rightarrow\) true or false}

Returns true if the named file is writable by the effective user id of this process.

\section*{writable_real?(file_name) \(\rightarrow\) true or false}

Returns true if the named file is writable by the real user id of this process.
zero?(file_name) \(\rightarrow\) true or false

Returns true if the named file exists and has a zero size.
file_name can be an IO object.

Generated by RDoc 3.12.2. Generated with the Darkfish Rdoc Generator 3.

\section*{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.

\section*{In Files}
numeric.c

\section*{Parent}

\section*{Integer}

\section*{Public Instance Methods}
fix \% other \(\rightarrow\) real
modulo(other) \(\rightarrow\) real
Returns fix modulo other.
See Numeric\#divmod for more information.
fix \& integer \(\rightarrow\) integer_result
Bitwise AND.
fix * numeric \(\rightarrow\) 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.

\section*{fix ** numeric \(\rightarrow\) numeric_result}

Raises fix to the power of numeric, which may be negative or fractional.
\begin{tabular}{lll}
2 & \(* *\) & 3 \\
2 & ** & -1 \\
2 & ** & 0.5
\end{tabular}
fix + numeric \(\rightarrow\) 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 \(\rightarrow\) 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.
-fix \(\rightarrow\) integer
Negates fix, which may return a Bignum.

\section*{fix / numeric \(\rightarrow\) 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.

\section*{fix < real \(\rightarrow\) true or false}

Returns true if the value of fix is less than that of real.
fix \(\ll\) count \(\rightarrow\) integer
Shifts fix left count positions, or right if count is negative.
fix \(<=\) real \(\rightarrow\) true or false
Returns true if the value of fix is less than or equal to that of real.

\section*{fix <=> numeric \(\rightarrow-1,0,+1\) or nil}

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 Comparable module.
nil is returned if the two values are incomparable.
fix \(==\) other \(\rightarrow\) true or false

Return true if fix equals other numerically.
```

1 == 2
1 == 1.0
true

```

\section*{fix \(==\) other \(\rightarrow\) true or false}

Return true if fix equals other numerically.
```

1 == 2
1 == 1.0
true

```

\section*{fix \(>\) real \(\rightarrow\) true or false}

Returns true if the value of fix is greater than that of real.
fix >= real \(\rightarrow\) true or false
Returns true if the value of fix is greater than or equal to that of real.

\section*{fix >> count \(\rightarrow\) integer}

Shifts fix right count positions, or left if count is negative.
fix[n] \(\rightarrow \mathbf{0 , 1}\)
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

## fix ^ integer $\rightarrow$ integer_result

Bitwise EXCLUSIVE OR.
abs $\rightarrow$ integer
magnitude $\rightarrow$ integer
Returns the absolute value of fix.

```
-12345.abs
12345
12345.abs ##> 12345
```


## bit_length $\rightarrow$ 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 $2^{* *} \mathrm{n}$ is $\mathrm{n}+1$.) 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
```


## div(numeric) $\rightarrow$ integer

Performs integer division: returns integer result of dividing fix by numeric.

## divmod(numeric) $\rightarrow$ array

See Numeric\#divmod.

## even? $\rightarrow$ true or false

Returns true if fix is an even number.
fdiv(numeric) $\rightarrow$ float
Returns the floating point result of dividing fix by
numeric.
inspect(p1 = v1)
Alias for: to_s
abs $\rightarrow$ integer
magnitude $\rightarrow$ integer
Returns the absolute value of fix.

```
-12345.abs
12345.abs
12345 12345
```

fix \% other $\rightarrow$ real
modulo(other) $\rightarrow$ real
Returns fix modulo other.
See Numeric\#divmod for more information.
odd? $\rightarrow$ true or false
Returns true if fix is an odd number.
size $\rightarrow$ fixnum
Returns the number of bytes in the machine representation of fix.

```
1.size
-1.size
2147483647.size
```

(5) next $\rightarrow$ integer
succ $\rightarrow$ integer
Returns the Integer equal to int +1 .

```
1.next
(-1).next
```

to_f $\rightarrow$ float
Converts fix to a Float.
to_s(base=10) $\rightarrow$ string
Returns a string containing the representation of fix radix base (between 2 and 36 ).

```
12345.to_s(2)
12345.to_s(8)
12345.to_s(10)
12345.to_s(16)
12345.to_s(36)
```

Also aliased as: inspect

## zero? $\rightarrow$ true or false

Returns true if fix is zero.

## fix | integer $\rightarrow$ integer_result

Bitwise OR.
(8) $\sim$ fix $\rightarrow$ integer

One's complement: returns a number where each bit is flipped.

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 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:

- docs.sun.com/source/8063568/ncg_goldberg.html
- wiki.github.com/rdp/ruby_tutorials_core/ruby-talk-faq\#wiki-floats_imprecise
- en.wikipedia.org/wiki/Floating_point\#Accuracy_pri


## In Files

complex.c
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 $2.2204460492503131 \mathrm{e}-16$.

## 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 $1.7976931348623157 \mathrm{e}+308$.

## 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

MIN. 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

float \% other $\rightarrow$ float
modulo(other) $\rightarrow$ float
Return the modulo after division of float by other.
6543.21. modulo(137)
6543.21.modulo(137.24)

## float * other $\rightarrow$ float

Returns a new float which is the product of float and other.
float ** other $\rightarrow$ float
Raises float to the power of other.

```
2.0**3
```


## float + other $\rightarrow$ float

Returns a new float which is the sum of float and other.
float - other $\rightarrow$ float
Returns a new float which is the difference of float and other.
-float $\rightarrow$ float
Returns float, negated.
float / other $\rightarrow$ float

Returns a new float which is the result of dividing float by other.

## float $<$ real $\rightarrow$ true or false

Returns true if float is less than real.
The result of NaN < NaN is undefined, so the implementation-dependent value is returned.

## float <= real $\rightarrow$ 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.

## float <=> real $\rightarrow-1,0,+1$ 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.

## float $==$ obj $\rightarrow$ true or false

Returns true only if obj has the same value as float. Contrast this with \#eql?, which requires obj to be a Float.

The result of $\mathrm{NaN}==$ NaN is undefined, so the implementation-dependent value is returned.

```
1.0 == 1
```


## float $==$ obj $\rightarrow$ true or false

Returns true only if obj has the same value as float.
Contrast this with \#eql?, which requires obj to be a Float.

The result of $\mathrm{NaN}==\mathrm{NaN}$ is undefined, so the implementation-dependent value is returned.

```
1.0 == 1
```


## float $>$ real $\rightarrow$ true or false

Returns true if float is greater than real.
The result of $\mathrm{NaN}>\mathrm{NaN}$ is undefined, so the implementation-dependent value is returned.

## float >= real $\rightarrow$ true or false

Returns true if float is greater than or equal to real.
The result of $\mathrm{NaN}>=\mathrm{NaN}$ is undefined, so the implementation-dependent value is returned.
abs $\rightarrow$ float
magnitude $\rightarrow$ float
Returns the absolute value of float.

## (-34.56).abs

-34.56.abs

## arg $\rightarrow 0$ or float

angle $\rightarrow 0$ or float
phase $\rightarrow 0$ or float
Returns 0 if the value is positive, pi otherwise.
(8) $\arg \rightarrow 0$ or float
angle $\rightarrow 0$ or float
phase $\rightarrow 0$ or float
Returns 0 if the value is positive, pi otherwise.

## ceil $\rightarrow$ integer

Returns the smallest Integer greater than or equal to float.

```
1.2.ceil
2.0.ceil
(-1.2).ceil
(-2.0).ceil
```

coerce(numeric) $\rightarrow$ 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.
1.2.coerce(3)
2.5.coerce(1.1)

## denominator $\rightarrow$ integer

Returns the denominator (always positive). The result is machine dependent.
See numerator.
divmod(numeric) $\rightarrow$ array
See Numeric\#divmod.
42.0.divmod 6

## eql?(obj) $\rightarrow$ true or false

Returns true only if obj is a Float 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)
```


## fdiv(numeric) $\rightarrow$ float

quo(numeric) $\rightarrow$ float
Returns float / numeric, same as Float\#/.

## finite? $\rightarrow$ true or false

Returns true if float is a valid IEEE floating point number (it is not infinite, and \#nan? is false).

## floor $\rightarrow$ integer

Returns the largest integer less than or equal to float.

```
1.2.floor
2.0.floor
(-1.2).floor
(-2.0).floor
```

hash $\rightarrow$ integer
Returns a hash code for this float.
See also Object\#hash.
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 $\rightarrow$ float
magnitude $\rightarrow$ float
Returns the absolute value of float.
(-34.56).abs
-34.56.abs
float $\%$ other $\rightarrow$ float
modulo(other) $\rightarrow$ float
Return the modulo after division of float by other.
6543.21. modulo(137)
6543.21.modulo(137.24)
nan? $\rightarrow$ true or false
Returns true if float is an invalid IEEE floating point

## number.

```
a = -1.0
a.nan?
#=> false
a = 0.0/0.0 #=> NaN
a.nan? ### true
```


## next_float $\rightarrow$ 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:

|  |  |  |
| :---: | :---: | :---: |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

```
f = 0.0
100.times { f += 0.1 }
p f
                                    .999999999999
p 10-f
p(10.0.next_float-10)
#=> 1.77635683940
p((10-f)/(10.0.next_float-10)) ##> 11.0
p((10-f)/(10*Float::EPSILON)) #=> 8.8
p "%a" % f
```


## numerator $\rightarrow$ integer

Returns the numerator. The result is machine dependent.

```
n = 0.3.numerator
5404319552844595
d = 0.3.denominator
    18014398509481984
n.fdiv(d)
```


## arg $\rightarrow 0$ or float

sangle $\rightarrow 0$ or float
phase $\rightarrow 0$ or float
Returns 0 if the value is positive, pi otherwise.

## prev_float $\rightarrow$ float

Returns the previous representable floatint-point number.
(-Float::MAX).\#prev_float and (-
Float::INFINITY).\#prev_float is -Float::INFINITY.
Float::NAN.prev_float is Float::NAN.
For example:

```
p 0.01.prev_float
p 1.0.prev_float
p 100.0.prev_float
0.009999999999999998
p 1.0.prev_float
0.9999999999999999
p 100.0.prev_float
99.99999999999999
```

```
p 0.01 - 0.01.prev_float ##> 1.734723475976807
p 1.0 - 1.0.prev_float ##> 1.110223024625156.
p 100.0 - 100.0.prev_float ##> 1.421.085471520200
f = 0.01; 20.times { printf "%-20a %s\n", f, f.tc
#=> 0x1.47ae147ac147bp-7 0.01
    0x1.47ae147ae147ap-7 0.009999999999999998
    0x1.47ae147ae1479p-7 0.009999999999999997
    0x1.47ae147ae1478p-7 0.009999999999999995
    0x1.47ae147ae1477p-7 0.009999999999999993
    0x1.47ae147ae1476p-7 0.009999999999999992
    0x1.47ae147ae1475p-7 0.00999999999999999
    0x1.47ae147ae1474p-7 0.009999999999999988
    0x1.47ae147ae1473p-7 0.009999999999999986
    0x1.47ae147ae1472p-7 0.009999999999999985
    0x1.47ae147ae1471p-7 0.009999999999999983
    0x1.47ae147ae147p-7 0.009999999999999981
    0x1.47ae147ae146fo-7 0.00999999999999998
    0x1.47ae147ae146ep-7 0.009999999999999978
    0x1.47ae147ae146dp-7 0.009999999999999976
    0x1.47ae147ae146cp-7 0.009999999999999974
    0x1.47ae147ae146bp-7 0.009999999999999972
    0x1.47ae147ae146ap-7 0.009999999999999997
    0x1.47ae147ae1469p-7 0.009999999999999969
    0x1.47ae147ae1468p-7 0.009999999999999967
```


## fdiv(numeric) $\rightarrow$ float <br> quo(numeric) $\rightarrow$ float

Returns float / numeric, same as Float\#/.

## 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.

```
0.3.rationalize
1.333.rationalize
1.333.rationalize(0.01)
#=> (3/10)
#=> (1333/1000)
#=> (4/3)
```

See to r.

## round([ndigits]) $\rightarrow$ 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.

```
1.4.round
1.5.round
1.6.round
(-1.5).round
1.234567.round(2)
1.234567.round(3) ##> 1.235
1.234567.round(4) #=> 1.2346
1.234567.round(5) ##> 1.23457
34567.89.round(-5) ##> 0
34567.89.round(-4) #=> 30000
34567.89.round(-3) #=> 35000
34567.89.round(-2) ##> 34600
34567.89.round(-1) ##> 34570
34567.89.round(0) ##> 34568
34567.89.round(1) #=> 34567.9
34567.89.round(2) #=> 34567.89
34567.89.round(3) #=> 34567.89
```


## to_f $\rightarrow$ self

Since float is already a float, returns self.

## to_i $\rightarrow$ integer

to_int $\rightarrow$ integer
truncate $\rightarrow$ integer
Returns the float truncated to an Integer.
Synonyms are to_i, to_int, and truncate.
to_i $\rightarrow$ integer
to_int $\rightarrow$ integer
truncate $\rightarrow$ integer
Returns the float truncated to an Integer.
Synonyms are to_i, to_int, and truncate.

## to_r $\rightarrow$ 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.

## to_s $\rightarrow$ 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
to_i $\rightarrow$ integer
to_int $\rightarrow$ integer
truncate $\rightarrow$ integer
Returns the float truncated to an Integer.
Synonyms are to_i, to_int, and truncate.

Returns true if float is 0.0 .

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 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

## In Files

numeric.c

## Parent

## RangeError

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## module GC

The GC module provides an interface to Ruby's mark and sweep garbage collection mechanism.

Some of the underlying methods are also available via the ObjectSpace module.

You may obtain information about the operation of the GC through GC::Profiler.

## In Files

gc.c

## Constants

## INTERNAL_CONSTANTS

OPTS

## Public Class Methods

count $\rightarrow$ Integer
The number of times GC occurred.
It returns the number of times GC occurred since the process started.

## disable $\rightarrow$ true or false

Disables garbage collection, returning true if garbage collection was already disabled.

GC. disable<br>GC. disable

## enable $\rightarrow$ true or false

Enables garbage collection, returning true if garbage collection was previously disabled.

```
GC.disable
GC.enable
GC.enable
```

latest_gc_info -> \{:gc_by $\rightarrow$ :newobj\}
latest_gc_info(hash) $\rightarrow$ hash
latest_gc_info(:major_by) $\rightarrow$ :malloc

Returns information about the most recent garbage collection.
malloc_allocated_size $\rightarrow$ Integer
Returns the size of memory allocated by malloc().
Only available if ruby was built with
CALC_EXACT_MALLOC_SIZE.
malloc_allocations $\rightarrow$ Integer
Returns the number of malloc() allocations.
Only available if ruby was built with
CALC_EXACT_MALLOC_SIZE.
start $\rightarrow$ nil
garbage_collect $\rightarrow$ nil
start(full_mark: true, immediate_sweep:
true) $\rightarrow$ nil
garbage_collect(full_mark: true, 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 GC. 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 $\rightarrow$ Hash

stat(hash) $\rightarrow$ hash
stat(:key) $\rightarrow$ Numeric
Returns a Hash containing information about the GC.
The hash includes information about internal statistics about GC such as:

```
{
    :count=>0,
    :heap_allocated_pages=>24,
    :heap_sorted_length=>24,
    :heap_allocatable_pages=>0,
    :heap_available_slots=>9783,
    :heap_live_slots=>7713,
```

```
    :heap_free_slots=>2070,
    :heap_final_slots=>0,
    :heap_marked_slots=>0,
    : heap_swept_slots=>0,
    :heap_eden_pages=>24,
    :heap_tomb_pages=>0,
    :total_allocated_pages=>24,
    :total_freed_pages=>0,
    :total_allocated_objects=>7796,
    :total_freed_objects=>83,
    :malloc_increase_bytes=>2389312,
    :malloc_increase_bytes_limit=>16777216,
    :minor_gc_count=>0,
    :major_gc_count=>0,
    :remembered_wb_unprotected_objects=>0,
    :remembered_wb_unprotected_objects_limit=>0,
    :old_objects=>0,
    :old_objects_limit=>0,
    : oldmalloc_increase_bytes=>2389760,
    : oldmalloc_increase_bytes_limit=>16777216
    }
4
```

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 $\rightarrow$ fixnum, true or false

Returns current status of GC stress mode.

## stress $=$ flag $\rightarrow$ flag

Updates the GC stress mode.
When stress mode is enabled, the GC is invoked at every 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

## 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 $\rightarrow$ nil
garbage_collect $\rightarrow$ nil
start(full_mark: true, immediate_sweep:
true) $\rightarrow$ nil
garbage_collect(full_mark: true, 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 GC. 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 RDoc 3.12.2. Generated with the Darkfish Rdoc Generator 3.

## module GC::Profiler

The GC profiler provides access to information on GC runs including time, length and object space size.

Example:

> GC::Profiler.enable
> require 'rdoc/rdoc'
> GC: :Profiler.report
> GC::Profiler.disable

See also GC.count, GC.malloc_allocated_size and GC.malloc_allocations

## In Files

gc.c

## Public Class Methods

GC::Profiler.clear $\rightarrow$ nil
Clears the GC profiler data.

GC::Profiler.disable $\rightarrow$ nil
Stops the GC profiler.

## GC::Profiler.enable $\rightarrow$ nil

Starts the GC profiler.

## GC::Profiler.enabled? $\rightarrow$ true or false

The current status of GC profile mode.

## GC::Profiler.raw_data $\rightarrow$ [Hash, ...]

Returns an Array of individual raw profile data Hashes ordered from earliest to latest by : GC_INVOKE_TIME.

For example:

```
t
    {
        :GC_TIME=>1.3000000000000858e-05,
        :GC_INVOKE_TIME=>0.010634999999999999,
        :HEAP_USE_SIZE=>289640,
        :HEAP_TOTAL_SIZE=>588960,
        :HEAP_TOTAL_OBJECTS=>14724,
        :GC_IS_MARKED=>false
    },
]
```

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 $\rightarrow$ String

Returns a profile data report such as:


## GC::Profiler.total_time $\rightarrow$ float

The total time used for garbage collection in seconds

Generated by RDoc 3.12.2.

Generated with the Darkfish Rdoc Generator 3.

## class Hash

A Hash is a dictionary-like collection of unique keys and their values. Also called associative arrays, they are similar to Arrays, but where an Array uses integers as its index, a Hash allows you to use any object type.

Hashes enumerate their values in the order that the corresponding keys were inserted.

A Hash 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: "Aria

Each named key is a symbol you can access in hash:

A Hash can also be created through its $::$ new 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 Hash requires using its key:

```
puts grades["Jane Doe"]
```


## 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 eql? to each other.

A user-defined class may be used as a hash key if the hash and eql? 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 eql? 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
```

See also Object\#hash and Object\#eql?

## In Files

hash.c

## Parent

## Object

## Included Modules

## - Enumerable

## Public Class Methods

## Hash[ key, value, ... ] $\rightarrow$ new_hash

 Hash[ [ [key, value], ... ]] $\rightarrow$ new_hash Hash[ object ] $\rightarrow$ new_hashCreates 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.

new $\rightarrow$ new_hash
new(obj) $\rightarrow$ new_hash
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.


## try_convert(obj) $\rightarrow$ hash or nil

Try to convert obj into a hash, using \#to_hash method. Returns converted hash or nil if obj cannot be converted for any reason.

```
Hash.try_convert({1=>2})
Hash.try_convert("1=>2") # #> nil
```


## Public Instance Methods

## 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 object\#==) 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
```

hsh[key] $\rightarrow$ value
Element Reference—Retrieves the value object corresponding to the key object. If not found, returns the default value (see Hash: :new for details).

```
h = { "a" => 100, "b" => 200 }
h["a"]
h["c"]
```

hsh[key] = value $\rightarrow$ value
store(key, value) $\rightarrow$ value

## Element Assignment

Associates the value given by value with the key given by key.

```
h = { "a" => 100, "b" => 200 }
h["a"] = 9
h["c"] = 4
h
h.store("d", 42)
h
```

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
h.key(200).equal? b #=> true
```


## any? [\{ |(key, value)| block \}] $\rightarrow$ true or false

See also Enumerable\#any?

## assoc(obj) $\rightarrow$ 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.

```
h = {"colors" => ["red", "blue", "green"],
    "letters" => ["a", "b", "c" ]}
h.assoc("letters")
h.assoc("foo")
```



## clear $\rightarrow$ hsh

Removes all key-value pairs from $h s h$.


## compare_by_identity $\rightarrow$ hsh

Makes hsh compare its keys by their identity, i.e. it will consider exact same objects as same keys.

```
h1 = { "a" => 100, "b" => 200, :c => "c" }
h1["a"]
h1.compare_by_identity
h1.compare_by_identity?
h1["a".dup] #=> ni.1
h1[:c]
> true
different objects
```



## compare_by_identity? $\rightarrow$ true or false

Returns true if $h s h$ will compare its keys by their identity. Also see Hash\#compare_by_identity.

## default(key=nil) $\rightarrow$ obj

Returns the default value, the value that would be returned by hsh 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)
```

h = Hash.new {|h,k| h[k] = k.to_i*10}
h.default
\#=> {
h.default(2) \#=> 20

```
\(\rightarrow \square \rightarrow\)
default \(=\) obj \(\rightarrow\) 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.
```

h = { "a" => 100, "b" => 200 }
h.default = "Go fish"
h["a"]
\#=> 100
h["z"] \#=> "Go fish"

# This doesn't do what you might hope

h.default = proc do |hash, key|
hash[key] = key + key
end
h[2]
h["cat"] \#=> \#[Proc:0x401b3948@-:6](Proc:0x401b3948@-:6)

```

\section*{default_proc \(\rightarrow\) anObject}

If Hash: : new was invoked with a block, return that block, otherwise return nil.
```

h = Hash.new {|h,k| h[k] = k*k }
p = h.default_proc
a = []
p.call(a, 2)
a

default_proc = proc_obj or nil
Sets the default proc to be executed on each failed key lookup.

```
h.default_proc = proc do |hash, key|
```

```
        hash[key] = key + key
end
h[2]
h["cat"]
```

delete(key) $\rightarrow$ value
delete(key) \{| key | block \} $\rightarrow$ 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.

```
h = { "a" => 100, "b" => 200 }
h.delete("a")
h.delete("z")
h.delete("z") { |el| "#{el} not found" }
```

delete_if \{| key, value | block \} $\rightarrow$ hsh delete_if $\rightarrow$ an_enumerator
Deletes every key-value pair from hsh for which block evaluates to true.

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

```
h = { "a" => 100, "b" => 200, "c" => 300 }
h.delete_if {|key, value| key >= "b" }
```

each \{| key, value | block \} $\rightarrow$ hsh
each_pair \{| key, value | block \} $\rightarrow$ hsh
seach $\rightarrow$ an_enumerator
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.

```
h = { "a" => 100, "b" => 200 }
h.each {|key, value| puts "#{key} is #{value}" }
```

produces:
a is 100
b is 200
each_key \{| key | block \} $\rightarrow$ hsh
each_key $\rightarrow$ 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.

```
h = { "a" => 100, "b" => 200 }
h.each_key {|key| puts key }
```

produces:

## a <br> b

each $\{\mid$ key, value | block $\} \rightarrow$ hsheach_pair \{| key, value | block \} $\rightarrow$ hsh
each $\rightarrow$ an_enumerator
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.

```
h = { "a" => 100, "b" => 200 }
h.each {|key, value| puts "#{key} is #{value}" }
```

produces:

```
a is 100
b is 200
```


## each_value \{| value | block \} $\rightarrow$ hsh

 each_value $\rightarrow$ an_enumeratorCalls 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:

```
1 0 0
```

200
empty? $\rightarrow$ true or false
Returns true if hsh contains no key-value pairs.

```
{}.empty?
```

eql?(other) $\rightarrow$ true or false
Returns true if hash and other are both hashes with the same content.
fetch(key [, default] ) $\rightarrow$ obj
fetch(key) \{| key | block \} $\rightarrow$ 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
```


## flatten $\rightarrow$ an_array

flatten(level) $\rightarrow$ 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 Array\#flatten, this method does not flatten recursively by default. The optional level argument determines the level of recursion to flatten.

has_key?(key) $\rightarrow$ true or false
$\rightarrow$ include?(key) $\rightarrow$ true or false
key?(key) $\rightarrow$ true or false
member?(key) $\rightarrow$ true or false
Returns true if the given key is present in hsh.

```
h = { "a" => 100, "b" => 200 }
h.has_key?("a")
h.has_key?("z")
```

has_value?(value) $\rightarrow$ true or false value?(value) $\rightarrow$ true or false
Returns true if the given value is present for some key in hsh.

```
h = { "a" => 100, "b" => 200 }
h.has_value?(100)
h.has_value?(999)
```

hash $\rightarrow$ fixnum
Compute a hash-code for this hash. Two hashes with the same content will have the same hash code (and will compare using eql?).

See also Object\#hash.
has_key?(key) $\rightarrow$ true or false
include? (key) $\rightarrow$ true or false
key?(key) $\rightarrow$ true or false
member?(key) $\rightarrow$ true or false
Returns true if the given key is present in hsh.

```
h = { "a" => 100, "b" => 200 }
```

h.has_key?("a")

## to_s $\rightarrow$ string

inspect $\rightarrow$ string
Return the contents of this hash as a string.


Also aliased as: to_s

## invert $\rightarrow$ new_hash

Returns a new hash created by using hsh's values as keys, and the keys as values.

keep_if \{| key, value | block \} $\rightarrow$ hsh keep_if $\rightarrow$ 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.

## key(value) $\rightarrow$ key

Returns the key of an occurrence of a given value. If the value is not found, returns nil.

```
h = { "a" => 100, "b" => 200, "c" => 300, "d" =>
h.key(200)
h.key(300)
h.key(999)
```

has_key?(key) $\rightarrow$ true or false
include?(key) $\rightarrow$ true or false
key?(key) $\rightarrow$ true or false
member? (key) $\rightarrow$ true or false
Returns true if the given key is present in hsh.

```
h = { "a" => 100, "b" => 200 }
h.has_key?("a")
h.has_key?("z")
```

keys $\rightarrow$ array
Returns a new array populated with the keys from this hash. See also Hash\#values.

length $\rightarrow$ fixnum
size $\rightarrow$ fixnum
Returns the number of key-value pairs in the hash.

$\pm$ -
has_key?(key) $\rightarrow$ true or false
include?(key) $\rightarrow$ true or false
key?(key) $\rightarrow$ true or false
member?(key) $\rightarrow$ true or false
Returns true if the given key is present in hsh.

```
h = { "a" => 100, "b" => 200 }
h.has_key?("a")
h.has_key?("z")
```


## merge(other_hash) $\rightarrow$ new_hash

 merge(other_hash)\{|key, oldval, newval| block\} $\rightarrow$ new_hashReturns 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) $\rightarrow$ hsh
update(other_hash) $\rightarrow$ hsh
merge!(other_hash)\{|key, oldval, newval| block\} $\rightarrow$ hsh
update(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.
h1 = \{ "a" => 100, "b" => 200 \}

```
h2 = { "b" => 254, "c" => 300 }
h1.merge!(h2) ##> {"a"=>100, "b"=>254
h1 = { "a" => 100, "b" => 200 }
h2 = { "b" => 254, "c" => 300 }
h1.merge!(h2) { |key, v1, v2| v1 }
```


## rassoc(obj) $\rightarrow$ an_array or nil

Searches through the hash comparing obj with the value using ==. Returns the first key-value pair (twoelement array) that matches. See also Array\#rassoc.

```
a = {1=> "one", 2 => "two", 3 => "three", "ii" =>
a.rassoc("two")
a.rassoc("four") ### ni.1
```

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 $h s h$. If Hash\#rehash is called while an iterator is traversing the hash, an RuntimeError will be raised in the iterator.

reject $\{\mid k e y$, value| block\} $\rightarrow$ a_hash

## reject $\rightarrow$ 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.

reject! \{| key, value | block \} $\rightarrow$ hsh or nil reject! $\rightarrow$ an_enumerator
Equivalent to Hash\#delete_if, but returns nil if no changes were made.

## replace(other_hash) $\rightarrow$ hsh

Replaces the contents of hsh with the contents of other_hash.

```
h = { "a" => 100, "b" => 200 }
h.replace({ "c" => 300, "d" => 400 })
```

select $\{\mid k e y$, value| block\} $\rightarrow$ a_hash select $\rightarrow$ 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.

select! \{| key, value | block \} $\rightarrow$ hsh or nil
select! $\rightarrow$ an_enumerator
Equivalent to Hash\#keep_if, but returns nil if no changes were made.
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.

```
h = { 1 => "a", 2 => "b", 3 => "c" }
h.shift
h
```

length $\rightarrow$ fixnum
size $\rightarrow$ fixnum
Returns the number of key-value pairs in the hash.

hsh[key] = value $\rightarrow$ value
store(key, value) $\rightarrow$ value

## Element Assignment

Associates the value given by value with the key given by key.

```
h = { "a" => 100, "b" => 200 }
h["a"] = 9
h["c"] = 4
h ##> {"a"=>9, "b"=>200, "c"=>4}
h.store("d", 42) #=> 42
h
```

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
h.key(200).equal? b #=> true
```

to_a $\rightarrow$ array
Converts hsh to a nested array of [ key, value ]
arrays.


## to_h $\rightarrow$ hsh or new_hash

Returns self. If called on a subclass of Hash, converts the receiver to a Hash object.
to_hash $\rightarrow$ hsh
Returns self.
to_s()
Alias for: inspect
merge!(other_hash) $\rightarrow$ hsh
update(other_hash) $\rightarrow$ hsh
merge!(other_hash)\{|key, oldval, newval| block\} $\rightarrow$ hsh
update(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.

has_value?(value) $\rightarrow$ true or false
value?(value) $\rightarrow$ true or false
Returns true if the given value is present for some key in hsh.

```
h = { "a" => 100, "b" => 200 }
```

h.has_value?(100)
h.has_value?(999)

## values $\rightarrow$ array

Returns a new array populated with the values from hsh. See also Hash\#keys.

```
h = { "a" => 100, "b" => 200, "c" => 300 }
h.values
```

values_at(key, ...) $\rightarrow$ array
Return an array containing the values associated with the given keys. Also see Hash. select.


Generated by RDoc 3.12.2. Generated with the Darkfish Rdoc Generator 3.

## class IO

The IO 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 Kernel\#open method can create an IO (or File) object for these types of arguments:

- A plain string represents a filename suitable for the underlying operating system.
- A string starting with " $\mid$ " 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 " 1 - " 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 ::new for these options. See Kernel\#open 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:<br>gumby<br>ruby<br>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:



## 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

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".

```
IO.binread("testfile")
IO.binread("testfile",
IO.binread("testfile", 20, 10)
```

binwrite(name, string, [offset] ) $\rightarrow$ fixnum
binwrite(name, string, [offset], open_args )
$\rightarrow$ 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 IO. 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 IO, ::copy_stream doesn't move the current file offset.
for_fd(fd, mode [, opt]) $\rightarrow$ io
Synonym for Io.new.
foreach(name, sep=\$/ [, open_args]) \{|line| block \} $\rightarrow$ nil
foreach(name, limit [, open_args]) \{|line|
block $\} \rightarrow$ nil
foreach(name, sep, limit [, open_args]) \{|line|
block \} $\rightarrow$ nil
foreach(...) $\rightarrow$ 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.

```
new(fd [, mode] [, opt]) \(\rightarrow\) io
```

Returns a new IO 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.
::new 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 IO open mode string, ext_enc is the external encoding for the IO and int_enc is the internal encoding.

## IO Open Mode

Ruby allows the following open modes:
\(\left.$$
\begin{array}{l}\text { "r+" Read-write, starts at beginning of file. } \\
\text { "w" } \begin{array}{l}\text { Write-only, truncates existing file } \\
\text { to zero length or creates a new file for wri }\end{array}
$$ <br>
"w+" Read-write, truncates existing file to zero <br>

or creates a new file for reading and writir\end{array}\right\}\)| Write-only, each write call appends data at |
| :--- |
| Creates a new file for writing if file does |

The following modes must be used separately, and along with one or more of the modes seen above.


When the open mode of original IO 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 Encoding for further details of transcoding on input and output.

If "BOM|UTF-8", "BOM|UTF-16LE" or "BOM|UTF16BE" 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 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 String\#encode for controlling conversion between the external encoding and the internal encoding.

## Example 1

```
fd = IO.sysopen("/dev/tty", "w")
a = IO.new(fd,"w")
$stderr.puts "Hello"
a.puts "World"
```


## Produces:

```
Hello
```

World

## Example 2

```
require 'fcntl'
fd = STDERR.fcntl(Fcntl::F_DUPFD)
io = IO.new(fd, mode: 'w:UTF-16LE', cr_newline:
io.puts "Hello, World!"
fd = STDERR.fcntl(Fcntl::F_DUPFD)
io = IO.new(fd, mode: 'w', cr_newline: true,
    external_encoding: Encoding::UTF_16L
io.puts "Hello, World!"
```

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]) $\rightarrow$ io <br> open(fd, mode="r" [, opt]) \{ |io| block \} $\rightarrow$ 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, $\because:$ open returns the value of the block.
See ::new for a description of the fd, mode and opt parameters.
pipe $\rightarrow$ [read_io, write_io]
pipe(ext_enc) $\rightarrow$ [read_io, write_io]
pipe("ext_enc:int_enc" [, opt]) $\rightarrow$ [read_io,
write_io]
pipe(ext_enc, int_enc [, opt]) $\rightarrow$ [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 io 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 " $\mathrm{A}: \mathrm{B}$ ", 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]) \(\rightarrow\) io
popen([env,] cmd, mode="r" [, opt]) \{|io|
block \} \(\rightarrow\) obj
```

Runs the specified command as a subprocess; the
subprocess's standard input and output will be connected to the returned io object.

The PID of the started process can be obtained by \#pid method.
cmd is a string or an array as follows.


If $c m d$ is a string " - ", then a new instance of Ruby is started as the subprocess.
If $c m d$ 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 IO. The last argument opt qualifies mode.



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 = IO.popen("uname")
p f.readlines
f.close
puts "Parent is #{Process.pid}"
IO.popen("date") { |f| puts f.gets }
IO.popen("-") {|f| $stderr.puts "#{Process.pid}
p $?
IO.popen(%Wsed -e s|^|<foo>| -e s&$&;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 #<IO:fd 3>
21352 is here, f is nil
#<Process::Status: pid 21352 exit 0>
<foo>bar;zot;
```


## read(name, [length [, offset]] [, opt] ) $\rightarrow$ 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 ::new 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:


```
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.

```
a = IO.readlines("testfile")
a[0]
```

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
[, timeout]]]) $\rightarrow$ array or nil
Calls select(2) system call. It monitors given arrays of io objects, waits one or more of io objects ready for reading, are ready for writing, and have pending exceptions respectively, and returns an array that contains arrays of those IO objects. It will return nil if optional timeout value is given and no io 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 opensSL::SSL::SSLSocket buffers some data. Io. select doesn't see the buffer. So io. select 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::SSL::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 OpensSL::SSL::SSLSocket\#readpartial 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.

```
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 io objects that wait until ready for read

## write_array

an array of io objects that wait until ready for write

## error_array

an array of io objects that wait for exceptions

## timeout

a numeric value in second

## Example

```
rp, wp = IO.pipe
mesg = "ping
100.times {
    # 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)
ping
```

sysopen(path, [mode, [perm]]) $\rightarrow$ fixnum Opens the given path, returning the underlying file descriptor as a Fixnum.

```
I0.sysopen("testfile")
```


## try_convert(obj) $\rightarrow$ io or nil

Try to convert obj into an IO, using \#to_io method. Returns converted IO or nil if obj cannot be converted for any reason.

```
IO.try_convert(STDOUT)
STDOUT
IO.try_convert("STDOUT")
require 'zlib'
f = open("/tmp/zz.gz")
z = Zlib::GzipReader.open(f)
IO.try_convert(z)
#=> #<Zli.6
GzipRea
tmp/zz
```

1
write(name, string, [offset] ) $\rightarrow$ fixnum write(name, string, [offset], open_args ) $\rightarrow$ 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.


## Public Instance Methods

ios << obj $\rightarrow$ ios
String Output-Writes obj to ios. obj will be converted to a string using to_s.

```
$stdout << "Hello " << "world!\n"
```


## 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 io 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 Symbol, 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 $\rightarrow$ 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? $\rightarrow$ true or false

Returns true if the underlying file descriptor of ios will be closed automatically at its finalization, otherwise false.

## 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? $\rightarrow$ 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.

## 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 $\rightarrow$ true or false

Sets a close-on-exec flag.

```
f = open("/dev/null")
f.close_on_exec = true
system("cat", "/proc/self/fd/#{f.fileno}")
f.closed?
```



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? $\rightarrow$ true or false

Returns true if ios will be closed on exec.

```
f = open("/dev/null")
f.close_on_exec?
f.close_on_exec = true
f.close_on_exec?
f.close_on_exec = false
f.close_on_exec?
```

close_read $\rightarrow$ 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 IOError if the stream is not duplexed.

```
f = IO.popen("/bin/sh","r+")
f.close_read
f.readlines
```

produces:


## close_write $\rightarrow$ 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 IoError if the stream is not duplexed.

```
f = IO.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
```


## closed? $\rightarrow$ true or false

Returns true if ios is completely closed (for duplex streams, both reader and writer), false otherwise.

```
f = File.new("testfile")
f.close
f.closed?
f = IO.popen("/bin/sh","r+")
f.close_write
f.closed?
f.close_read
f.closed?
```


## codepoints()

This is a deprecated alias for each_codepoint.
each(sep=\$/) \{|line| block \} $\rightarrow$ ios
each(limit) \{|line| block \} $\rightarrow$ ios
each(sep,limit) \{|line| block \} $\rightarrow$ ios
each(...) $\rightarrow$ an_enumerator
each_line(sep=\$/) \{|line| block \} $\rightarrow$ ios
each_line(limit) \{|line| block \} $\rightarrow$ ios
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 IoError 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 \} \(\rightarrow\) ios
each_byte \(\rightarrow\) 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 \(\} \rightarrow\) ios
each_char \(\rightarrow\) 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.

each_codepoint \{|c| block \} \(\rightarrow\) ios
codepoints \(\{|c|\) block \(\} \rightarrow\) ios
each_codepoint \(\rightarrow\) an_enumerator
codepoints \(\rightarrow\) 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.
each(sep=\$/) \(\{\mid\) line| block \(\} \rightarrow\) ios
each(limit) \{|line| block \} \(\rightarrow\) ios
each(sep,limit) \{|line| block \} \(\rightarrow\) ios
each(...) \(\rightarrow\) an_enumerator
each_line(sep=\$/) \{|line| block \} \(\rightarrow\) ios

\section*{each_line(limit) \(\{\mid\) line| block \(\} \rightarrow\) ios} each_line(sep,limit) \(\{\) lline| 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 IOError 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..

```

\section*{eof \(\rightarrow\) true or false}
eof? \(\rightarrow\) 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

```

If ios is a stream such as pipe or socket, Io\#eof? blocks until the other end sends some data or closes it.
```

r, w = IO.pipe
Thread.new { sleep 1; w.close }
r.eof?
r, w = IO.pipe
Thread.new { sleep 1; w.puts "a" }

```
```

r.eof?
r, w = IO.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).

\section*{eof \(\rightarrow\) true or false}
eof? \(\rightarrow\) 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

```

If ios is a stream such as pipe or socket, Io\#eof? blocks until the other end sends some data or closes it.
```

r, w = IO.pipe
Thread.new { sleep 1; w.close }
r.eof? \#=> true after 1 second blocking
r, w = IO.pipe
Thread.new { sleep 1; w.puts "a" }
r.eof? \#=> false after 1 second blocking
r, w = IO.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).

\section*{external_encoding \(\rightarrow\) encoding}

Returns the Encoding object that represents the encoding of the file. If io is write mode and no encoding is specified, returns nil.
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 fantl(2) for details. Not implemented on all platforms.

\section*{fdatasync \(\rightarrow \mathbf{0}\) 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).
fileno \(\rightarrow\) fixnum
to_i \(\rightarrow\) fixnum
Returns an integer representing the numeric file descriptor for ios.
\$stdin.fileno
\$stdout.fileno
Also aliased as: to_i
flush \(\rightarrow\) 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

\section*{fsync \(\rightarrow \mathbf{0}\) 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).

\section*{getbyte \(\rightarrow\) 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
f.getbyte

```

\section*{getc \(\rightarrow\) string or nil}

Reads a one-character string from ios. Returns nil if called at end of file.
```

f = File.new("testfile")
f.getc
f.getc

```
gets(sep=\$/) \(\rightarrow\) string or nil gets(limit) \(\rightarrow\) string or nil
gets(sep, limit) \(\rightarrow\) 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 IOError 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.


\section*{inspect \(\rightarrow\) string}

Return a string describing this IO object.

\section*{internal_encoding \(\rightarrow\) encoding}

Returns the Encoding of the internal string if conversion is specified. Otherwise returns nil.

\section*{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 ioctl(2) for details. Not
implemented on all platforms.
isatty \(\rightarrow\) true or false
tty? \(\rightarrow\) true or false
Returns true if ios is associated with a terminal device (tty), false otherwise.
```

File.new("testfile").isatty
File.new("/dev/tty").isatty

```

\section*{lineno \(\rightarrow\) integer}

Returns the current line number in ios. The stream must be opened for reading. lineno counts the number of times gets is called rather than the number of newlines encountered. The two values will differ if gets is called with a separator other than newline.
Methods that use \$/ like each, lines and readline will also increment lineno.

See also the \$. variable.
```

f = File.new("testfile")
f.lineno
f.gets
f.lineno
f.gets
f.lineno

```

\section*{lineno \(=\) integer \(\rightarrow\) integer}

Manually sets the current line number to the given value. \$. is updated only on the next read.
```

f = File.new("testfile")
f.gets
\$.
f.lineno = 1000

```


\section*{lines(*args)}

This is a deprecated alias for each_line.

\section*{pid \(\rightarrow\) 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

```

\section*{pos \(\rightarrow\) integer}
tell \(\rightarrow\) integer
Returns the current offset (in bytes) of ios.
```

f = File.new("testfile")
f.pos
f.gets
f.pos \#=> 17

```

\section*{pos = integer \(\rightarrow\) 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

```
print() \(\rightarrow\) nil
print(obj, ...) \(\rightarrow\) nil
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.

```

\section*{printf(format_string [, obj, ...]) \(\rightarrow\) nil}

Formats and writes to ios, converting parameters under control of the format string. See Kernel\#sprintf for details.

\section*{putc(obj) \(\rightarrow\) 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.
```

\$stdout.putc "A"
\$stdout.putc 65

```
produces:

\section*{AA}

\section*{puts(obj, ...) \(\rightarrow\) 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:
```

this
is
a
test

```

\section*{read([length [, outbuf]]) \(\rightarrow\) string, outbuf, or \\ 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 String, 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.


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.

\section*{read_nonblock(maxlen) \(\rightarrow\) string}
read_nonblock(maxlen, outbuf) \(\rightarrow\) 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 String, 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 IO::WaitReadable. So IO::WaitReadable can be used to rescue the exceptions for retrying read_nonblock.

\section*{\#read_nonblock causes EOFError on EOF.}

If the read byte buffer is not empty, \#read_nonblock reads from the buffer like readpartial. In this case, the read(2) system call is not called.
When \#read_nonblock raises an exception kind of IO::WaitReadable, \#read_nonblock 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 IO::WaitReadable
IO.select([io])
retry
end

```

Although \#read_nonblock 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.

\section*{readbyte \(\rightarrow\) fixnum}

Reads a byte as with Io\#getbyte, but raises an EOFError on end of file.

\section*{readchar \(\rightarrow\) string}

Reads a one-character string from ios. Raises an EOFError on end of file.
```

f = File.new("testfile")
f.readchar
f.readchar

```
readline(sep=\$/) \(\rightarrow\) string
(9)
readline(limit) \(\rightarrow\) string
seadline(sep, limit) \(\rightarrow\) string

Reads a line as with Io\#gets, but raises an EOFError on end of file.
(3) readlines(sep=\$/) \(\rightarrow\) array
readlines(limit) \(\rightarrow\) array
readlines(sep, limit) \(\rightarrow\) array
Reads all of the lines in ios, and returns them in anArray. 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]

```

\section*{readpartial(maxlen) \(\rightarrow\) string readpartial(maxlen, outbuf) \(\rightarrow\) 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 String, 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.
© the byte buffer in the IO object is empty.
- the content of the stream is empty.
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.
\begin{tabular}{|c|c|c|}
\hline r, w = IO.pipe & H & buffer \\
\hline w << "abc" & \# & \\
\hline r.readpartial(4096) & \multicolumn{2}{|l|}{\multirow[t]{2}{*}{}} \\
\hline r.readpartial(4096) & & \\
\hline r, w = IO.pipe & \# & buffer \\
\hline w << "abc" & \# & \\
\hline W.close & \# & \\
\hline r.readpartial(4096) & \multicolumn{2}{|l|}{\multirow[t]{2}{*}{\# raises EOFError}} \\
\hline r.readpartial(4096) & & \\
\hline r, w = IO.pipe & \# & buffer \\
\hline w << "abc\ndef\n" & \# & \\
\hline \(r\).gets & \#=> "abc \(\backslash\) " & "def\n" \\
\hline w << "ghi\n" & \# & "def\n" \\
\hline r.readpartial(4096) & \#=> "def\n" & \\
\hline r.readpartial(4096) & \#\#> "ghivn" & "11 \\
\hline
\end{tabular}

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 IO (IOError)".
- 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 \#sysread causes Errno::EWOULDBLOCK as if the fd is blocking mode.
```

reopen(other_IO) $\rightarrow$ ios
reopen(path, mode_str) $\rightarrow$ 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.
```

f1 = File.new("testfile")
f2 = File.new("testfile")
f2.readlines[0]
f2.reopen(f1)
f2.readlines[0]

```

\section*{rewind \(\rightarrow 0\)}

Positions ios to the beginning of input, resetting
lineno to zero.
```

f = File.new("testfile")
f.readline
f.rewind
f.lineno
f.readline

```

Note that it cannot be used with streams such as pipes, ttys, and sockets.

\section*{seek(amount, whence=IO::SEEK_SET) \(\rightarrow 0\)}

Seeks to a given offset anInteger in the stream according to the value of whence:
\begin{tabular}{|c|c|}
\hline :CUR or IO::SEEK_CUR & Seeks to _amount_ plus cl \\
\hline :END or IO::SEEK_END & Seeks to _amount_ plus el probably want a negative \\
\hline
\end{tabular}
```

:SET or IO::SEEK_SET | Seeks to the absolute lod

```

Example:
```

f = File.new("testfile")
f.seek(-13, IO::SEEK_END)
f.readline

```
set_encoding(ext_enc) \(\rightarrow\) io
set_encoding("ext_enc:int_enc") \(\rightarrow\) io set_encoding(ext_enc, int_enc) \(\rightarrow\) io set_encoding("ext_enc:int_enc", opt) \(\rightarrow\) io set_encoding(ext_enc, int_enc, opt) \(\rightarrow\) 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.

\section*{stat \(\rightarrow\) stat}

Returns status information for ios as an object of type File::Stat.
```

f = File.new("testfile")
s = f.stat
"%o" % s.mode
s.blksize
s.atime

```

\section*{sync \(\rightarrow\) 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

```

\section*{sync \(=\) boolean \(\rightarrow\) 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)

\section*{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 String, 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)

```

\section*{sysseek(offset, whence=IO::SEEK_SET) \(\rightarrow\) 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.
```

f = File.new("testfile")
f.sysseek(-13, IO::SEEK_END)
f.sysread(10)

```

\section*{syswrite(string) \(\rightarrow\) 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.
```

f = File.new("out", "w")
f.syswrite("ABCDEF")

```
pos \(\rightarrow\) integer
tell \(\rightarrow\) integer

Returns the current offset (in bytes) of ios.
```

f = File.new("testfile")
f.pos
f.gets
f.pos

```
to_i()
Alias for: fileno
to_io \(\rightarrow\) ios
Returns ios.

\section*{isatty \(\rightarrow\) true or false}
tty? \(\rightarrow\) true or false
Returns true if ios is associated with a terminal device (tty), false otherwise.

\author{
File.new("testfile").isatty \\ File.new("/dev/tty").isatty
}

\section*{ungetbyte(string) \(\rightarrow\) nil \\ ungetbyte(integer) \(\rightarrow\) 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).
```

f = File.new("testfile")
b = f.getbyte
f.ungetbyte(b)
f.getbyte

```

\section*{ungetc(string) \(\rightarrow\) 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 = File.new("testfile")
c = f.getc
f.ungetc(c)

```

\section*{write(string) \(\rightarrow\) 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) \(\rightarrow\) integer write_nonblock(string [, options]) \(\rightarrow\) 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.
\#write_nonblock 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 IO::WaitWritable. So IO::WaitWritable can be used to rescue the exceptions for retrying write_nonblock.


If the write buffer is not empty, it is flushed at first.
When \#write_nonblock raises an exception kind of IO::WaitWritable, \#write_nonblock 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 IO::WaitWritable, Errno::EINTR
IO.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, \#write_nonblock is not supported according to the kind of the IO object. In such cases, \#write_nonblock raises Errno::EBADF.

By specifying `exception: false`, the options hash allows you to indicate that \#write_nonblock should not raise an IO::WaitWritable exception, but return the symbol :wait_writable instead.

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{class \\ IO::EAGAINWaitReadable}

\section*{In Files}
file.c

\section*{Parent}
rb_eEAGAIN

\section*{Included Modules}
- IO::WaitReadable

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\title{
class \\ IO::EAGAINWaitWritable
}

\section*{In Files}
file.c

\section*{Parent}
rb_eEAGAIN

\section*{Included Modules}
-IO::WaitWritable
Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\title{
class \\ IO::EINPROGRESSWaitRead
}

\section*{In Files}
file.c

\section*{Parent}
rb_eEINPROGRESS

\section*{Included Modules}
- IO::WaitReadable

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\title{
class \\ IO::EINPROGRESSWaitWrita
}

\section*{In Files}

\author{
file.c
}

\section*{Parent}
rb_eEINPROGRESS

\section*{Included Modules}
- IO::WaitWritable

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\title{
class \\ IO::EWOULDBLOCKWaitRea
}

\section*{In Files}
file.c

\section*{Parent}
rb_eEWOULDBLOCK

\section*{Included Modules}
- \(10:\) :WaitReadable

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\title{
class \\ IO::EWOULDBLOCKWaitWri'
}

\section*{In Files}

\author{
file.c
}

\section*{Parent}
rb_eEWOULDBLOCK

\section*{Included Modules}
- IO::WaitWritable

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\title{
module IO::WaitReadable
}

\author{
In Files \\ file.c
}

Generated by RDoc 3.12.2. Generated with the Darkfish Rdoc Generator 3.

\title{
module IO::WaitWritable
}

\section*{In Files}
file.c

Generated by RDoc 3.12.2. Generated with the Darkfish Rdoc Generator 3.

\section*{class IOError}

Raised when an IO operation fails.
File.open("/etc/hosts") \{|f| f << "example"\}
\#=> IoError: not opened for writing
File.open("/etc/hosts") \{|f| f.close; f.read
\# IoError: closed stream

Note that some IO failures raise + SystemCallError+s and these are not subclasses of IOError:


\section*{In Files}
io.c

\section*{Parent}

\section*{StandardError}

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{class IndexError}

Raised when the given index is invalid.


\section*{In Files}
error.c

\section*{Parent}

\section*{StandardError}

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{class Integer}

This class is the basis for the two concrete classes that hold whole numbers, Bignum and Fixnum.

\section*{In Files}
numeric.c
- rational.c

\section*{Parent}

\section*{Numeric}

\section*{Public Instance Methods}
to_i \(\rightarrow\) integer
As int is already an Integer, all these methods simply return the receiver.

Synonyms are to_int, floor, ceil, truncate.
\(\operatorname{chr}([\) encoding]) \(\rightarrow\) string
Returns a string containing the character represented by the int's value according to encoding.
```

65.chr
230.chr
255.chr(Encoding::UTF_8)

```
denominator \(\rightarrow 1\)
Returns 1.
downto(limit) \(\{|i|\) block \(\} \rightarrow\) self
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.
```

5.downto(1) { |n| print n, ".. " } print " Liftoff!\n"

```

\section*{even? \(\rightarrow\) true or false}

Returns true if int is an even number.
to_i \(\rightarrow\) integer
As int is already an Integer, all these methods simply return the receiver.

Synonyms are to_int, floor, ceil, truncate.
gcd(int2) \(\rightarrow\) integer
Returns the greatest common divisor (always positive). \(0 . \operatorname{gcd}(\mathrm{x})\) and \(\mathrm{x} . \operatorname{gcd}(0)\) return \(\operatorname{abs}(\mathrm{x})\).
```

2.gcd(2)
3.gcd(-7)
((1<<31)-1).gcd((1<<61)-1)

```

\section*{gcdlcm(int2) \(\rightarrow\) array}

Returns an array; [int.gcd(int2), int.Icm(int2)].

integer? \(\rightarrow\) true
Since int is already an Integer, this always returns true.

Icm(int2) \(\rightarrow\) integer
Returns the least common multiple (always positive). \(0 . \operatorname{lcm}(x)\) and \(x . I c m(0)\) return zero.

next \(\rightarrow\) integer
succ \(\rightarrow\) integer
Returns the Integer equal to int + 1, same as \#next.
```

1.next
(-1).next

```
8) numerator \(\rightarrow\) self

Returns self.
odd? \(\rightarrow\) true or false
Returns true if int is an odd number.
ord \(\rightarrow\) self

Returns the int itself.
a.ord

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.
( pred \(\rightarrow\) integer
Returns the Integer equal to int - 1 .
```

1.pred
(-1).pred

```
sationalize([eps]) \(\rightarrow\) rational
Returns the value as a rational. The optional argument eps is always ignored.
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.
```

1. round
2. round(2)
3. round(-1)
```
next \(\rightarrow\) integer
succ \(\rightarrow\) integer
Returns the Integer equal to int + 1, same as \#next.
```

1.next

```

Iterates the given block int times, passing in values from zero to int - 1.

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

5.times do |i|
print i, " "
end

```
to_i \(\rightarrow\) integer
As int is already an Integer, all these methods simply
return the receiver.

Synonyms are to_int, floor, ceil, truncate.
to_i \(\rightarrow\) integer
As int is already an Integer, all these methods simply return the receiver.

Synonyms are to_int, floor, ceil, truncate.

\section*{to_r \(\rightarrow\) rational}

Returns the value as a rational.
```

1.to_r
(1<<64).to_r
(1/1)

```
to_i \(\rightarrow\) integer
As int is already an Integer, all these methods simply
return the receiver.
Synonyms are to_int, floor, ceil, truncate.

\section*{upto(limit) \(\{|i|\) block \(\} \rightarrow\) self \\ upto(limit) \(\rightarrow\) 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:


Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{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.
begin
puts "Press ctrl-C when you get bored"
loop \(\}\)
rescue Interrupt => e
puts "Note: You will typically use Signal.t
end
produces:

Press ctrl-C when you get bored
then waits until it is interrupted with Control-C and then prints:

Note: You will typically use Signal.trap inst

\section*{In Files}
- error.c
- signal.c

\section*{Parent}

SignalException
Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{module Kernel}

The Kernel module is included by class Object, 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:

\section*{sprintf "\%.1f", 1.234}

\section*{In Files}
complex.c
cont.c
- error.c
\(\square\) eval.c
- eval_jump.c
\(\bigcirc\) file.c
- io.c
\(\square\) load.c
- object.c
- proc.c
- process.c
@ random.c
- rational.c
- ruby.c
- signal.c
- vm_backtrace.cvm_eval.cvm_trace.c

\section*{Public Instance Methods}

\section*{Array(arg) \(\rightarrow\) array}

Returns arg as an Array.
First tries to call to_ary on arg, then to_a.
```

Array(1..5)

```

\section*{Complex(x[, y]) \(\rightarrow\) numeric}

Returns \(x+i^{*} y\);
```

Complex(1, 2)
Complex('1+2i')
Complex(nil)
Complex(1, nil) \#=> TypeError

```

\section*{Syntax of string form:}
```

string form = extra spaces , complex , extra spad
complex = real part | [ sign ] , imaginary part
| real part , sign , imaginary part
| rational , "@" , rational ;
real part = rational
imaginary part = imaginary unit | unsigned ratior
rational = [ sign ] , unsigned rational ;
unsigned rational = numerator | numerator , "/"
numerator = integer part | fractional part | int\epsilon
denominator = digits ;
integer part = digits ;
fractional part = "." , digits , [ ( "e" | "E" )
imaginary unit = "i" | "I" | "j" | "J" ;
sign = "-" | "+"
digits = digit , { digit | "_" , digit };
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6"
extra spaces = ? \s* ? ;

```

See String\#to_c.

\section*{Float(arg) \(\rightarrow\) float}

Returns arg converted to a float. Numeric types are converted directly, the rest are converted using arg.to_f. Converting nil generates a TypeError.
```

Float(1)
Float("123.456")

```

\section*{Hash(arg) \(\rightarrow\) hash}

Converts arg to a Hash by calling arg.to_hash.
Returns an empty Hash when arg is nil or [].
```

Hash([])
Hash(nil)
Hash(key: :value)
Hash([1, 2, 3])

```

\section*{Integer(arg, base=0) \(\rightarrow\) integer}

Converts arg to a Fixnum or Bignum. Numeric 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 ( \(0,0 \mathrm{~b}\), and 0 x ) 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)
Integer("0x1a")
Integer(Time.new)
Integer("0930", 10)

```
```

Integer("111", 2)
Integer(nil)

```

Rational \((x[, y]) \rightarrow\) numeric
Returns x/y;
```

Rational(1, 2)
Rational('1/2')
Rational(nil)
Rational(1, nil) \#\#>> TypeError

```

\section*{Syntax of string form:}
```

string form = extra spaces , rational , extra spa
rational = [ sign ] , unsigned rational ;
unsigned rational = numerator | numerator , "/"
numerator = integer part | fractional part | int\epsilon
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 String\#to_r.

\section*{String(arg) \(\rightarrow\) string}

Returns arg as a string.
First tries to call its to_str method, then its to_s method.
```

String(self)
String(self.class)
String(123456)

Symbol. If called outside of a method, it returns nil.

## 8) __dir__ $\rightarrow$ 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 $\qquad$ FILE is nil, it returns nil. The return value equals to File.dirname(File.realpath(__FILE__)).

## 8) __method <br> $\qquad$ $\rightarrow$ symbol

Returns the name at the definition of the current method as a Symbol. If called outside of a method, it returns nil.

## `cmd` $\rightarrow$ 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 $m s g$ is given, it is written to STDERR prior to terminating.

## at_exit \{ block \} $\rightarrow$ 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
```


## 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.


## autoload?(name) $\rightarrow$ String or nil

Returns filename to be loaded if name is registered as autoload.

```
autoload(:B, "b")
autoload?(:B)
```


## 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.

```
def get_binding(param)
    return binding
end
b = get_binding("hello")
eval("param", b)
```

block_given? $\rightarrow$ true or false
iterator? $\rightarrow$ true or false

Returns true if yield would execute a block in the current context. The iterator? form is mildly deprecated.

```
def try
    if block_given?
        yield
    else
        "no block"
    end
end
try
try { "hello" }
try do "hello" end ##> "hel.1o"
```

callcc $\{\mid$ cont| block $\} \rightarrow$ obj
Generates a Continuation object, which it passes to the associated block. You need to require
'continuation' before using this method. Performing a cont. call will cause the callcc to return (as will falling through the end of the block). The value returned by the callcc is the value of the block, or the value passed to cont.call. See class Continuation for more details. Also see \#throw for an alternative mechanism for unwinding a call stack.
caller(start=1, length=nil) $\rightarrow$ array or nil caller(range) $\rightarrow$ 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.

caller_locations(start=1, length=nil) $\rightarrow$ array or nil
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.

## catch([tag]) $\{|t a g|$ 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 }
```

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) }
catch(1) { throw(1) }
```

When tag is passed as the first argument, catch yields it as the parameter of the block.

```
catch(1) {|x| x + 2 }
```

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
# => 456
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
# => 123
```


## chomp $\rightarrow$ \$

chomp(string) $\rightarrow$ \$
Equivalent to \$_ = \$_.chomp(string). See String\#chomp. Available only when -p/-n command line option specified.
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]]]) $\rightarrow$ 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, $\operatorname{arg1}, \ldots$ )
command name and one or more
arguments (no shell)
exec([cmdname, argv0], arg1, ...)
command name, $\frac{\text { argv }}{\text { arguments (no shell) }}$

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
— 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 IO\#pid for IO.popen) 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", "argve"], "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 \#system if this is not acceptable.

| exec "echo *" <br> \# never get here |
| :--- |
| exec "echo", "*" \# echoes list of files in cur |
| \# never get here asterisk |

## 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).

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.

```
Process.exit!(true)
```


## 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

## fork [\{ block \}] $\rightarrow$ fixnum or nil fork [\{ block \}] $\rightarrow$ 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().

## format(format_string [, arguments...] ) $\rightarrow$ string <br> sprintf(format_string [, arguments...] ) $\rightarrow$ 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:


| e | Convert floating point argument into exp with one digit before the decimal point The precision specifies the number of di point (defaulting to six). |
| :---: | :---: |
| E | Equivalent to 'e', but uses an uppercase the exponent. |
| $f$ | Convert floating point argument as [-]dd where the precision specifies the number the decimal point. |
| g | Convert a floating point number using ex if the exponent is less than -4 or great equal to the precision, or in dd.dddd fo The precision specifies the number of si |
| G | Equivalent to 'g', but use an uppercase |
| a | Convert floating point argument as [-]0x which is consisted from optional sign, as hexadecimal, "p", and exponential par |
| A | Equivalent to 'a', but use uppercase ' $\mathrm{X}^{\prime}$ |
| Field | Other Format |
| C | Argument is the numeric code for a singl a single character string itself. |
| p | The valuing of argument.inspect. |
| S | Argument is a string to be substituted. sequence contains a precision, at most tl will be copied. |
| \% | A percent sign itself will be displayed. |

The flags modifies the behavior of the formats. The flag characters are:

| Flag | Applies to | Meaning |
| :---: | :---: | :---: |
| space | bBdiouxX | Leave a space at the |
|  | aAeEfgG | non-negative numbers. |
|  | (numeric fmt) | For ${ }^{\circ} \mathrm{o}$ ', `\(\mathrm{x}^{\prime}\),` $\mathrm{X}^{\prime}$, `b |
|  |  | a minus sign with absc |
|  |  | negative values |
| (digit)\$ | all | Specifies the absolut |
|  |  | for this field. Abso. |
|  |  | argument numbers cannc |
|  |  | sprintf string. |
| \# | bBoxX | Use an alternative for |

|  | aAeEfgG | For the conversions until the first digit it is not formatted as For the conversions on non-zero, prefix tl `0X'', `b'' and <br> For `a', `A', `e', `E force a decimal point even if no digits foll For 'g' and 'G', do nc |
| :---: | :---: | :---: |
| + | bBdiouxX <br> aAeEfgG <br> (numeric fmt) | Add a leading plus si numbers. <br> For `o', `x', `X', `b a minus sign with absc negative values. |
| - | all | Left-justify the resul |
|  | ```bBdiouxX aAeEfgG (numeric fmt)``` | Pad with zeros, not sr For `o', `x', ‘X', `b is used for negative complements. |
| * | all | Use the next argument If negative, left-jus asterisk is followed sign, use the indicat |
| 4 |  | , |

## Examples of flags:



```
sprintf("%x", 123)
sprintf("%#x", 123)
sprintf("%+x", -123) ##> "-7b"
sprintf("%x", -123) #=> " . .f85
sprintf("%#x", -123) ### "0x. .f85
sprintf("%#x", 0)
    for `X' uses the prefix `0X
sprintf("%X", 123) #=> "7B"
sprintf("%#X", 123) ##> "0x7B
    flag for `b" add a prefix `0b' for non-zer
    and space flag disables complements for ne
sprintf("%b", 123) #=> "11111011
sprintf("%#b", 123) #=> "0b11111011
sprintf("%+b", -123) ##> "-11111011"
sprintf("%b", -123) #=> "..10000101'
sprintf("%#b", -123) #=> "0b..10000101'
sprintf("%#b", 0)
    for 'B' uses the prefix `0B
sprintf("%B", 123) #=> "1111011
sprintf("%#B", 123) #=> "0B1111011"
    for `e' forces to show the decimal point
sprintf("%.0e", 1) #=> "1e+00'
sprintf("%#.0e", 1) #=> "1.e+00
sprintf("%.0f", 1234) #=> "1234
sprintf("%#.0f", 1234) ##> "1234
    for 'g' forces to show the decimal point
# It also disables stripping lowest zeros.
sprintf("%g", 123.4) ##> "123.4"
sprintf("%#g", 123.4) #=> "123.400"
sprintf("%g", 123456) ##> "123456
sprintf("%#g", 123456) ##=> "123456
```

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:



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:




## Examples:

```
sprintf("%d %04x", 123, 123)
sprintf("%08b '%4s'", 123, 123)
sprintf("%1$*2$s %2$d %1$s", "hello", 8)
sprintf("%1$*2$s %2$d", "hello", -8)
sprintf("%+g:% g:%-g", 1.23, 1.23, 1.23)
sprintf("%u", -123)
```



For more complex formatting, Ruby supports a reference by name. \%<name>s style uses format style, but \%\{name\} style doesn't.

## Examples:

```
sprintf("%<foo>d : %<bar>f", { :foo => 1, :bar =>
sprintf("%{foo}f", { :foo => 1 })
```

gets(sep=\$/) $\rightarrow$ string or nil gets(limit) $\rightarrow$ string or nil gets(sep,limit) $\rightarrow$ 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
```

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.

## global_variables $\rightarrow$ array

Returns an array of the names of global variables.

gsub(pattern, replacement) $\rightarrow$ \$_ gsub(pattern) \{|...| block \} $\rightarrow$ \$

Equivalent to \$_.gsub.... except that \$_ will be updated if substitution occurs. Available only when $\mathrm{p} /-\mathrm{n}$ command line option specified.

## block_given? $\rightarrow$ true or false <br> iterator? $\rightarrow$ true or false

Returns true if yield would execute a block in the current context. The iterator? form is mildly deprecated.

```
def try
    if block_given?
        yield
    else
        "no block"
    end
end
try
try { "hello" }
try do "hello" end
```


## lambda \{ |...| block \} $\rightarrow$ a_proc

Equivalent to Proc.new, except the resulting Proc objects check the number of parameters passed when called.

## 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.

## local_variables $\rightarrow$ array

Returns the names of the current local variables.

```
fred = 1
for i in 1..10
end
local_variables
```


## loop \{ block \}

loop $\rightarrow$ 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
```

Stoplteration raised in the block breaks the loop.
open(path [, mode [, perm]] [, opt]) $\rightarrow$ io or nil
open(path [, mode [, perm]] [, opt]) \{|io| block
$\rightarrow$ \} $\rightarrow$ obj
Creates an IO 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 IO.new 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 File.new 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 IO object as a parameter, and the IO 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 IO 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 " $\mid-$ "), 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 IO 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 IO object:

```
open "|-" do |f|
    if f then
        # parent process
        puts "Got: #{f.gets}"
    else
        # child process
        puts "in Child"
```

```
    end
end
```

Produces:

Got: in Child
p(obj) $\rightarrow$ obj
$\mathrm{p}(\mathrm{obj} 1, \mathrm{obj} 2, \ldots) \rightarrow[\mathrm{obj}, \ldots]$
p() $\rightarrow$ nil
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, ...) $\rightarrow$ 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"
$, = ","
print "cat", [1,2,3], 99
```

produces:

## cat12399

# printf(io, string [, obj ... ]) $\rightarrow$ nil <br> printf(string [, obj ... ]) $\rightarrow$ nil 

Equivalent to:
io.write(sprintf(string, obj, ...))
or
\$stdout.write(sprintf(string, obj, ...))
proc $\{|. .$.$| block \} \rightarrow$ a_proc
Equivalent to Proc.new.
putc(int) $\rightarrow$ int
Equivalent to:
\$stdout.putc(int)
Refer to the documentation for IO\#putc for important information regarding multi-byte characters.
puts(obj, ...) $\rightarrow$ 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

## 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
```

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)
```

When max is a Range, rand returns a random number where range.member?(number) == true.

Negative or floating point values for max are allowed, but may give surprising results.

```
rand(-100)
rand(-0.5)
rand(1.9)
```

$\square$
\#srand may be used to ensure that sequences of random numbers are reproducible between different runs of a program.

See also Random\#rand.

## readline(sep=\$/) $\rightarrow$ string <br> readline(limit) $\rightarrow$ string <br> readline(sep, limit) $\rightarrow$ string

Equivalent to Kernel::gets, except readline raises EOFError at end of file.

## readlines(sep=\$/) $\rightarrow$ array <br> readlines(limit) $\rightarrow$ array <br> readlines(sep,limit) $\rightarrow$ array

Returns an array containing the lines returned by calling Kernel.gets(sep) until the end of file.

## require(name) $\rightarrow$ true 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.

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.

## 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]]]) $\rightarrow$ array or nil
Calls select(2) system call. It monitors given arrays of io objects, waits one or more of io objects ready for reading, are ready for writing, and have pending exceptions respectively, and returns an array that contains arrays of those IO objects. It will return nil if optional timeout value is given and no io 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 opensSL: :SSL: :SSLSocket buffers some data. Io. select doesn't see the buffer. So io. select 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::SSL::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 OpenSSL::SSL::SSLSocket\#readpartial 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 io objects that wait until ready for read
write_array
an array of io objects that wait until ready for write

## error_array

an array of io objects that wait for exceptions

## timeout

a numeric value in second

## Example

```
rp, wp = IO.pipe
```

```
mesg = "ping
100.times {
    # 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
}
```

produces:

```
ping pong
ping pong
ping pong
(snipped)
ping
```

set_trace_func(proc) $\rightarrow$ proc
set_trace_func(nil) $\rightarrow$ nil

Establishes proc as the handler for tracing, or disables tracing if the parameter is nil.

Note: this method is obsolete, please use TracePoint 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
$a=1$
$\mathrm{b}=2$
end
end
set_trace_func proc \{ |event, file, line, id,
printf "\%8s \%s:\%-2d \%10s \%8s\n", event, fil
\}
t = Test.new
t.test

| c-call prog.rb:11 | new | Class |
| ---: | ---: | ---: |
| c-call prog.rb:11 | initialize | Object |
| c-return prog.rb:11 | initialize | Object |
| 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 |
|  |  |  |

## sleep([duration]) $\rightarrow$ 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.

```
Time.new
sleep 1.2
Time.new
sleep 1.9
Time.new
```


## spawn([env,] command... [,options]) $\rightarrow$ pid

 spawn([env,] command... [,options]) $\rightarrow$ pidspawn executes specified command and return its pid.

```
pid = spawn("tar xf ruby-2.0.0-p195.tar.bz2")
Process.wait pid
pid = spawn(RbConfig.ruby, "-eputs'Hello, world!
Process.wait pid
```

This method is similar to \#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 (de1
    process group:
        :pgroup => true or 0 : make a new process grd
        :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 prc
    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
            string : redirect to fi-
            [string] : redirect to fi]
            [string, open_mode] : redirect to fi]
            [string, open_mode, perm] : redirect to fi]
            [:child, FD] : redirect to the
            :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.

```
# set F00 as BAR and unset BAZ
pid = spawn({"F00"=>"BAR", "BAZ"=>nil}, command)
```

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 IO 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, STDERR=>:out)
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.

```
pid = spawn(command, :in=>["file"])
pid = spawn(command, :in=>["file", "r"])
pid = spawn(command, :out=>["log", "w"])
pid = spawn(command, :out=>["log", "w", 0600])
pid = spawn(command, :out=>["log", File::WRONLY|F
```

4 •


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 IO.popen. In this case, IO. popen 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.

## format(format_string [, arguments...] ) $\rightarrow$ string <br> sprintf(format_string [, arguments...] ) $\rightarrow$ 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:


| i | Identical to `d' \\ \hline \multirow[t]{2}{*}{0} & Convert argument as an octal number \\ \hline & Negative numbers will be displayed as a prefixed with `..7'. |
| :---: | :---: |
| u | Identical to 'd' |
| x | Convert argument as a hexadecimal number |
|  | Negative numbers will be displayed as a |
|  | prefixed with '..f' (representing an infi |
|  | leading 'ff's). |
| X | Equivalent to 'x', but uses uppercase let |
| Field | Float Format |
| e | Convert floating point argument into expd with one digit before the decimal point |
|  | The precision specifies the number of did |
|  | 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 great |
|  | 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 [-]0xt |
|  | which is consisted from optional sign, "C |
|  | as hexadecimal, "p", and exponential part |
| A | Equivalent to 'a', but use uppercase ' $\mathrm{X}^{\prime}$ |
| Field | Other Format |
| c | Argument is the numeric code for a singlє a single character string itself. |
| p | The valuing of argument.inspect |
|  | 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 non-negative numbers. For `o', ‘x', ‘X', `b a minus sign with absc negative values. |
| :---: | :---: | :---: |
| (digit)\$ | all | Specifies the absolut for this field. Abso1 argument numbers cannc sprintf string. |
| \# | bBoxX <br> aAeEfgG | Use an alternative for For the conversions until the first digit it is not formatted as For the conversions on non-zero, prefix 0X'', `\(0 b^{\prime \prime}\) and For`a', `A', `e', `E force a decimal point even if no digits fol. For 'g' and 'G', do nc \\ \hline + & bBdiouxX aAeEfgG (numeric fmt) & \begin{tabular}{l} Add a leading plus sis numbers. \\ For `o', `x', ‘X', `b a minus sign with absc negative values. | <br>

\hline - \& all \& Left-justify the resul <br>

\hline $$
0 \text { (zero) }
$$ \& bBdiouxX aAeEfgG (numeric fmt) \& Pad with zeros, not sp For 'o', ‘x', ‘X', `b is used for negative complements. <br>

\hline * \& all \& Use the next argument If negative, left-jus asterisk is followed sign, use the indicat <br>
\hline
\end{tabular}

## Examples of flags:



```
sprintf("%+d", 123)
sprintf("% d", 123)
    flag for 'o' increases number of digits to
    and space flag changes format of negative
sprintf("%o", 123)
sprintf("%#0", 123)
sprintf("%+0", -123) ##> "-173'
sprintf("%o", -123) ##> "..7605"
sprintf("%#0", -123) ##> " . . 7605"
# `#' flag for ` x' add a prefix ` 0x' for non-zer
    and space flag disables complements for ne
sprintf("%x", 123) ##> "7b"
sprintf("%#x", 123) ##> "0x7b"
sprintf("%+x", -123) ### "-7b"
sprintf("%x", -123) ### " . .f85"
sprintf("%#x", -123) ### "0x. .f85"
sprintf("%#x", 0)
sprintf("%X", 123)
sprintf("%#X", 123) ##> "0x7B"
# `#' flag for `b' add a prefix ` 0b' for non-zerc
sprintf("%b", 123) ##> "1111011"
sprintf("%#b", 123) ##> "0b1111011"
sprintf("%+b", -123) ##> "-1111.011"
sprintf("%b", -123) ##> "..10000101"
sprintf("%#b", -123) ##> "0b...10000101'
sprintf("%#b", 0)
# `#' for `B' uses the prefix `0B
sprintf("%B", 123) ##> "1111.011.
sprintf("%#B", 123) ##> "0B1111011"
sprintf("%.0e", 1) ##> "1e+00"
sprintf("%#.0e", 1) ##> "1.e+00"
    *or 'f' forces to show the decimal point
sprintf("%.0f", 1234) ##> "1234
sprintf("%#.0f", 1234) ##> "1234
```

sprintf("%g", 123.4)
sprintf("%\#g", 123.4) \#\#> "123.400"
sprintf("%g", 123456) \#\#> "123456"
sprintf("%\#g", 123456) \#=> "123456

```

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.

\section*{Examples of width:}


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.)

\section*{Examples of precisions:}



\section*{Examples:}
```

sprintf("%d %04x", 123, 123)
sprintf("%08b '%4s'", 123, 123)
sprintf("%1$*2$s %2$d %1$s", "hello", 8)
sprintf("%1$*2$s %2\$d", "hello", -8)
sprintf("%+g:% g:%-g", 1.23, 1.23, 1.23)
sprintf("%u", -123)
|

```

For more complex formatting, Ruby supports a reference by name. \%<name>s style uses format style, but \%\{name\} style doesn't.

Examples:


\section*{srand(number \(=\) Random.new_seed) \(\rightarrow\) 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.

sub(pattern, replacement) \(\rightarrow\) \$_
sub(pattern) \(\{|. .\).\(| block \} \rightarrow\) _
Equivalent to \$_.sub(args), except that \$_ will be updated if substitution occurs. Available only when \(\mathrm{p} /-\mathrm{n}\) command line option specified.

\section*{syscall(num [, args...]) \(\rightarrow\) integer}

Calls the operating system function identified b) returns the result of the function or raises Syst it failed.

Arguments for the function can follow _num_. Thes +String+ objects or +Integer+ objects. A +Stringas a pointer to the byte sequence. An +Integer+
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>.
\[
\text { syscall 4, 1, "hello\n", } 6 \text { \# '4' is write(2) }
\]
<em>produces:</em>
hello
Calling +syscall+ on a platform which does not he an arbitrary system function just fails with Not.

\section*{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.

\section*{system([env,] command... [,options]) \(\rightarrow\) true, false or nil}

Executes command... in a subshell. command... is one of following forms.
\begin{tabular}{ll}
\begin{tabular}{l} 
commandline \\
cmdname, arg1, \(\ldots\)
\end{tabular} & \begin{tabular}{l} 
: command line string \\
[cmdname, argv0], arg1, \(\ldots\).
\end{tabular} \\
\hline
\end{tabular}
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.

\section*{test(cmd, file1 [, file2] ) \(\rightarrow\) obj}

Uses the character cmd to perform various tests on file1 (first table below) or on file1 and file2 (second table).

\section*{File tests on a single file:}
\begin{tabular}{c|l|l}
\hline 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 de \\
"C" & Time & Last change time for file1 \\
"d" & boolean & True if file1 exists and is a ds \\
"e" & boolean & True if file1 exists \\
"f" & boolean & True if file1 exists and is a r¢ \\
"g" & boolean & True if file1 has the \CF\{setgid \\
& & set (false under NT) \\
"G" & boolean & True if file1 exists and has a \\
"k" & boolean & True if file1 exists and has the \\
"l" & boolean & True if file1 exists and is a s \\
"M" & Time & Last modification time for file \\
"o" & boolean & True if file1 exists and is own
\end{tabular}
\begin{tabular}{l|l|l} 
& & the caller's effective uid \\
\(" 0 "\) & boolean & True if file1 exists and is owne \\
"p" & boolean & The caller's real uid
\end{tabular}

Tests that take two files:
\begin{tabular}{|c|c|c|}
\hline "-" & boolean & True if file1 and file2 are ider \\
\hline "=" & boolean & True if the modification times and file2 are equal \\
\hline "<" & boolean & True if the modification time o is prior to that of file2 \\
\hline ">" & boolean & True if the modification time o is after that of file2 \\
\hline
\end{tabular}

\section*{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.
```

trace_var(symbol, cmd ) -> nil
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'

```

\section*{trap( signal, command ) \(\rightarrow\) obj} 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

## 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.

## warn(msg, ...) $\rightarrow$ 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).


Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## class KeyError

Raised when the specified key is not found. It is a subclass of IndexError.


## In Files

error.c

## Parent

IndexError

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## class LoadError

Raised when a file required (a Ruby script, extension library, ...) fails to load.
require 'this/file/does/not/exist'
raises the exception:


## In Files

error.c

## Parent

## ScriptError

## Attributes

rb_intern_const("path") ${ }^{\text {[R] }}$
the path failed to load

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## 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

## In Files

proc.c

## Parent

StandardError

## Public Instance Methods

exit_value $\rightarrow$ obj
Returns the exit value associated with this LocalJumpError.
reason $\rightarrow$ symbol
The reason this block was terminated: :break, :redo, :retry, :next, :return, or :noreason.

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## 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 $-\mathrm{d},-\mathrm{v},-\mathrm{w}$, or -verbose) the major and minor numbers must match exactly. Marshal 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
str[0].ord
str[1].ord
```

Some objects cannot be dumped: if the objects to be dumped include bindings, procedure or method objects, instances of class IO, 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 Marshal strings.

## Security considerations

By design, $\because:$ load can deserialize almost any class loaded into the Ruby process. In many cases this can lead to remote code execution if the Marshal data is loaded from an untrusted source.

As a result, ::load 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 String, Array, Hash, 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 Integer which indicates the maximum depth of objects to dump (a value of -1 implies that you should disable depth checking). _dump must return a String containing the information necessary to reconstitute the object.

The class method _load should take a String 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 Marshal 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

## dump( obj [, anIO] , limit=-1 ) $\rightarrow$ anIO

Serializes obj and all descendant objects. If anIO is specified, the serialized data will be written to it, otherwise the data will be returned as a String. 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
```

Marshal can't dump following objects:
Banonymous Class/Module.

- objects which are related to system (ex: Dir, File::Stat, IO, File, Socket and so on)
$\square$ an instance of MatchData, Data, Method, UnboundMethod, Proc, Thread, ThreadGroup, Continuation
objects which define singleton methods


## load( source [, proc] ) $\rightarrow$ obj <br> restore( source [, proc] ) $\rightarrow$ 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 IO 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.

## load( source [, proc] ) $\rightarrow$ obj

restore( source [, proc] ) $\rightarrow$ 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 IO 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 RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 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

mtch $==$ mtch $2 \rightarrow$ true or false eql?(mtch2) $\rightarrow$ true or false
Equality-Two matchdata are equal if their target strings, patterns, and matched positions are identical.
mtch[i] $\rightarrow$ str or nil
mtch[start, length] $\rightarrow$ array
mtch[range] $\rightarrow$ array
mtch[name] $\rightarrow$ 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) $\rightarrow$ 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.

```
m = /(.)(.)(\d+)(\d)/.match("THX1138.")
m.begin(0)
m.begin(2)
m = /(?<foo>.)(.)(?<bar>.)/.match("hoge")
p m.begin(:foo)
p m.begin(:bar)
```

captures $\rightarrow$ array
Returns the array of captures; equivalent to mtch.to_a[1..-1].
f1,f2,f3,f4 = /(.)(.)(\d+)(\d)/.match("THX1138."


## 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.

```
m = /(.)(.)(\d+)(\d)/.match("THX1138.")
m. end(0)
m. end(2)
m = /(?<foo>.)(.)(?<bar>.)/.match("hoge")
p m.end(:foo)
p m.end(:bar)
```


## mtch $==$ mtch2 $\rightarrow$ true or false

eql? (mtch2) $\rightarrow$ true or false
Equality-Two matchdata are equal if their target
strings, patterns, and matched positions are identical.

## hash $\rightarrow$ integer

Produce a hash based on the target string, regexp and matched positions of this matchdata.

See also Object\#hash.
inspect $\rightarrow$ str
Returns a printable version of $m t c h$.

```
puts /.$/.match("foo").inspect
```

```
puts /(.)(.)(.)/.match("foo").inspect
puts /(.)(.)?(.)/.match("fo").inspect
#=> #<MatchData "fo" 1:"f" 2:ni. 3:"0
puts /(?<foo>.)(?<bar>.)(?<baz>.)/.match("hoge")
#=> #<MatchData "hog" foo:"h" bar:"o" baz:"g">
```

length $\rightarrow$ integer
size $\rightarrow$ integer
Returns the number of elements in the match array.

```
m = /(.)(.)(\d+)(\d)/.match("THX1138.")
m.length
m.size
```


## names $\rightarrow$ [name1, name2, ...]

Returns a list of names of captures as an array of strings. It is same as mtch.regexp.names.

```
/(?<foo>.)(?<bar>.)(?<baz>.)/.match("hoge").names
#=> ["foo", "bar", "baz".
m =/(?<x>.)(?<y>.)?/.match("a") ##> #<MatchData
m.names
```


## 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.

```
m = /(.)(.)(\d+)(\d)/.match("THX1138.")
m.offset(0)
m.offset(4)
m = /(?<foo>.)(.)(?<bar>.)/.match("hoge")
```

```
p m.offset(:foo)
p m.offset(:bar)
```

post_match $\rightarrow$ str
Returns the portion of the original string after the current match. Equivalent to the special variable \$'.

pre_match $\rightarrow$ str
Returns the portion of the original string before the current match. Equivalent to the special variable $\$$.

```
m = /(.)(.)(\d+)(\d)/.match("THX1138.")
m.pre_match
```

```
regexp -> regexp
```

Returns the regexp.

```
m = /a.*b/.match("abc")
m.regexp
```

length $\rightarrow$ integer
size $\rightarrow$ integer

Returns the number of elements in the match array.

```
m = /(.)(.)(\d+)(\d)/.match("THX1138.")
m.length
m.size
```

string $\rightarrow$ str
Returns a frozen copy of the string passed in to
match.

```
m = /(.)(.)(\d+)(\d)/.match("THX1138.")
m.string
```


## to_a $\rightarrow$ anArray

Returns the array of matches.

```
m = /(.)(.)(\d+)(\d)/.match("THX1138.")
m.to_a #=> ["HX1138", "H", "X", "113",
```

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 $\rightarrow$ str

Returns the entire matched string.

```
m = /(.)(.)(\d+)(\d)/.match("THX1138.")
m.to_s ##> "HX11.38'
```

8. values_at([index]*) $\rightarrow$ array

Uses each index to access the matching values, returning an array of the corresponding matches.


Generated by RDoc 3.12.2. Generated with the Darkfish Rdoc Generator 3.

## module Math

The Math module contains module functions for basic trigonometric and transcendental functions. See class Float 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

## E

Definition of the mathematical constant $\underline{E}$ (e) as a Float number.

## PI

Definition of the mathematical constant PI as a Float number.

## Public Class Methods

```
acos(x) }->\mathrm{ Float
```

Computes the arc cosine of $x$. Returns 0..PI.
Domain: [-1, 1]
Codomain: [0, PI]
Math.acos(0) == Math::PI/2
(8) $\operatorname{acosh}(x) \rightarrow$ Float

Computes the inverse hyperbolic cosine of $x$.
Domain: [1, INFINITY)
Codomain: [0, INFINITY)
Math.acosh(1)
(8) $\operatorname{asin}(x) \rightarrow$ Float

Computes the arc sine of $x$. Returns -PI/2..PI/2.
Domain: [-1, -1]
Codomain: [-PI/2, PI/2]
Math.asin(1) == Math::PI/2
asinh(x) $\rightarrow$ Float
Computes the inverse hyperbolic sine of $x$.
Domain: (-INFINITY, INFINITY)
Codomain: (-INFINITY, INFINITY)
Math.asinh(1)
atan(x) $\rightarrow$ Float
Computes the arc tangent of $x$. Returns - $\mathrm{Pl} / 2$..PI/2.
Domain: (-INFINITY, INFINITY)

Codomain: (-PI/2, PI/2)

```
Math.atan(0)
```


## 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: [-PI, PI]


## atanh(x) $\rightarrow$ Float

Computes the inverse hyperbolic tangent of $x$.
Domain: (-1, 1)
Codomain: (-INFINITY, INFINITY)
Math.atanh(1) \#=> Infinity

## cbrt(x) $\rightarrow$ Float

Returns the cube root of $x$.

Domain: [0, INFINITY)
Codomain:[0, INFINITY)

```
-9.upto(9) {|x|
    p [x, Math.cbrt(x), Math.cbrt(x)**3]
}
##> [-9, -2.0800838230519, -9.0]
-2.0, -8.0]
    -1.91293118277239, -7.0
    -1.81712059283214, -6.0]
    -1.7099759466767, -5.0]
    -1.5874010519682, -4.0]
    -1.44224957030741, -3.0]
    -1.25992104989487, -2.0]
    -1.0, -1.0]
    0.0, 0.0]
    1.0, 1.0]
    1.25992104989487, 2.01
    1.44224957030741, 3.0]
    1.5874010519682, 4.0]
    1.7099759466767, 5.0]
    1.81712059283214, 6.0]
    1.91293118277239, 7.0]
    2.0, 8.0]
    2.0800838230519, 9.0]
```

$\cos (x) \rightarrow$ Float
Computes the cosine of $x$ (expressed in radians).
Returns a Float in the range -1.0..1.0.
Domain: (-INFINITY, INFINITY)
Codomain: [-1, 1]

## $\cosh (x) \rightarrow$ Float

Computes the hyperbolic cosine of $x$ (expressed in radians).

Domain: (-INFINITY, INFINITY)

## Codomain: [1, INFINITY)

```
Math.cosh(0)
```


## $\operatorname{erf}(x) \rightarrow$ Float

Calculates the error function of $x$.
Domain: (-INFINITY, INFINITY)
Codomain: $(-1,1)$
Math.erf(0)

$$
\text { erfc }(x) \rightarrow \text { Float }
$$

Calculates the complementary error function of $x$.
Domain: (-INFINITY, INFINITY)
Codomain: $(0,2)$

```
Math.erfc(0)
```

$\exp (x) \rightarrow$ Float
Returns e**x.
Domain: (-INFINITY, INFINITY)
Codomain: (0, INFINITY)

```
Math. exp(0)
Math. exp(1)
Math.exp(1.5)
```

frexp $(x) \rightarrow$ [fraction, exponent]
Returns a two-element array containing the normalized fraction (a Float) and exponent (a Fixnum) of $x$.


## gamma(x) $\rightarrow$ 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.

$\operatorname{hypot}(x, y) \rightarrow$ Float

Returns sqrt(x**2 + $\mathrm{y}^{* *} 2$ ), the hypotenuse of a rightangled triangle with sides $x$ and $y$.

Math.hypot(3, 4)

## Idexp(fraction, exponent) $\rightarrow$ float

Returns the value of fraction*(2**exponent).
fraction, exponent = Math.frexp(1234)
Math.ldexp(fraction, exponent)

## Igamma(x) $\rightarrow$ [float, -1 or 1]

Calculates the logarithmic gamma of $x$ and the sign of gamma of $x$.
::Igamma is same as
[Math.log(Math.gamma(x).abs), Math.gamma(x) < 0
but avoid overflow by : : gamma for large x.

```
Math.lgamma(0)
```


## $\log (x) \rightarrow$ Float

$\log (x$, 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)
Math.log(1)
Math.log(Math::E)
Math.log(Math::E**3)
```

```
Math. \(\log (12,3)\)
```


## $\log 10(x) \rightarrow$ Float

Returns the base 10 logarithm of $x$.
Domain: (0, INFINITY)
Codomain: (-INFINITY, INFINITY)
Math.log10(1)
Math.log10(10)
Math.log10(10**100) \#\#> 100.0

## $\log 2(x) \rightarrow$ 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)
```

$\sin (x) \rightarrow$ Float
Computes the sine of $x$ (expressed in radians).
Returns a Float in the range -1.0..1.0.
Domain: (-INFINITY, INFINITY)
Codomain: [-1, 1]
Math.sin(Math::PI/2)
$\sinh (x) \rightarrow$ Float
Computes the hyperbolic sine of $x$ (expressed in radians).

Domain: (-INFINITY, INFINITY)
Codomain: (-INFINITY, INFINITY)

```
Math.sinh(0) ##> 0.0
```


## sqrt(x) $\rightarrow$ Float

Returns the non-negative square root of $x$.
Domain: [0, INFINITY)
Codomain:[0, INFINITY)

```
0.upto(10) {|x|
    p [x, Math.sqrt(x), Math.sqrt(x)**2]
}
#=> [0, 0.0, 0.0]
    [1, 1.0, 1.0]
    [2, 1.4142135623731, 2.0]
    [3, 1.73205080756888, 3.0]
    2.0, 4.0]
    2.23606797749979, 5.0]
    2.44948974278318, 6.0]
    2.64575131106459, 7.0]
    2.82842712474619, 8.01
    3.0, 9.0]
    [10, 3.16227766016838, 10.0]
```


## $\tan (x) \rightarrow$ Float

Computes the tangent of $x$ (expressed in radians).
Domain: (-INFINITY, INFINITY)
Codomain: (-INFINITY, INFINITY)

```
Math.tan(0) ### 0.0
```


## $\tanh (x) \rightarrow$ Float

Computes the hyperbolic tangent of $x$ (expressed in radians).

Domain: (-INFINITY, INFINITY)
Codomain: $(-1,1)$
Math.tanh(0)

Generated by RDoc 3.12.2. Generated with the Darkfish Rdoc Generator 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:

Math::DomainError: Numerical argument is out

## In Files

math.c

## Parent

## StandardError

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## class Method

## Proc

## In Files

proc.c

## Parent

## Object

## Public Instance Methods

eql?(other_meth) $\rightarrow$ true or false meth $==$ other_meth $\rightarrow$ 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.
call(args, ...) $\rightarrow$ obj
meth[args, ...] $\rightarrow$ obj
Invokes the meth with the specified arguments, returning the method's return value.

```
m = 12.method("+")
m.call(3)
m.call(20)
```


## arity $\rightarrow$ 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 $-\mathrm{n}-1$, 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
```


## call(args, ...) $\rightarrow$ obj

 meth[args, ...] $\rightarrow$ objInvokes the meth with the specified arguments, returning the method's return value.

```
m = 12.method("+")
m.call(3)
m.call(20) #=> 32
```


## clone $\rightarrow$ new_method

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"
```


## curry $\rightarrow$ proc

## curry(arity) $\rightarrow$ 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)
proc2.call(3)
def vararg(*args)
    args
end
proc = self.method(:vararg).curry(4)
proc2 = proc.call(:x)
```

```
proc3 = proc2.call(:y, :z)
proc3.call(:a)
```

eql?(other_meth) $\rightarrow$ true or false meth $==$ other_meth $\rightarrow$ 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.
hash $\rightarrow$ integer
Returns a hash value corresponding to the method object.

See also Object\#hash.
to_s $\rightarrow$ string
inspect $\rightarrow$ string
Returns the name of the underlying method.

name $\rightarrow$ symbol
Returns the name of the method.

## original_name $\rightarrow$ symbol

Returns the original name of the method.

## owner $\rightarrow$ class_or_module

Returns the class or module that defines the method.

## parameters $\rightarrow$ array

Returns the parameter information of this method.
receiver $\rightarrow$ object
Returns the bound receiver of the method object.

## 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 Method of superclass, which would be called when super is used.
to_proc $\rightarrow$ proc
Returns a proc object corresponding to this method.
to_s $\rightarrow$ string
inspect $\rightarrow$ string
Returns the name of the underlying method.


## unbind $\rightarrow$ 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 RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 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).

```
module Mod
    include Math
    CONST = 1
    def meth
    end
end
Mod.class
Mod.constants
Mod.instance_methods
```


## In Files

- class.c
eval.c
$\square$ load.c
- object.c
$\square$ proc.c
- vm_eval.c
- vm_method.c


## Parent

## Object

## Public Class Methods

constants $\rightarrow$ array
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.

```
Module.constants.first(4)
Module.constants.include?(:SEEK_SET)
class IO
    Module.constants.include?(:SEEK_SET)
end
```

The second form calls the instance method constants.

## nesting $\rightarrow$ array

Returns the list of Modules nested at the point of call.

```
module M1
    module M2
        $a = Module.nesting
```

```
    end
end
$a
[M1::M2, M1]
$a[0].name
```

new $\rightarrow$ mod
new \{|mod| block \} $\rightarrow$ 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.

```
fred = Module.new do
    def meth1
        "hello"
    end
    def meth2
        "bye"
    end
end
a = "my string"
a.extend(fred)
#=> "my string"
#=> "hel1o"
a.meth2
#=> "bye"
```

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

## Public Instance Methods

## 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 " $A<B$ ".)

## 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 $\rightarrow-1,0,+1$, 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.

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

equal?(other) $\rightarrow$ true or false
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
obj == other
obj.equal? other
obj.equal? obj
```

The eql? method returns true if obj and other refer to the same hash key. This is used by Hash 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
1.eql? 1.0
```


## mod $===$ obj $\rightarrow$ 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.

## 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 " $B>A$ ".)

## 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 " $B>A$ ".)

Returns a list of modules included/prepended in mod (including mod itself).


## 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.

```
module A
end
A.autoload(:B, "b")
A: :B.doit
autoloads "b"
```


## autoload? (name) $\rightarrow$ String or nil

Returns filename to be loaded if name is registered as autoload in the namespace of mod.

```
module A
end
A.autoload(:B, "b")
A.autoload?(:B)
```

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:

module_exec(arg...) \{|var...| block $\} \rightarrow$ obj 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!
```


## class_variable_defined?(symbol) $\rightarrow$ true or

 falseclass_variable_defined?(string) $\rightarrow$ true or false
Returns true if the given class variable is defined in obj. String arguments are converted to symbols.

```
class Fred
    @@foo = 99
end
Fred.class_variable_defined?(:@@foo)
Fred.class_variable_defined?(:@@bar)
```

class_variable_get(symbol) $\rightarrow$ obj class_variable_get(string) $\rightarrow$ 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. String arguments are converted to symbols.

```
class Fred
    @@foo = 99
end
Fred.class_variable_get(:@@foo)
```

class_variable_set(symbol, obj) $\rightarrow$ obj class_variable_set(string, obj) $\rightarrow$ 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
```


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.

```
class One
    @@var1 = 1
end
class Two < One
    @@var2 = 2
end
One.class_variables
Two.class_variables
Two.class_variables(false) #=> [:@@var2]
```

const_defined?(sym, inherit=true) $\rightarrow$ true or
false
const_defined?(str, inherit=true) $\rightarrow$ true or false
Says whether mod or its ancestors have a constant with the given name:

```
Float.const_defined?(:EPSILON)
Float.const_defined?("String")
BasicObject.const_defined?(:Hash)

If mod is a Module, additionally Object and its ancestors are checked:


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:
```

module Admin
autoload :User, 'admin/user'
end
Admin.const_defined?(:User )

```

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:

IO. const_defined?(: SYNC)
IO.const_defined?(:SYNC, false)

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) \(\rightarrow\) obj
const_get(str, inherit=true) \(\rightarrow\) 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.
```

Math.const_get(:PI)

```

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".


\section*{const_missing(sym) \(\rightarrow\) 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:
```

def Foo.const_missing(name)
name \# return the constant name as Symbol.
end
Foo : : UNDEFINED_CONST
UNDEFINED_CONST

```


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.
```

def Object.const_missing(name)
@looked_for ||= {}
str_name = name.to_s
raise "Class not found: \#{name}" if @looked_for
@looked_for[str_name] = 1
file = str_name.downcase
require file
klass = const_get(name)
return klass if klass
raise "Class not found: \#{name}"
end

```

const_set(sym, obj) \(\rightarrow\) obj
const_set(str, obj) \(\rightarrow\) obj
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".


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.
```

IO.constants.include?(:SYNC)
IO.constants(false).include?(:SYNC)
true
false

```

Also see Module: :const_defined?.
freeze \(\rightarrow\) mod
Prevents further modifications to mod.
This method returns self.
include(module, ...) \(\rightarrow\) self
Invokes Module.append_features on each parameter in reverse order.

\section*{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
B.include?(A)
C.include?(A)
A.include?(A)

```
```

module Mixin
end
module Outer
include Mixin
end
Mixin.included_modules
Outer.included_modules
\#=> [Mixin]

```

\section*{inspect()}

Alias for: to_s

\section*{instance_method(symbol) \(\rightarrow\) 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(sel1
end
end
interpreter = Interpreter.new
interpreter.interpret('dave')

```

\section*{produces:}
```

Hello there, Dave!

```

\section*{instance_methods(include_super=true) \(\rightarrow\) 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)

```

\section*{method_defined?(symbol) \(\rightarrow\) true or false method_defined?(string) \(\rightarrow\) 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. String 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
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]]) \(\rightarrow\) obj
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:
```

Hello there!
dummy:123:in `module_eval': undefined local varia     or method `code' for Thing:Class
4

```
module_exec(arg...) \{|var...| block \} \(\rightarrow\) obj
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!

```
name \(\rightarrow\) string
Returns the name of the module mod. Returns nil for anonymous modules.

\section*{prepend(module, ...) \(\rightarrow\) self}

Invokes Module.prepend_features on each parameter in reverse order.
private_class_method(symbol, ...) \(\rightarrow\) mod 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

```

\title{
private_constant(symbol, ...) \(\rightarrow\) mod
}

Makes a list of existing constants private.
private_instance_methods(include_super=true)

\section*{(8) \(\rightarrow\) 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.
```

module Mod
def method1() end
private :method1
def method2() end
end
Mod.instance_methods
:method2]
Mod.private_instance_methods
:method1]

```
private_method_defined?(symbol) \(\rightarrow\) true or false
private_method_defined?(string) \(\rightarrow\) true or false
Returns true if the named private method is defined by _mod_ (or its included modules and, if mod is a class, its ancestors). String arguments are converted to symbols.
```

module A
def method1() end
end
class B
private
def method2() end
end
class C < B
include A

```
```

        def method3() end
    end
A.method_defined? :method1
\#=> true
C.private_method_defined? "method1" \#=> false
C.private_method_defined? "method2" \#=> true
C.method_defined? "method2"

```

\section*{protected_instance_methods(include_super=trı}

\section*{\(\rightarrow\) 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) -> true
or false
protected_method_defined?(string) -> true
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). String 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
C.protected_method_defined? "method1"
C.protected_method_defined? "method2"
C.method_defined? "method2"
c.method_defined? "method2"

```
public_class_method(symbol, ...) \(\rightarrow\) mod public_class_method(string, ...) \(\rightarrow\) mod Makes a list of existing class methods public.

String arguments are converted to symbols.
public_constant(symbol, ...) \(\rightarrow\) mod
Makes a list of existing constants public.
public_instance_method(symbol) \(\rightarrow\) unbound_method
Similar to \#instance_method, searches public method only.
public_instance_methods(include_super=true)

\section*{\(\rightarrow\) 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 false
public_method_defined?(string) \(\rightarrow\) true or false
Returns true if the named public method is defined by mod (or its included modules and, if mod is a class, its ancestors). String 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" \#\#> false
C.method_defined? "method2"
\#=> true

```

\section*{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

```

\section*{singleton_class? \(\rightarrow\) true or false}

Returns true if mod is a singleton class or false if it is an ordinary class or module.
```

class C
end
C.singleton_class?
\#=> false
C.singleton_class.singleton_class? \#=> true

```
to_s \(\rightarrow\) 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 RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{class Mutex}

Mutex implements a simple semaphore that can be used to coordinate access to shared data from multiple concurrent threads.

Example:
```

require 'thread'
semaphore = Mutex.new
a = Thread.new {
semaphore.synchronize {
\# access shared resource
}
}
b = Thread.new {
semaphore.synchronize {
\# access shared resource
}
}

```

\section*{In Files}
thread.c

\section*{Parent}

\section*{Object}

\section*{Public Class Methods}

\section*{Public Instance Methods}

\section*{lock \(\rightarrow\) self}

Attempts to grab the lock and waits if it isn't available. Raises ThreadError if mutex was locked by the current thread.

\section*{locked? \(\rightarrow\) true or false}

Returns true if this lock is currently held by some thread.

\section*{owned? \(\rightarrow\) true or false}

Returns true if this lock is currently held by current thread. This API is experimental, and subject to change.

\section*{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 Thread\#wakeup call. For example, receiving signal and so on.

\section*{synchronize \(\{\)... \} \(\rightarrow\) result of the block}

Obtains a lock, runs the block, and releases the lock when the block completes. See the example under Mutex.
try_lock \(\rightarrow\) true or false
Attempts to obtain the lock and returns immediately. Returns true if the lock was granted.

\section*{unlock \(\rightarrow\) self}

Releases the lock. Raises ThreadError if mutex wasn't locked by the current thread.

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{class NameError}

Raised when a given name is invalid or undefined.

\author{
puts foo
}
raises the exception:

NameError: undefined local variable or method 4

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

\section*{Parent}

\section*{StandardError}

\section*{Public Class Methods}
new(msg [, name]) \(\rightarrow\) name_error
Construct a new NameError exception. If given the name parameter may subsequently be examined using the NameError . name method.

\section*{Public Instance Methods}

\section*{name \(\rightarrow\) string or nil}

Return the name associated with this NameError exception.

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{class NilClass}

The class of the singleton object nil.

\section*{In Files}
complex.c
object.c
- rational.c

\section*{Parent}

\section*{Object}

\section*{Public Instance Methods}

\author{
false \& obj \(\rightarrow\) false \\ nil \& obj \(\rightarrow\) false
}

And-Returns false. obj is always evaluated as it is the argument to a method call-there is no shortcircuit evaluation in this case.
false \({ }^{\wedge}\) obj \(\rightarrow\) true or false
nil ^ obj \(\rightarrow\) true or false
Exclusive Or—lf obj is nil or false, returns false;
otherwise, returns true.
inspect \(\rightarrow\) "nil"

Always returns the string "nil".

\section*{nil? \(\rightarrow\) true}

Only the object nil responds true to nil?.

\section*{rationalize \(([\mathrm{eps}]) \rightarrow\) (0/1)}

Returns zero as a rational. The optional argument eps is always ignored.

\section*{to_a \(\rightarrow\) []}

Always returns an empty array.
```

nil.to_a

```
\[
\text { to_c } \rightarrow(0+0 i)
\]

Returns zero as a complex.
\[
\text { to_f } \rightarrow 0.0
\]

Always returns zero.
```

nil.to_f

```
to_h \(\rightarrow\) \{\}
Always returns an empty hash.
```

nil.to_h

```
to_i \(\rightarrow 0\)
Always returns zero.
```

nil.to_i

```
\[
\text { to_r } \rightarrow \text { (0/1) }
\]

Returns zero as a rational.
to_s \(\rightarrow\) ""
Always returns the empty string.
false | obj \(\rightarrow\) true or false
nil | obj \(\rightarrow\) true or false
Or—Returns false if obj is nil or false; true otherwise.

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\title{
class NoMemoryError
}

Raised when memory allocation fails.

\section*{In Files}
- error.c

\section*{Parent}

\section*{Exception}

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{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:

NoMethodError: undefined method 'to_ary' for

\section*{In Files}
error.c

\section*{Parent}

\section*{NameError}

\section*{Public Class Methods}
new(msg, name [, args]) \(\rightarrow\) no_method_error Construct 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.

\section*{Public Instance Methods}
args \(\rightarrow\) obj
Return the arguments passed in as the third parameter to the constructor.

Generated by RDoc 3.12.2. Generated with the Darkfish Rdoc Generator 3.

\section*{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.

\section*{In Files}
error.c

\section*{Parent}

\section*{ScriptError}

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{class Numeric}

The top-level number class.

\section*{In Files}
complex.c
numeric.c
- rational.c

\section*{Parent}

\section*{Object}

\section*{Included Modules}
- Comparable

\section*{Public Instance Methods}
modulo(numeric) \(\rightarrow\) real
x.modulo(y) means \(x-y^{*}(x / y) \cdot f l o o r\)

Equivalent to num.divmod(numeric) [1].
See \#divmod.
+num \(\rightarrow\) num
Unary Plus-Returns the receiver's value.
(8) -num \(\rightarrow\) numeric

Unary Minus—Returns the receiver's value, negated.
number <=> other \(\rightarrow 0\) or nil
Returns zero if number equals other, otherwise nil is returned if the two values are incomparable.
abs \(\rightarrow\) numeric
magnitude \(\rightarrow\) numeric
Returns the absolute value of num.
```

12.abs
(-34.56).abs
-34.56.abs

```
\#magnitude is an alias of \#abs.
abs2 \(\rightarrow\) real
Returns square of self.
arg \(\rightarrow 0\) or float
angle \(\rightarrow 0\) or float
phase \(\rightarrow 0\) or float
Returns 0 if the value is positive, pi otherwise.
arg \(\rightarrow 0\) or float
angle \(\rightarrow 0\) or float
phase \(\rightarrow 0\) or float
Returns 0 if the value is positive, pi otherwise.
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.
```

1.ceil
1.2.ceil
(-1.2).ceil
(-1.0).ceil

```

\section*{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)

```
conj \(\rightarrow\) self
conjugate \(\rightarrow\) self
Returns self.
conj \(\rightarrow\) self
conjugate \(\rightarrow\) self
Returns self.
denominator \(\rightarrow\) integer

Returns the denominator (always positive).

\section*{div(numeric) \(\rightarrow\) 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.

\section*{divmod(numeric) \(\rightarrow\) array}

Returns an array containing the quotient and modulus obtained by dividing num by numeric.

If \(q, r=\) * \(x \cdot \operatorname{divmod}(y)\), then
```

q= floor(x/y)
x = q* y+r

```

The quotient is rounded toward -infinity, as shown in the following table:


\section*{Examples}
```

11.divmod(3)
11.divmod(-3)
11.divmod(3.5)
(-11).divmod(3.5)
(11.5).divmod(3.5)

```

\section*{eql?(numeric) \(\rightarrow\) true or false}

Returns true if num and numeric are the same type and have equal values.
```

1 == 1.0
1.eql?(1.0)
(1.0).eql?(1.0)
false

```
fdiv(numeric) \(\rightarrow\) float
Returns float division.
floor \(\rightarrow\) 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.
1.floor
(-1).floor
i \(\rightarrow\) Complex(0,num)
Returns the corresponding imaginary number. Not
available for complex numbers.
imag \(\rightarrow 0\)
imaginary \(\rightarrow 0\)
Returns zero.
imag \(\rightarrow 0\)
imaginary \(\rightarrow 0\)
Returns zero.
initialize_copy(p1)
Numerics are immutable values, which should not be copied.

Any attempt to use this method on a Numeric will raise a TypeError.
integer? \(\rightarrow\) true or false
Returns true if num is an Integer (including Fixnum and Bignum).
(1.0).integer?
(1).integer?
abs \(\rightarrow\) numeric
magnitude \(\rightarrow\) numeric
Returns the absolute value of num.
(-34.56).abs
-34.56.abs
\#magnitude is an alias of \#abs.
modulo(numeric) \(\rightarrow\) real
x.modulo(y) means \(x-y^{*}(x / y)\).floor

Equivalent to num.divmod(numeric) [1].

See \#divmod.
nonzero? \(\rightarrow\) self or nil
Returns self if num is not zero, nil otherwise.
This behavior is useful when chaining comparisons:

numerator \(\rightarrow\) integer
Returns the numerator.
(8) \(\arg \rightarrow 0\) or float
angle \(\rightarrow 0\) or float
phase \(\rightarrow 0\) or float
Returns 0 if the value is positive, pi otherwise.
polar \(\rightarrow\) array
Returns an array; [num.abs, num.arg].
quo(int_or_rat) \(\rightarrow\) rat
quo(flo) \(\rightarrow\) flo
Returns most exact division (rational for integers, float for floats).
real \(\rightarrow\) self
Returns self.
real? \(\rightarrow\) true or false

Returns true if num is a Real number. (i.e. not Complex).
rect \(\rightarrow\) array
rectangular \(\rightarrow\) array
Returns an array; [num, 0].
rect \(\rightarrow\) array
rectangular \(\rightarrow\) array
Returns an array; [num, 0].
remainder(numeric) \(\rightarrow\) real
\(x . r e m a i n d e r(y)\) means \(x-y^{*}(x / y) . t r u n c a t e\)
See \#divmod.

\section*{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.

\section*{singleton_method_added(p1)}

Trap attempts to add methods to Numeric objects. Always raises a TypeError.

Numerics should be values; singleton_methods should not be added to them.

\title{
step(by: step, to: limit) \(\{\) li| block \(\} \rightarrow\) self step(by: step, to: limit) \(\rightarrow\) an_enumerator step(limit=nil, step=1) \(\{|i|\) block \(\} \rightarrow\) self 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:

\section*{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:
p 1.step.take(4)
p 10.step(by: -1).take(4)
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

Will produce:


\section*{to_c \(\rightarrow\) complex}

Returns the value as a complex.
to_int \(\rightarrow\) 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

```

\section*{truncate \(\rightarrow\) integer}

Returns num truncated to an Integer.
Numeric implements this by converting its value to a Float and invoking Float\#truncate.
zero? \(\rightarrow\) true or false
Returns true if num has a zero value.

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{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 Object you do not need to use the full namespace. For example, referencing File inside Yourclass will find the top-level File class.

In the descriptions of Object's methods, the parameter symbol refers to a symbol, which is either a quoted string or a Symbol (such as : name).
```

In Files
class.c
enumerator.c
eval.c
= gc.c
hash.c

```
io.c
object.c
parse.c
proc.c
ruby.c
version.c
\(=\mathrm{vm} . \mathrm{c}\)
@ vm_eval.c
- vm_method.c

\section*{Parent}

\section*{BasicObject}

\section*{Included Modules}

Kernel

\section*{Constants}

\section*{ARGF}

ARGF is a stream designed for use in scripts that process files given as command-line arguments or passed in via STDIN.

See ARGF (the class) for more details.

\section*{ARGV}

ARGV 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.

\section*{DATA}

DATA is a File that contains the data section of the executed file. To create a data section use \(\qquad\) END \(\qquad\) :
```

\$ 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.

\section*{FALSE}

An alias of false

\section*{NIL}

An alias of nil

\section*{RUBY_COPYRIGHT}

The copyright string for ruby

\section*{RUBY_DESCRIPTION}

The full ruby version string, like ruby -v prints'

\section*{RUBY_ENGINE}

The engine or interpreter this ruby uses.

\section*{RUBY_PATCHLEVEL}

The patchlevel for this ruby. If this is a development build of ruby the patchlevel will be -1

\section*{RUBY_PLATFORM}

The platform for this ruby

\section*{RUBY_RELEASE_DATE}

The date this ruby was released

\section*{RUBY_REVISION}

The SVN revision for this ruby.

\section*{RUBY_VERSION}

The running version of ruby

\section*{SCRIPT_LINES}

When a Hash is assigned to SCRIPT_LINES__ the contents of files loaded after the assignment will be added as an Array of lines with the file name as the key.

\section*{STDERR}

Holds the original stderr

\section*{STDIN}

Holds the original stdin

\section*{STDOUT}

Holds the original stdout

\section*{TOPLEVEL_BINDING}

The Binding of the top level scope

\section*{TRUE}

An alias of true

\section*{Public Instance Methods}

\section*{obj !~ other \(\rightarrow\) true or false}

Returns true if two objects do not match (using the =~ method), otherwise false.

\section*{obj <=> other \(\rightarrow \mathbf{0}\) 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 Enumerable\#sort, 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 Comparable to gain the methods <=, <, ==, >=, > and between?.

\section*{obj \(===\) other \(\rightarrow\) true or false}

Case Equality - For class Object, effectively the same as calling \#==, but typically overridden by descendants to provide meaningful semantics in case statements.
obj \(=\sim\) other \(\rightarrow\) nil

Pattern Match—Overridden by descendants (notably Regexp and string) to provide meaningful patternmatch semantics.

\section*{class \(\rightarrow\) 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
self.class
Obiect

```

\section*{clone \(\rightarrow\) 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 object\#dup.
```

class Klass
attr_accessor :str
end
s1 = Klass.new
s1.str = "Hello"
s2 = s1.clone
s2.str[1,4] = "i"
s1.inspect
s2.inspect

```


This method may have class-specific behavior. If so, that behavior will be documented under the \#initialize_copy method of the class.
define_singleton_method(symbol, method)
\(\rightarrow\) new_method
define_singleton_method(symbol) \{ block \}

83 \(\rightarrow\) 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.


\section*{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:

\section*{dup \(\rightarrow\) 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.

\section*{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 dup, any modules that the object has been extended with will not be copied.

enum_for(method \(=\) :each, *args) \(\rightarrow\) enum to_enum(method \(=\) :each, *args) \(\{\mid\) *args| block\} \(\rightarrow\) enum enum_for(method = :each, *args)\{|*args| block\} \(\rightarrow\) 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).

\section*{Examples}

It is typical to call \#to_enum 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
unless block_given?
return to_enum(__method
n) do
metho
sz = size \# Call size and multiply
sz * n if sz \# but return nill if size
end

```
```

    end
    each do |*val|
        n,times { yield *val }
        end
    end
    end
    %[hello world],repeat(2) { |w| puts w }
    # => Prints rhellor, rhellor, rworldr, rworld'
    enum = (1..14).repeat(3)
    num.first(4)
    enum.size
    ```
    obj \(==\) other \(\rightarrow\) true or false
    equal?(other) \(\rightarrow\) true or false
    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
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 Hash 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

```

\section*{extend(module, ...) \(\rightarrow\) obj}

Adds to obj the instance methods from each module given as a parameter.
```

module Mod
def hello
"Hello from Mod.\n"
end
end
class Klass
def hello
"Hello from Klass.\n"
end
end
k = Klass.new
k.hello \#=> "Hello from Klass.\n'
k.extend(Mod) \#=> \#[Klass:0x401b3bc8](Klass:0x401b3bc8)
k.hello \#=> "Hell.lo from Mod.\n"

```
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.

\section*{frozen? \(\rightarrow\) true or false}

Returns the freeze status of obj.
```

a = [ "a", "b", "c" ]
a.freeze
a.frozen?

```
hash \(\rightarrow\) fixnum
Generates a Fixnum hash value for this object. This function must have the property that a.eql? (b) implies a.hash == b.hash.

The hash value is used along with eql? by the Hash class to determine if two objects reference the same hash key. Any hash value that exceeds the capacity of a Fixnum 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.

\section*{inspect \(\rightarrow\) 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 inspect 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.
\begin{tabular}{|c|c|}
\hline [ 1, 2, 3..4, 'five' ].inspect Time.new.inspect & \[
\begin{aligned}
& \# \Rightarrow=[1,2,3 \ldots \\
& \#=>=2008-03-08
\end{aligned}
\] \\
\hline class Foo end & \\
\hline Foo.new. inspect & \#=> "\#<Foo:0x03 \\
\hline class Bar & \\
\hline def initialize & \\
\hline \begin{tabular}{l}
@bar = 1 \\
end
\end{tabular} & \\
\hline end & \\
\hline Bar.new. inspect & \#\#=> "\#<Bar:0x03 \\
\hline 1 & \(\square\) \\
\hline
\end{tabular}

\section*{instance_of?(class) \(\rightarrow\) true or false}

Returns true if obj is an instance of the given class.
See also object\#kind_of?.
```

class A; end
class B < A; end
class C < B; end
b = B.new
b.instance_of? A \#=> false
b.instance_of? B \#=> true
b.instance_of? C \#\#> false

```
instance_variable_defined?(symbol) \(\rightarrow\) true
or false
instance_variable_defined?(string) \(\rightarrow\) true or
false

Returns true if the given instance variable is defined
in obj. 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_defined?(:@a)
fred.instance_variable_defined?("@b")
fred.instance_variable_defined?("@c")

```
instance_variable_get(symbol) \(\rightarrow\) obj
instance_variable_get(string) \(\rightarrow\) 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)
fred.instance_variable_get("@b")

```
instance_variable_set(symbol, obj) \(\rightarrow\) obj instance_variable_set(string, obj) \(\rightarrow\) 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.
```

class Fred
def initialize(p1, p2)
@a, @b = p1, p2
end
end
fred = Fred.new('cat', 99)
fred.instance_variable_set(:@a, 'dog') \#=> "dog
fred.instance_variable_set(:@c, 'cat') \#=> "cal
fred.inspect

```

\section*{instance_variables \(\rightarrow\) array}

Returns an array of instance variable names for the receiver. Note that simply defining an accessor does not create the corresponding instance variable.
```

class Fred
attr_accessor :a1
def initialize
@iv = 3
end
end
Fred.new.instance_variables \#=> [:@iv]

```

\section*{is_a?(class) \(\rightarrow\) true or false \\ kind_of?(class) \(\rightarrow\) 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.

```

b = B.new
b.is_a? A
\#=> true
b.is_a? B
\#=> true
b.is_a? C \#=> false
b.is_a? M \#\#> true
b.kind_of? A \#\#> true
b.kind_of? B \#\#> true
b.kind_of? C \#\#\# false
b.kind_of? M \#\#> true

```

\section*{itself \(\rightarrow\) an_object}

Returns obj.
```

string = 'my string' \#\#=> "my string"
string.itself.object_id == string.object_id \#\#=>

```
4
is_a?(class) \(\rightarrow\) true or false
kind_of?(class) \(\rightarrow\) 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.


\section*{method(sym) \(\rightarrow\) 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 \#\#> "Hel.10, @iv = Fred

```

\section*{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.
```

class Klass
def klass_method()
end
end
k = Klass.new

```
k.methods[0..9] \# \(\#\) [:klass method, :nil?
```

k.methods.length
k.methods(false)
def k.singleton_method; end
k.methods(false) \#=> [:singleton method]
module M123; def m123; end end
k.extend M123
k.methods(false) \#=> [:singleton method]

```

\section*{nil? \(\rightarrow\) true or false}

Only the object nil responds true to nil?.
```

Object.new.nil?
false

```
nil.nil?
true

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) \(\rightarrow\) 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.

\section*{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.

\section*{public_method(sym) \(\rightarrow\) method}

Similar to method, searches public method only.

\section*{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...]) $\rightarrow$ obj
public_send(string [, args...]) $\rightarrow$ obj

```

Invokes the method identified by symbol, passing it any arguments specified. Unlike send, \#public_send calls public methods only. When the method is identified by a string, the string is converted to a symbol.

remove_instance_variable(symbol) \(\rightarrow\) obj
Removes the named instance variable from obj, returning that variable's value.
```

class Dummy
attr_reader :var
def initialize
@var = 99
end
def remove
remove_instance_variable(:@var)
end
end
d = Dummy.new
d.var
d.remove
d.var

```
respond_to?(symbol, include_all=false) \(\rightarrow\)
true or false
respond_to?(string, include_all=false) \(\rightarrow\)
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 Process.fork on
Windows, File.Ichmod 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?(symbol, include_all)
\(\rightarrow\) true or false
respond_to_missing?(string, include_all) \(\rightarrow\) 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 respond_to?, and the example of BasicObject.
send(symbol [, args...]) \(\rightarrow\) obj
8)
__send__(symbol [, args...]) \(\rightarrow\) obj send(string [, args...]) \(\rightarrow\) obj
© _ send__(string [, args...]) \(\rightarrow\) 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.


\section*{singleton_class \(\rightarrow\) 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 NilClass,
TrueClass, or FalseClass, respectively. If obj is a
Fixnum or a Symbol, it raises a TypeError.
```

Object.new.singleton_class
String.singleton_class
nil.singleton_class

```

\section*{singleton_method(sym) \(\rightarrow\) method}

Similar to method, searches singleton method only.
```

class Demo
def initialize(n)
@iv = n
end
def hello()
"Hello, @iv = \#{@iv}"
end
end
k = Demo.new(99)
def k.hi
"Hi, @iv = \#{@iv}"
end
m = k.singleton_method(:hi)
m.call
m = k.singleton_method(:hello) \#=> NameError

```

\section*{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

```


\section*{taint \(\rightarrow\) 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 Kernel\#gets, 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.

\section*{tainted? \(\rightarrow\) true or false}

Returns true if the object is tainted.
See taint for more information.

\section*{\(\operatorname{tap}\{|x| . ..\} \rightarrow\) 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.
\begin{tabular}{|c|c|c|c|c|c|}
\hline (1. 10) & .tap & \{|x| & & & \\
\hline to_a & .tap & \{|x| & puts & " & arr \\
\hline select \(\{|x| \times \% 2=0\}\) & .tap & \{|x| & put & & e \\
\hline .map \(\left\{|x| x^{*} \times\right\}\) & .tap & \{|x| & & & \\
\hline
\end{tabular}
to_enum(method \(=\) :each, *args) \(\rightarrow\) enum
enum_for(method = :each, *args) \(\rightarrow\) enum to_enum(method \(=\) :each, *args) \{|*args|
block\} \(\rightarrow\) enum
enum_for(method = :each, *args)\{|*args| block\} \(\rightarrow\) 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).

\section*{Examples}


It is typical to call \#to_enum 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:


\section*{to_s \(\rightarrow\) 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".

\section*{trust \(\rightarrow\) obj}

Deprecated method that is equivalent to untaint.

\section*{untaint \(\rightarrow\) obj}

Removes the tainted mark from the object.
See taint for more information.
untrust \(\rightarrow\) obj
Deprecated method that is equivalent to taint.
untrusted? \(\rightarrow\) true or false
Deprecated method that is equivalent to tainted?.

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{module ObjectSpace}

The ObjectSpace 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.
```

a = "A"
b = "B"
ObjectSpace.define_finalizer(a, proc {|id| pu
ObjectSpace.define_finalizer(b, proc {|id| pu

```

produces:

Finalizer two on 537763470
Finalizer one on 537763480

\section*{In Files}
gc.c

\section*{Public Class Methods}

\section*{8) _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.


\section*{count_objects([result_hash]) \(\rightarrow\) hash}

Counts objects for each type.
It returns a hash, such as:
```

{
:TOTAL=>10000,
:FREE=>3011,
:T_OBJECT=>6,
:T_CLASS=>404,
}

```

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.

\section*{define_finalizer(obj, aProc=proc())}

Adds aProc as a finalizer, to be called after obj was destroyed.
each_object([module]) \(\{|0 b j| \ldots\} \rightarrow\) 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.
```

a = 102.7
b = 95
c = 12345678987654321
count = ObjectSpace.each_object(Numeric) {|x| p
puts "Total count: \#{count}"

```

produces:

\section*{12345678987654321}
102.7
2.71828182845905
3.14159265358979
2. \(22044604925031 e-16\)
1.7976931348623157e+308
2. 2250738585072e-308

Total count: 7
start \(\rightarrow\) nil
garbage_collect \(\rightarrow\) nil
start(full_mark: true, immediate_sweep:
true) \(\rightarrow\) nil
garbage_collect(full_mark: true, 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 GC. 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.

\section*{undefine_finalizer(obj)}

Removes all finalizers for obj.

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\title{
class \\ ObjectSpace::WeakMap
}

An ObjectSpace::WeakMap 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.

\section*{In Files}
gc.c

\section*{Parent}

\section*{Object}

\section*{Included Modules}
- Enumerable

\section*{Public Instance Methods}

\section*{[ p 1 1)}

Retrieves a weakly referenced object with the given key

\section*{[]=(p1, p2)}

Creates a weak reference from the given key to the given value

\section*{each()}

Iterates over keys and objects in a weakly referenced object

\section*{each_key()}

Iterates over keys and objects in a weakly referenced object

\section*{each_pair()}

Iterates over keys and objects in a weakly referenced object

\section*{each_value()}

Iterates over keys and objects in a weakly referenced object

\section*{include?(p1)}

Returns true if key is registered
inspect()

\section*{key?(p1)}

Returns true if key is registered

\section*{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 RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{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.
```

def gen_times(factor)
return Proc.new {|n| n*factor }
end
times3 = gen_times(3)
times5 = gen_times(5)
times3.call(12)
times5.call(5)
times3.call(times5.call(4))
36
times3.call(12)
times3.call(times5.call(4))

## In Files

proc.c

## Parent

## Object

## Public Class Methods

new $\{\mid$...| block $\} \rightarrow$ a_proc
new $\rightarrow$ 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.

```
def proc_from
    Proc.new
end
proc = proc_from { "hello" }
proc.call
```


## Public Instance Methods

## proc === obj $\rightarrow$ result_of_proc

Invokes the block with obj as the proc's parameter like \#call. It is to allow a proc object to be a target of when clause in a case statement.
call(params,...) $\rightarrow$ obj
prc[params,...] $\rightarrow$ obj
(params,...) $\rightarrow$ 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 Proc 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.

```
a_proc = Proc.new {|a, *b| b.collect {|i| i*a }}
a_proc.call(9, 1, 2, 3)
a_proc[9, 1, 2, 3]
a_proc = lambda {|a,b| a}
a_proc.call(1, 2, 3)
```


produces:


## arity $\rightarrow$ 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.



## binding $\rightarrow$ binding

Returns the binding associated with prc. Note that Kernel\#eval accepts either a Proc or a Binding object as its second parameter.

```
def fred(param)
    proc {}
end
b = fred(99)
eval("param", b.binding)
call(params,...) \(\rightarrow\) obj prc[params,...] \(\rightarrow\) obj (params,...) \(\rightarrow\) 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 Proc 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.

```
a_proc = Proc.new {|a, *b| b.collect {|i| i*a }}
a_proc.call(9, 1, 2, 3)
a_proc[9, 1, 2, 3]
a_proc = lambda {|a,b| a}
a_proc.call(1, 2, 3)
```

produces:

```
prog.rb:4:in `block in <main>': wrong number of
    from prog.rb:5:in `call'
    from prog.rb:5:in `<main>'
```


## curry $\rightarrow$ a_proc

89) curry(arity) $\rightarrow$ 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.

```
b = proc {|x, y, z| (x||0) + (y||0) + (z||0) }
p b.curry[1][2][3]
p b.curry[1, 2][3, 4]
p b.curry(5)[1][2][3][4][5]
p b.curry(5)[1, 2][3, 4][5]
p b.curry(1)[1]
b = proc {|x, y, z, *w| (x||0) + (y||0) + (z||0)
p b.curry[1][2][3]
```

| p b.curry[1, 2][3, 4] | \# $=>10$ |
| :---: | :---: |
| p b.curry(5)[1][2][3][4][5] | \%=> 15 |
| p b.curry(5)[1, 2][3, 4][5] | \#=> 15 |
| p b.curry(1)[1] | \#= |
| $\mathrm{b}=$ lambda $\{\|\mathrm{x}, \mathrm{y}, \mathrm{z}\|(\mathrm{x}\|\mid 0)$ | $+(y\| \| 0)+(z\| \| 0)\}$ |
| p b.curry[1][2][3] | \# $=>6$ |
| p b.curry[1, 2][3, 4] | \#=> wrong number of |
| p b.curry(5) | \#=> wrong number of |
| p b.curry(1) | \#=> wrong number of |
| $\mathrm{b}=$ lambda $\left\{\left\|\mathrm{x}, \mathrm{y}, \mathrm{z},{ }^{*} \mathrm{w}\right\| \mathrm{x} \mid\right.$ | $\mid 0)+(y\| \| 0)+(z\| \| C$ |
| p b.curry[1][2][3] | \#=> 6 |
| p b.curry[1, 2][3, 4] | \#=> 10 |
| p b.curry(5)[1][2][3][4][5] | \#=> 15 |
| p b.curry(5)[1, 2][3, 4][5] | \#=> 15 |
| p b.curry(1) | \#=> wrong number of |
| $\mathrm{b}=\operatorname{proc}\{$ :foo \} |  |
| p b.curry[] | \#=> : foo |
| 4 | $\bullet$ |

hash $\rightarrow$ integer
Returns a hash value corresponding to proc body.
See also Object\#hash.
inspect()
Alias for: to_s

## lambda? $\rightarrow$ true or false

Returns true for a Proc object for which argument handling is rigid. Such procs are typically generated by lambda.

A Proc object generated by proc ignores extra arguments.

```
proc {|a,b| [a,b] }.call(1,2,3)
```

It provides nil for missing arguments.

```
proc {|a,b| [a,b] }.call(1)
```

\#=> [1, nil]

It expands a single array argument.

```
\(\operatorname{proc}\{|a, b|[a, b]\} . c a l l([1,2])\)
```

\#=> [1,2]

A Proc object generated by lambda doesn't have such tricks.

```
lambda {|a,b| [a,b] }.call(1,2,3)
lambda {|a,b| [a,b] }.call(1)
lambda {|a,b| [a,b] }.call([1,2])
#=> ArgumentE
#=> ArgumentE
ArgumentE
```


\#lambda? is a predicate for the tricks. It returns true if no tricks apply.

```
lambda {}.lambda?
#=> true
proc {}.lambda?
```

:.new is the same as proc.

```
Proc.new {}.lambda?
```

lambda, proc and ::new preserve the tricks of a Proc object given by \& argument.

```
lambda(&lambda {}).lambda?
#=> true
proc(&lambda {}).lambda? ##> true
Proc.new(&lambda {}).lambda? ##> true
lambda(&proc {}).lambda?
proc(&proc {}).lambda?
Proc.new(&proc {}).lambda? #=> false
```

A Proc object generated by \& argument has the tricks

```
def n(&b) b.lambda? end
```

n \{\}
\#=> false
The \& argument preserves the tricks if a Proc object
is given by \& argument.

```
n(&lambda {})
#=> true
n(&proc {})
#=> false
n(&Proc.new {})
\#=> false
```

A Proc object converted from a method has no tricks.

```
def m() end
method(:m).to_proc.lambda? ##> true
n(&method(:m))
n(&method(:m).to_proc) #=> true
```

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 Proc 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)
```

The wrapper def2 defines a method which has no tricks.

## parameters $\rightarrow$ array

Returns the parameter information of this proc.

```
prc = lambda{|x, y=42, *other|}
prc.parameters
```



## 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)

## to_proc $\rightarrow$ proc

Part of the protocol for converting objects to Proc objects. Instances of class Proc simply return themselves.

## to_s $\rightarrow$ string

Returns the unique identifier for this proc, along with an indication of where the proc was defined.

Also aliased as: inspect
call(params,...) $\rightarrow$ obj
prc[params,...] $\rightarrow$ obj
(params,...) $\rightarrow$ 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 Proc 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 RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## module Process

## In Files

- process.c
- 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 (addressspace) 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 $m s g$ is given, it is written to STDERR prior to terminating.

## argv0 $\rightarrow$ 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.

## 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_REAI
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_CPI :hertz)+ returns the clock ticks per second.
+::clock_getres(:CLOCK_BASED_CLOCK_PROCESS_CF :hertz)+ returns CLOCKS_PER_SEC.

clock_gettime(clock_id [, unit]) $\rightarrow$ number
Returns a time returned by POSIX :::clock_gettime() 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

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_CPUTII
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 CLOCK_REALTIME 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. Time.now is recommended over CLOCK_REALTIME.

## daemon() $\rightarrow 0$

daemon(nochdir=nil,noclose=nil) $\rightarrow 0$
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 $/ \mathrm{dev} /$ null. Return zero on success, or raise one of Errno::*.

## 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 $\rightarrow$ fixnum

Process::GID.eid $\rightarrow$ fixnum
Process::Sys.geteid $\rightarrow$ fixnum
Returns the effective group ID for this process. Not available on all platforms.

## egid $=$ fixnum $\rightarrow$ fixnum

Sets the effective group ID for this process. Not available on all platforms.

## euid $\rightarrow$ fixnum

Process::UID.eid $\rightarrow$ fixnum
Process::Sys.geteuid $\rightarrow$ fixnum
Returns the effective user ID for this process.

## 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, 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
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 IO\#pid for IO.popen) 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", "argve"], "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 Kernel\#system if this is not acceptable.
exec "echo *"
\# never get here $\quad$ \# echoes list of files in cur

## 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).

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.

```
Process.exit!(true)
```


## fork [\{ block \}] $\rightarrow$ fixnum or nil

 fork [\{ block \}] $\rightarrow$ fixnum or nilCreates 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) $\rightarrow$ integer

Returns the process group ID for the given process id. Not available on all platforms.

Process.getpgid(Process.ppid())

## getpgrp $\rightarrow$ integer

Returns the process group ID for this process. Not available on all platforms.

```
Process.getpgid(0)
Process.getpgrp
```


## getpriority(kind, integer) $\rightarrow$ 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.

```
Process.getpriority(Process: :PRIO_USER, 0)
Process.getpriority(Process::PRIO_PROCESS,
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 :: setrlimit and the
system getrlimit(2) manual for details.

\section*{getsid() \(\rightarrow\) integer \\ getsid(pid) \(\rightarrow\) integer}

Returns the session ID for for the given process id. If not give, return current process sid. Not available on all platforms.
```

Process.getsid()
Process.getsid(0)
Process.getsid(Process.pid())

```

\section*{gid \(\rightarrow\) fixnum}

Process::GID.rid \(\rightarrow\) fixnum
Process::Sys.getgid \(\rightarrow\) fixnum
Returns the (real) group ID for this process.
```

Process.gid

```
gid= fixnum \(\rightarrow\) fixnum
Sets the group ID for this process.
groups \(\rightarrow\) array

Get an Array of the gids of groups in the supplemental group access list for this process.
```

Process.groups

```

\section*{groups= array \(\rightarrow\) array}

Set the supplemental group access list to the given Array of group IDs.


\section*{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.


\section*{kill(signal, pid, ...) \(\rightarrow\) 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 or 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.

\section*{maxgroups \(\rightarrow\) fixnum}

Returns the maximum number of gids allowed in the supplemental group access list.

Process.maxgroups

\section*{maxgroups= fixnum \(\rightarrow\) fixnum}

Sets the maximum number of gids allowed in the supplemental group access list.
pid \(\rightarrow\) fixnum

Returns the process id of this process. Not available on all platforms.
```

Process.pid

```

\section*{ppid \(\rightarrow\) 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

```

\section*{setpgid(pid, integer) \(\rightarrow 0\)}

Sets the process group ID of pid (0 indicates this process) to integer. Not available on all platforms.

\section*{setpgrp \(\rightarrow 0\)}

Equivalent to setpgid(0,0). Not available on all platforms.

\section*{setpriority(kind, integer, priority) \(\rightarrow \mathbf{0}\)}

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)

\section*{setproctitle(string) \(\rightarrow\) 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 NotlmplementedError be raised even if the platform does not support the feature.

Calling this method does not affect the value of \(\$ 0\).


This method first appeared in Ruby 2.1 to serve as a global variable free means to change the process title.

\section*{setrlimit(resource, cur_limit, max_limit) \(\rightarrow\)}
nil
setrlimit(resource, cur_limit) \(\rightarrow\) 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)

```

\section*{CORE}
```

core size (bytes) (SUSv3)

```

\section*{CPU}

CPU time (seconds) (SUSv3)

\section*{DATA}
data segment (bytes) (SUSv3)

\section*{FSIZE}
file size (bytes) (SUSv3)

\section*{MEMLOCK}
total size for mlock(2) (bytes) (4.4BSD, GNU/Linux)

\section*{MSGQUEUE}
allocation for POSIX message queues (bytes) (GNU/Linux)

\section*{NICE}
ceiling on process's nice(2) value (number) (GNU/Linux)

\section*{NOFILE}
file descriptors (number) (SUSv3)

\section*{NPROC}
number of processes for the user (number) (4.4BSD, GNU/Linux)

\section*{RSS}
resident memory size (bytes) (4.2BSD, GNU/Linux)

\section*{RTPRIO}
\begin{tabular}{l} 
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 \\
\hline stack size (bytes) (SUSv3) \\
\hline
\end{tabular}
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.

\section*{Process.setrlimit(:CORE, Process.getrlimit(:CORE}

\section*{setsid \(\rightarrow\) 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]) \(\rightarrow\) pid spawn([env,] command... [,options]) \(\rightarrow\) pid
spawn executes specified command and return its pid.
```

pid = spawn("tar xf ruby-2.0.0-p195.tar.bz2")
Process.wait pid
pid = spawn(RbConfig.ruby, "-eputs'Hello, world!
Process.wait 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 (de1
process group:
:pgroup => true or 0 : make a new process grd
: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 prc
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
string : redirect to fi]
[string] : redirect to fi]
[string, open_mode] : redirect to fi]
[string, open_mode, perm] : redirect to fi]
[:child, FD] : redirect to the
:close : close the file
FD is one of follows
:in : the file descriptor 0 which is tr
:out : the file descriptor 1 which is tr
:err : the file descriptor 2 which is tr
integer : the file descriptor of specified
io : the file descriptor specified as
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.
```

pid = spawn(command, :unsetenv_others=>true)
pid = spawn({"F00"=>"BAR"}, command, :unsetenv_o

```

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 ::setrlimit.
```

cur, max = Process.getrlimit(:CORE)
pid = spawn(command, :rlimit_core=>[0,max])
pid = spawn(command, :rlimit_core=>max)
pid = spawn(command, :rlimit_core=>0)

```

The :umask key in options specifies the umask.
```

pid = spawn(command, :umask=>077)

```

The :in, :out, :err, a fixnum, an IO 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, STDERR=>:out)
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 IO.popen. In this case, IO. popen redirects stdout to a pipe in the child process and [:child, :out] refers the redirected stdout.
```

io = IO.popen(["sh", "-c", "echo out; echo err
p io.read

```

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.

\section*{times \(\rightarrow\) aProcessTms}

Returns a Tms structure (see Process: :Tms) that contains user and system CPU times for this process, and also for children processes.
```

t = Process.times
[ t.utime, t.stime, t.cutime, t.cstime ]

```

\section*{uid \(\rightarrow\) fixnum}

Process::UID.rid \(\rightarrow\) fixnum
Process::Sys.getuid \(\rightarrow\) fixnum
Returns the (real) user ID of this process.
```

Process.uid

```

\section*{uid= user \(\rightarrow\) numeric}

Sets the (user) user ID for this process. Not available on all platforms.

\section*{wait() \(\rightarrow\) fixnum}
wait(pid=-1, flags=0) \(\rightarrow\) fixnum
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.

\section*{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 SystemCallError if there are no child processes. Not available on all platforms.
```

include Process
fork { exit 99 }
wait
\$?.exitstatus
pid = fork { sleep 3 }
Time.now
27440
2008-03-08
waitpid(pid, Process::WNOHANG)
Time.now
waitpid(pid, 0)
Time.now
27429
ni.l
2008-03-08
27440


## wait2(pid=-1, flags=0) $\rightarrow$ [pid, status] waitpid2(pid=-1, flags=0) $\rightarrow$ [pid, status]

Waits for a child process to exit (see ::waitpid for exact semantics) and returns an array containing the process id and the exit status (a Process:: Status object) of that child. Raises a SystemCallError if there are no child processes.

```
Process.fork { exit 99 }
pid, status = Process.wait2
pid
status.exitstatus
```


## waitall $\rightarrow$ [ [pid1,status1], ...]

Waits for all children, returning an array of pid/status pairs (where status is a Process::Status object).

| fork \{ sleep 0.1; exit 1 \} |
| :--- |
| fork $\{$ |
| p Process.waitall |

produces:

> [[30982, \#<Process::Status: pid 30982 exit 0>], $[30979, \#<$ Process: Status: pid 30979 exit 1>], $[30976, ~ \#<$ Process::Status: pid 30976 exit 2>]]

## wait() $\rightarrow$ fixnum <br> wait(pid=-1, flags=0) $\rightarrow$ fixnum <br> 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 SystemCallError if there are no child processes. Not available on all platforms.

| include Process  <br> fork \{ exit 99 \} $\#=>27429$ <br> wait $\#=>27429$ <br> \$?.exitstatus $\#=>99$ <br> pid = fork \{ sleep 3\} $\#=>27440$ <br> Time.now $\#=>2008-03-08$ <br> waitpid(pid, Process: :WNOHANG) $\#=>~ n i .1$ <br> Time.now $\#=>2008-03-08$ <br> waitpid(pid, 0) $\#=>27440$ <br> Time.now $\#=>2008-03-08$ |
| :--- | :--- |

## wait2(pid=-1, flags=0) $\rightarrow$ [pid, status]

waitpid2(pid=-1, flags $=0) \rightarrow$ [pid, status]
Waits for a child process to exit (see ::waitpid for exact semantics) and returns an array containing the process id and the exit status (a Process::Status object) of that child. Raises a SystemCallError if there are no child processes.

```
Process.fork { exit 99 }
pid, status = Process.wait2
pid
status.exitstatus
```


## 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

$\Rightarrow$ process.c

## Public Class Methods

## Process::GID.change_privilege(group) $\rightarrow$ 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.

```
[Process.gid, Process.egid]
Process::GID.change_privilege(33)
[Process.gid, Process.egid]
```

egid $\rightarrow$ fixnum
Process::GID.eid $\rightarrow$ fixnum
Process::Sys.geteid $\rightarrow$ fixnum
Returns the effective group ID for this process. Not available on all platforms.

## 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.from_name("wheel")
Process::GID.from_name("nosuchgroup")
```



## Process::GID.grant_privilege(group) $\rightarrow$

 fixnum
## Process::GID.eid = group $\rightarrow$ 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.

```
[Process.gid, Process.egid]
Process::GID.grant_privilege(31)
[Process.gid, Process.egid]
```


## 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, Process.egid]
Process::GID.re_exchange
[Process.gid, Process.egid]
```


## Process::GID.re_exchangeable? $\rightarrow$ true or false

Returns true if the real and effective group IDs of a process may be exchanged on the current platform.
gid $\rightarrow$ fixnum
Process::GID.rid $\rightarrow$ fixnum
Process::Sys.getgid $\rightarrow$ fixnum
Returns the (real) group ID for this process.
Process.gid

Process::GID.sid_available? $\rightarrow$ true or false
Returns true if the current platform has saved group ID functionality.

Process::GID.switch $\rightarrow$ fixnum
Process::GID.switch \{|| block\} $\rightarrow$ object
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 RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 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.
fork \{ exit 99 \}
Process.wait
\$?.class
\$?.to_i
\$? >> 8
\$?.stopped?
\$?.exited?
\$?.exitstatus

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

## Parent

## Object

## Public Instance Methods

stat \& num $\rightarrow$ fixnum
Logical AND of the bits in stat with num.

```
fork { exit 0x37 }
Process.wait
sprintf('%04x', $?.to_i)
sprintf('%04x', $? & 0x1e00)
```


## stat $==$ other $\rightarrow$ true or false

Returns true if the integer value of stat equals other.
stat $\gg$ num $\rightarrow$ fixnum
Shift the bits in stat right num places.

```
fork { exit 99 }
Process.wait
$?.to_i
$? >> 8
```

coredump? $\rightarrow$ true or false
Returns true if stat generated a coredump when it terminated. Not available on all platforms.

Returns true if stat exited normally (for example using an exit() call or finishing the program).

## exitstatus $\rightarrow$ fixnum or nil

Returns the least significant eight bits of the return code of stat. Only available if exited? is true.

```
fork { }
Process.wait
$?.exited?
$?.exitstatus
fork { exit 99 }
Process.wait
$?.exited? #=> true
$?.exitstatus #=> 99
#=> 26572
#=> 26572
#=> true
#=> 0
#=> 26573
#=> 26573
```


## inspect $\rightarrow$ string

Override the inspection method.

```
system("false")
p $?.inspect #=> "#<Process::Status: pid 12861
```

pid $\rightarrow$ fixnum

Returns the process ID that this status object represents.

```
fork { exit } ##> 26569
Process.wait ##> 26569
$?.pid
#=> 26569
```


## signaled? $\rightarrow$ true or false

Returns true if stat terminated because of an uncaught signal.

## 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 $\rightarrow$ fixnum or nil

Returns the number of the signal that caused stat to stop (or nil if self is not stopped).

## success? $\rightarrow$ true, false or nil

Returns true if stat is successful, false if not. Returns nil if exited? is not true.

## termsig $\rightarrow$ 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).

## to_i $\rightarrow$ fixnum

to_int $\rightarrow$ fixnum
Returns the bits in stat as a Fixnum. Poking around in these bits is platform dependent.

```
fork { exit 0xab }
Process.wait
sprintf('%04x', $?.to_i)
```


## to_s $\rightarrow$ string

Show pid and exit status as a string.

```
system("false")
p $?.to_s
```

Generated by RDoc 3.12.2. Generated with the Darkfish Rdoc Generator 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

egid $\rightarrow$ fixnum
Process::GID.eid $\rightarrow$ fixnum
Process::Sys.geteid $\rightarrow$ fixnum
Returns the effective group ID for this process. Not available on all platforms.

Process.egid

## euid $\rightarrow$ fixnum

Process::UID.eid $\rightarrow$ fixnum
Process::Sys.geteuid $\rightarrow$ fixnum
Returns the effective user ID for this process.

# gid $\rightarrow$ fixnum 

Process::GID.rid $\rightarrow$ fixnum
Process::Sys.getgid $\rightarrow$ fixnum
Returns the (real) group ID for this process.
Process.gid
uid $\rightarrow$ fixnum
Process::UID.rid $\rightarrow$ fixnum
Process::Sys.getuid $\rightarrow$ fixnum
Returns the (real) user ID of this process.

```
Process.uid
```


## Process::Sys.issetugid $\rightarrow$ 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) $\rightarrow$ nil

Set the effective group ID of the calling process to group. Not available on all platforms.

Process::Sys.seteuid(user) $\rightarrow$ nil
Set the effective user ID of the calling process to user. Not available on all platforms.

Process::Sys.setgid(group) $\rightarrow$ nil

Set the group ID of the current process to group. Not available on all platforms.

## Process::Sys.setregid(rid, eid) $\rightarrow$ 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.

## Process::Sys.setresgid(rid, eid, sid) $\rightarrow$ nil

 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.
## Process::Sys.setresuid(rid, eid, sid) $\rightarrow$ nil

 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) $\rightarrow$ 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) $\rightarrow$ nil

Set the real group ID of the calling process to group. Not available on all platforms.

## Process::Sys.setruid(user) $\rightarrow$ nil

Set the real user ID of the calling process to user. Not available on all platforms.

## Process::Sys.setuid(user) $\rightarrow$ nil

Set the user ID of the current process to user. Not available on all platforms.

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 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

$\Rightarrow$ process.c

## Public Class Methods

## Process::UID.change_privilege(user) $\rightarrow$ fixnum

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.

```
[Process.uid, Process.euid]
Process::UID.change_privilege(31)
[Process.uid, Process.euid]
```

euid $\rightarrow$ fixnum
Process::UID.eid $\rightarrow$ fixnum
Process::Sys.geteuid $\rightarrow$ fixnum
Returns the effective user ID for this process.

## 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.from_name("root") \#\#> 0
Process: :UID.from_name("nosuchuser")

## Process::UID.grant_privilege(user) $\rightarrow$ fixnum <br> Process::UID.eid= user $\rightarrow$ 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, Process.euid]
Process::UID.grant_privilege(31)
[Process.uid, Process.euid]
```


## Process::UID.re_exchange $\rightarrow$ fixnum

Exchange real and effective user IDs and return the new effective user ID. Not available on all platforms.

```
[Process.uid, Process.euid]
Process::UID.re_exchange
[Process.uid, Process.euid]
```


## Process::UID.re_exchangeable? $\rightarrow$ true or false <br> Returns true if the real and effective user IDs of a process may be exchanged on the current platform.

## uid $\rightarrow$ fixnum

Process::UID.rid $\rightarrow$ fixnum

## Process::Sys.getuid $\rightarrow$ fixnum

Returns the (real) user ID of this process.

Process::UID.sid_available? $\rightarrow$ true or false
Returns true if the current platform has saved user ID functionality.

## Process::UID.switch $\rightarrow$ fixnum <br> Process::UID.switch \{|| block\} $\rightarrow$ 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 RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

# class Process::Waiter 

## In Files

process.c

## Parent

## Thread

## Public Instance Methods

pid()
Generated by RDoc 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 $::$ srand.

The class method \#rand provides the base functionality of Kernel\#rand 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. Random 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

new(seed = Random.new_seed) $\rightarrow$ prng
Creates a new PRNG using seed to set the initial
state. If seed is omitted, the generator is initialized with ::new_seed.

See ::srand for more information on the use of seed values.
new_seed $\rightarrow$ integer
Returns an arbitrary seed value. This is used by ::new when no seed value is specified as an argument.

rand $\rightarrow$ float

## rand(max) $\rightarrow$ number

Alias of Random::DEFAULT.rand.

## srand(number = Random.new_seed) $\rightarrow$ 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

## prng1 == prng2 $\rightarrow$ 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
Random.new(1234) == Random.new(1234)
false
and have the same invocation history.

```
prng1 = Random.new(1234)
prng2 = Random.new(1234)
prng1 == prng2 # #> true
prng1.rand
0.1915194503788923
prng1 == prng2 # => false
prng2.rand # #> 0.1915194503788923
prng1 == prng2 # #> true
```


## bytes(size) $\rightarrow$ a_string

Returns a random binary string containing size bytes.


## rand $\rightarrow$ float

## rand(max) $\rightarrow$ number

When max is an Integer, rand returns a random integer greater than or equal to zero and less than max. Unlike Kernel\#rand, when max is a negative integer or zero, rand raises an ArgumentError.

```
prng = Random.new
prng.rand(100)
```

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 Range, 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.

## 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
prng1.rand(100)
prng2 = Random.new(prng1.seed)
prng2.rand(100)
```

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 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 : :new. 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 Comparable module is included so the $==$ method is defined in terms of <=>.

```
class Xs
represent a string
    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
```



An example of using xs to construct a range:


## In Files

range.c

## Parent

## Object

## Included Modules

## - Enumerable

## Public Class Methods

new(begin, end, exclude_end=false) $\rightarrow$ rng
Constructs a range using the given begin and end. If the exclude_end parameter is omitted or is false, the rng will include the end object; otherwise, it will be excluded.

## Public Instance Methods

## rng $==$ obj $\rightarrow$ true or false

Returns true only if obj is a Range, has equivalent begin and end items (by comparing them with ==), and has the same exclude_end? setting as the range.

```
(0..2) == (0..2)
(0..2) == Range.new(0,2)
(0..2) == (0...2)
```


## rng === obj $\rightarrow$ 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
when 51..75 then
when 76..100 then
end
```

produces:

```
high
```


## begin $\rightarrow$ obj

Returns the object that defines the beginning of the range.

```
(1..10).begin
```


## bsearch \{|obj| block \} $\rightarrow$ value

By using binary search, finds a value in range which meets the given condition in $\mathrm{O}(\log \mathrm{n})$ 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 $y(x<=y)$ so that:
the block returns a positive number for v if $\mathrm{v}<$ X ,
— the block returns zero for v if $\mathrm{x}<=\mathrm{v}<\mathrm{y}$, and
■ the block returns a negative number for v if y <= v.

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.

## cover?(obj) $\rightarrow$ true or false

Returns true if obj is between the begin and end of the range.

This tests begin <= obj <= end when exclude_end? is false and begin <= obj < end when exclude_end? is true.

```
("a".."z").cover?("c")
("a".."z").cover?("5")
("a".."z").cover?("cc")
```


## each \{| i | block \} $\rightarrow$ rng

## each $\rightarrow$ 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 TypeError is raised if the object does not have succ method defined (like Float).

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

[^0]```
(2.5..5).each {|n| print n, ' ' }
```

\# raises: TypeError: can't iterate from Float

## end $\rightarrow$ obj

Returns the object that defines the end of the range.

```
(1. .10).end
1 0
(1...10).end
1 0
```


## eql?(obj) $\rightarrow$ true or false

Returns true only if obj is a Range, has equivalent begin and end items (by comparing them with eql?), and has the same exclude_end? setting as the range.

```
(0. .2).eql?(0. . 2)
(0..2).eql?(Range.new(0, 2))
(0..2).eql?(0...2)
true
true
```


## exclude_end? $\rightarrow$ true or false

Returns true if the range excludes its end value.

```
(1. .5).exclude_end?
(1...5).exclude_end?
```

first $\rightarrow$ obj
first(n) $\rightarrow$ an_array
Returns the first object in the range, or an array of the first n elements.

```
(10..20).first
1 0
(10..20).first(3)
[10, 11, 12]
```


## hash $\rightarrow$ fixnum

Compute a hash-code for this range. Two ranges with equal begin and end points (using eql?), and the same exclude_end? value will generate the same hash-code.

See also Object\#hash.

## member?(obj) $\rightarrow$ true or false

include? (obj) $\rightarrow$ 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.
("a". ."z").include? ("g")
("a". ."z").include?("A")
("a". ."z").include?("cc")

## inspect $\rightarrow$ string

Convert this range object to a printable form (using inspect to convert the begin and end objects).
last $\rightarrow$ obj
last(n) $\rightarrow$ an_array
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.

```
(10..20).last
(10...20).last
(10..20).last(3)
(10...20).last(3)
```

$\max \rightarrow$ obj
$\max \{|\mathrm{a}, \mathrm{b}|$ block $\} \rightarrow$ obj
$\max (n) \rightarrow$ obj
$\max (\mathrm{n})\{|\mathrm{a}, \mathrm{b}|$ 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 <=> b.

```
(10. . 20).max
```

member?(obj) $\rightarrow$ true or false include?(obj) $\rightarrow$ 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.
("a". ."z").include?("g")
("a". "z").include?("A")
("a". "z").include?("cc")
min $\rightarrow$ obj
$\min \{|a, b|$ block $\} \rightarrow$ obj
$\min (n) \rightarrow$ array
$\min (n)\{|a, b| b l o c k\} \rightarrow$ 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 <=> b.

## size $\rightarrow$ num

Returns the number of elements in the range. Both the begin and the end of the Range must be Numeric, otherwise nil is returned.


## step(n=1) \{| obj | block \} $\rightarrow$ rng

 step( $n=1$ ) $\rightarrow$ an_enumeratorIterates 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 x
3 XXX
5 XXXXX
7 XXXXXXX
9 XXXXXXXXX
1 x
4 XXXX
7 XXXXXXX
1 0 ~ X X X X X X X X X X ~
```

See Range for the definition of class Xs.
to_s $\rightarrow$ string
Convert this range object to a printable form (using to_s to convert the begin and end objects).

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 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 

## In Files

error.c

## Parent

## StandardError

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 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 Rational, \#to_r, rationalize method or suffixing $r$ to a literal. The return values will be irreducible.

Rational(1)
Rational(2,
3 )
Rational(4, -6)
3.to_r

2/3r

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.

```
10.times.inject(0){|t,| t + 0.1}
10.times.inject(0){|t,| t + Rational('0.1')}
```

However, when an expression has inexact factor (numerical value or operation), will produce an inexact result.

```
Rational(10) / 3
Rational(10) / 3.0 ##> 3.3333333333333335
```

Rational(-8) ** Rational(1, 3)
\#=> (1.0000000000000002+1

In Files
rational.c

## Parent

## Numeric

## Public Instance Methods

## rat * numeric $\rightarrow$ numeric

Performs multiplication.

| Rational $(2,3)$ | $*$ Rational $(2,3)$ | $\#=>(4 / 9)$ |
| :--- | :--- | :--- |
| Rational $(900)$ | $*$ Rational 14$)$ | $\#=>(900 / 1)$ |
| Rational $(-2,9) *$ Rational $(-9,2)$ | $\#=>(1 / 1)$ |  |
| Rational $(9,8)$ | $* 4$ | $\#=>(9 / 2)$ |
| Rational $(20,9) * 9.8$ | $\#=>21.777777$ |  |

## rat ** numeric $\rightarrow$ numeric

## Performs exponentiation.

| Rational(2) | ** Rational(3) | \#=> | (8/1) |
| :---: | :---: | :---: | :---: |
| Rational(10) | ** -2 | > | (1/100) |
| Rational(10) | ** -2.0 | \#=> | 0.01 |
| Rational(-4) | ** Rational(1,2) | \#\#> | [1.22460635 |
| Rational(1, 2) | ** 0 | => | (1/1) |
| Rational(1, 2) | ** 0.0 | \#=> | 1.0 |
| 4. |  |  | - |

## rat + numeric $\rightarrow$ numeric

Performs addition.

| Rational $(2,3)+\operatorname{Rational}(2,3)$ | $\#=>(4 / 3)$ |
| :--- | :--- | :--- |
| Rational $(900)+$ Rational $(1)$ | $\#=>(900 / 1)$ |
| Rational $(-2,9)+\operatorname{Rational}(-9,2)$ | $\#=>(-85 / 18)$ |
| Rational $(9,8)+4$ | $\#=>(41 / 8)$ |
| Rational $(20,9)+9.8$ | $\#=>12.022222$ |
| 1 |  |

## rat - numeric $\rightarrow$ numeric

Performs subtraction.

| Rational $(2,3)$ | $-\operatorname{Rational}(2,3)$ | $\#=>(0 / 1)$ |
| :--- | :--- | :--- |
| Rational 900$)$ | $-\operatorname{Rational}(1)$ | $\#=>(899 / 1)$ |
| Rational $(-2,9)-\operatorname{Rational}(-9,2)$ | $\#=>(77 / 18)$ |  |
| Rational 9,8$)$ | -4 | $\#=>(23 / 8)$ |
| Rational(20, 9) -9.8 | $\#=>-7.577777$ |  |

rat / numeric $\rightarrow$ numeric quo(numeric) $\rightarrow$ numeric
Performs division.

| Rational(2, 3) | / Rational(2, 3) | => | (1/1) |
| :---: | :---: | :---: | :---: |
| Rational(900) | / Rational(1) | > | (900/1) |
| Rational(-2, 9) | / Rational(-9, 2) | \#=> | (4/81) |
| Rational(9, 8) | / 4 | +=> | (9/32) |

$\square$

## 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.

| Rational $(2,3)$ | $<=>$ Rational $(2,3)$ | $\#=>0$ |
| :--- | :--- | :--- |
| Rational $(5)$ | $<=>5$ | $\#=>0$ |
| Rational $(2,3)$ | $<=>$ Rational $(1,3)$ | $\#=>1$ |
| Rational 1,3$)$ | $<=>1$ | $\#=>-1$ |
| Rational(1,3) | $<=>~ 0.3$ | $\#=>1$ |

## rat == object $\rightarrow$ true or false

Returns true if rat equals object numerically.

| Rational $(2,3)$ | $==$ Rational $(2,3)$ | $\#=>$ true |
| :--- | :--- | :--- |
| Rational 5$)$ | $==5$ | $\#=>$ true |
| Rational $(0)$ | $==0.0$ | $\#=>$ true |
| Rational $\left(' 1 / 3^{\prime}\right)$ | $==0.33$ | $\#=>$ false |
| Rational('1/2') | $==~ ' 1 / 2^{\prime}$ | $\#=>$ false |

## ceil $\rightarrow$ integer <br> ceil(precision=0) $\rightarrow$ rational

Returns the truncated value (toward positive infinity).

| Rational(3).ceil |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| Rational(2, 3).ceil |  |  |  |  |  |
| Rational(-3, 2).ceil \#= |  |  |  |  |  |
| decimal - 123.456 |  |  |  |  |  |
| precision $\quad \hat{3}-2 \hat{-1} 00+\hat{1}+2$ |  |  |  |  |  |
|  |  |  |  |  |  |
| '\%f' \% Rational('-123.456').ceil(+1) \#=> "-123 |  |  |  |  |  |
| '\%f' \% Rational('-123.456').ceil(-1) \#=> "-120 |  |  |  |  |  |
|  |  |  |  |  |  |

## denominator $\rightarrow$ integer

Returns the denominator (always positive).

```
Rational(7).denominator
Rational(7, 1).denominator
Rational(9, -4).denominator
Rational(-2, -10).denominator
rat.numerator.gcd(rat.denominator)
```


## fdiv(numeric) $\rightarrow$ float

Performs division and returns the value as a float.

```
Rational(2, 3).fdiv(1)
#=> 0.66666666666664
Rational(2, 3).fdiv(0.5)
Rational(2).fdiv(3)
#=> 0.66666666666666
```



## floor $\rightarrow$ integer

floor(precision=0) $\rightarrow$ rational
Returns the truncated value (toward negative infinity).

```
Rational(3).floor
Rational(2, 3).floor
Rational(-3, 2).floor
```



```
'%f' % Rational('-123.456').floor(+1)
'%f' % Rational('-123.456').floor(-1)
```


## inspect $\rightarrow$ string

Returns the value as a string for inspection.

```
Rational(2).inspect
```

Rational(-8, 6).inspect
Rational('1/2').inspect

## numerator $\rightarrow$ integer

Returns the numerator.

> Rational(7).numerator
> Rational(7, 1).numerator
> Rational(9, -4).numerator
> Rational( $-2,-10) \cdot n u m e r a t o r$

## rat / numeric $\rightarrow$ numeric

quo(numeric) $\rightarrow$ numeric
Performs division.

rationalize $\rightarrow$ self

## 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.


## round $\rightarrow$ integer

round(precision=0) $\rightarrow$ rational
Returns the truncated value (toward the nearest
integer; 0.5 => 1; -0.5 => -1).


## to f $\rightarrow$ float

Return the value as a float.

```
Rational(2).to_f
Rational(9, 4).to_f
Rational(-3, 4).to_f #=> -0.75
Rational(20, 3).to_f ##> 6.666666666666667
```


## 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
Rational(300.6).to_i
#=> 300
Rational(98,71).to_i #=> 1
Rational(-30,2).to_i #=> -15
```

to_r $\rightarrow$ self
Returns self.

```
Rational(2).to_r
#=> (2/1)
Rational(-8, 6).to_r #=> (-4/3)
```


## to_s $\rightarrow$ string

Returns the value as a string.

| Rational(2).to_s | \#=> "2/1" |
| :---: | :---: |
| Rational(-8, 6).to_s | \#=> "-4/3" |
| Rational('1/2').to_s | \#=> "1/2" |

truncate $\rightarrow$ integer
truncate(precision=0) $\rightarrow$ rational
Returns the truncated value (toward zero).


Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

# class Rational::compatible 

## In Files

rational.c

## Parent

Object
Generated by RDoc 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 $/ \ldots$. and \%r $\{\ldots\}$ 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:

```
/hay/ =~ 'haystack'
/y/.match('haystack')
```

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')
```

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 \#match

Pattern matching may be achieved by using =~ operator or \#match method.

## =~ operator

$=\sim$ 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 String so the order of String and Regexp 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 $=\sim$ operator with a String and Regexp the \$~ global variable is set after a successful match.
\$~ holds a MatchData object. ::last_match is equivalent to \$~.

## \#match 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＞l｜ ＜／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")

Arbitrary Ruby expressions can be embedded into patterns with the \＃\｛．．．\} construct.
place＝＂東京都＂
／\＃\｛place\}/.match("Go to 東京都")

## 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 $b$.


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 $[a-d w-z]$. The order in which ranges or individual characters appear inside a character class is irrelevant.

```
/[0-9a-f]/.match('9f')
/[gf]/.match('gf')
```

If the first character of a character class is a caret ( $\wedge$ ) the class is inverted: it matches any character except those named.

```
/[^a-eg-z]/.match('f')
```

A character class may contain another character class. By itself this isn't useful because [a-z[09]] 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:

## /[a-w\&\&[^c-g]z]/ \# ([a-w] AND ([^c-g] OR z))

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)
-     - ww/ - A word character ([a-zA-z0-9_])
- /\w/ - A non-word character ([^a-zA-z0-9_]). Please take a look at Bug \#4044 if using / w / with the /i modifier.
- $\ \mathrm{~d} /$ - A digit character ([0-9])
- $\ \mathrm{D} /$ - A non-digit character ([^0-9])
- $\triangle h /-A$ hexdigit character ([0-9a-fA-F])
- $\$ н/ - A non-hexdigit character ([^0-9a-fA-F])
- /\s/ - A whitespace character: /[ \t\r\n\f]/
- /\s/ - A non-whitespace character: /[^ \t $\backslash \mathrm{r} \backslash n \backslash f] /$

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':
"Hello".match(/[[:upper:]]+[[:lower:]]+l\{2\}o/
1
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 $\backslash n$; outside of the pattern use MatchData[n].
'at' is captured by the first group of parentheses, then referred to later with $\backslash 1$ :

\#match returns a MatchData 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:

## /[aeiou]\w\{2\}/.match("Caenorhabditis elegans"

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 $\backslash 2$ :
/I(n)ves(ti)ga\2ons/.match("Investigations")

$\downarrow$

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

```
/"(?>.*)"/.match('"Quote"')
```


## Subexpression Calls

The $\backslash \mathrm{g}<$ 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
7. 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 (I) combines two expressions into a single one that matches either of the expressions. Each expression is an alternative.


## Character Properties

The $\backslash p\}$ construct matches characters with the named property, much like POSIX bracket classes.

- $\triangle p\{A l n u m\} /-$ Alphabetic and numeric character
- $\triangle p\{A l p h a\} /-A l p h a b e t i c ~ c h a r a c t e r ~$
- /\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
- $\triangle$ p $\{$ Print $\} /$ - Like $\backslash p\{G r a p h\}$, but includes the space character
- $\$ p\{Punct $\} /$ - Punctuation character
- /\p\{Space\}/ - Whitespace character ([:blank:], newline, carriage return, etc.)
- ハp\{Upper\}/- Uppercase alphabetical
- $\triangle$ p\{XDigit $\} /$ - Digit allowed in a hexadecimal number (i.e., 0-9a-fA-F)
- $\triangle$ p\{Word\}/-A member of one of the following Unicode general category Letter, Mark, Number, Connector_Punctuation
－$\$ p\｛ASCII\}/ - A character in the ASCII character set
－$\backslash$ 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 $\backslash p\{A b\}$ where $A b$ is the category＇s abbreviation as described below：
－$八$ p\｛L\}/ - 'Letter'
－ハp\｛L1\}/ - 'Letter: Lowercase'
－$\wedge p\{L m\} /$－＇Letter：Mark＇
－ハp\｛Lo\}/ - 'Letter: Other'
－$\$ p\｛Lt\}/ - 'Letter: Titlecase'
－／\p\｛Lu\}/ - 'Letter: Uppercase
－／\p\｛Lo\}/ - 'Letter: Other'
－$\triangle \mathrm{p}\{\mathrm{M}\} /$－＇Mark＇
－／\p\｛Mn\}/ - 'Mark: Nonspacing'
－$\triangle \mathrm{p}\{\mathrm{Mc}\} /$－＇Mark：Spacing Combining＇
－$\$ р\｛ме\}/ - 'Mark: Enclosing'
－／ $\mathrm{p}\{\mathrm{N}\} /$－＇Number＇
－$/ \backslash\{\{N d\} /-$＇Number：Decimal Digit＇
－／ $\mathrm{p}\{\mathrm{N} 1\} /$－＇Number：Letter＇
－$\triangle \mathrm{p}\{\mathrm{No}\} /$－＇Number：Other＇
－$\ \mathrm{p}\{\mathrm{P}\} /$－＇Punctuation＇
－$\ p\{$ Pc $\} /$－＇Punctuation：Connector＇
- $\ \mathrm{p}\{\mathrm{Pd}\} /$ - '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'
- ハp\{Sc\}/ - 'Symbol: Currency'
- ハ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'
- $\triangle \mathrm{p}\{\mathrm{c}\} /$ - 'Other'
- $\wedge p\{c c\} /-$-Other: Control'
- / p\{cff\}/ - 'Other: Format'
- $\triangle \mathrm{p}\{\mathrm{Cn}\} /$ - 'Other: Not Assigned'
- $\$ p\{co\}/ - 'Other: Private Use'
- / p\{ccs\}/- '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 ( $\wedge$ ).

Letter 'A' is not in the Unicode LI (Letter; Lowercase) category, so this match succeeds:

[^1]
## 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.
- $\backslash z$ - 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:

```
/real/.match("surrealist")
```

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")
```

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:


## 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:


## Free-Spacing Mode and Comments

As mentioned above, the $\times$ option enables freespacing 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 <br>, 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:

## $r=$ Regexp.new("a".force_encoding("iso-8859-1 "a\u3042" <br> (ISO-8859-1 regexp with UTF-8 string)



## Special global variables

Pattern matching sets some global variables :

- \$~ is equivalent to ::last_match;
- \$\& 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 as, a d, 4 as, and a c.


The following patterns match instantly as you would expect:
$/(b \mid a) /=\sim s$
$/(b \mid a+) /=\sim s$
$/(b \mid a+)^{*} /=\sim s$

However, the following pattern takes appreciably longer:

$$
/(\mathrm{b} \mid \mathrm{a}+)^{*} \mathrm{c} /=\sim \mathrm{s}
$$

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 as match the string, but this prevents the 29 mandatory as 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 \#options and ::new

## Public Class Methods

## compile(*args)

Synonym for Regexp.new
escape(str) $\rightarrow$ string
quote(str) $\rightarrow$ 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('\*?{}.')
```


## last_match $\rightarrow$ matchdata

last_match(n) $\rightarrow$ str
The first form returns the MatchData 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 MatchData object. $n$ can be a string or symbol to reference a named capture.

Note that the ::last_match is local to the thread and method scope of the method that did the pattern match.


## new(regexp) $\rightarrow$ regexp

compile(string, [options [, kcode]]) $\rightarrow$ regexp compile(regexp) $\rightarrow$ regexp

Constructs a new regular expression from pattern, which can be either a String or a Regexp (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 Fixnum, 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.


## escape(str) $\rightarrow$ string

quote(str) $\rightarrow$ 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 Regexp, using to_regexp method. Returns converted regexp or nil if obj cannot be converted for any reason.

```
Regexp.try_convert(/re/)
Regexp.try_convert("re")
o = Object.new
Regexp.try_convert(o)
def o.to_regexp() /foo/ end
Regexp.try_convert(o)
```

```
union(pat1, pat2, ...) > new_regexp
union(pats_ary) }->\mathrm{ 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 Regexp 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.

```
Regexp.union
Regexp.union("penzance")
Regexp.union("a+b*c")
Regexp.union("skiing", "sledding")
Regexp.union(["skiing", "sledding"])
Regexp.union(/dogs/, /cats/)
```

Note: the arguments for ::union will try to be converted into a regular expression literal via to_regexp.

## Public Instance Methods

sxp $==$ other_rxp $\rightarrow$ true or false
eql?(other_rxp) $\rightarrow$ 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
```


## rxp $=\sim$ str $\rightarrow$ integer or nil

Match—Matches rxp against str.

```
/at/ =~ "input data"
/ax/ =~ "input data" ##=> nil.
```

If =~ is used with a regexp literal with named captures, captured strings (or nil) is assigned to local variables named by the capture names.

```
/(?<lhs>\W+)\s*=\s*(?<rhs>\w+)/ =~ " x = y "
p lhs
p rhs
```

If it is not matched, nil is assigned for the variables.

```
/(?<lhs>\w+)\s*=\s*(?<rhs>\w+)/ =~ " x = "
p lhs
p rhs
```

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.

```
re = /(?<lhs>\w+)\s*=\s*(?<rhs>\w+)/
re =~ " x = y
p lhs # undefined local variable
p rhs # undefined local variable
```

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
```


## casefold? $\rightarrow$ true or false

Returns the value of the case-insensitive flag.

```
/a/.casefold?
/a/.casefold?
/(?i:a)/.casefold?
```

Returns the Encoding object that represents the encoding of obj.

## rxp $==$ other_rxp $\rightarrow$ true or false <br> eql?(other_rxp) $\rightarrow$ 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.


## fixed_encoding? $\rightarrow$ true or false

Returns false if rxp is applicable to a string with any ASCII compatible encoding. Returns true otherwise.

```
r = /a/
r.fixed_encoding?
r =~ "\u{6666} a"
r =~ "\xa1\xa2 a".force_encoding("euc-jp")
r =~ "abc".force_encoding("euc-jp")
r = /a/
r.fixed_encoding?
r.encoding
r =~ "\u{6666} a"
r =~ "\xa1\xa2".force_encoding("euc-jp")
r =~ "abc".force_encoding("euc-jp")
r = /\u{6666}/
r.fixed_encoding?
r.encoding
r =~ "\u{6666} a"
r =~ "\xa1\xa2".force_encoding("euc-jp")
r =~ "abc".force_encoding("euc-jp")
```


## hash $\rightarrow$ fixnum

Produce a hash based on the text and options of this regular expression.

See also Object\#hash.

## inspect $\rightarrow$ string

Produce a nicely formatted string-version of $r x p$.
Perhaps surprisingly, \#inspect actually produces the more natural version of the string than \#to_s.

```
/ab+c/x.inspect
```


## 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.

```
/(.)(.)(.)/.match("abc")[2]
/(.)(.)/.match("abc", 1)[2]
```

If a block is given, invoke the block with MatchData 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.

## named_captures $\rightarrow$ hash

Returns a hash representing information about named captures of $r x p$.
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
```


## names $\rightarrow$ [name1, name2, ...]

Returns a list of names of captures as an array of strings.


## options $\rightarrow$ 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.

```
Regexp::IGNORECASE
Regexp::EXTENDED
Regexp::MULTILINE
/cat/.options
/cat/x.options
Regexp.new('cat', true).options
\\xa1\xa2/.options
r = /cat/x
Regexp.new(r.source, r.options)
```


## source $\rightarrow$ str

Returns the original string of the pattern.

```
/ab+c/x.source
```

Note that escape sequences are retained as is.

```
\x20\+/.source
```


## to_s $\rightarrow$ 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 $r x p$.

```
r1 = /ab+c/x
```

```
s1 = r1.to_s
r2 = Regexp.new(s1)
r1 == r2
r1.source
r2.source
```

(9) $\operatorname{rxp} \rightarrow$ integer or nil

Match—Matches rxp against the contents of \$_.
Equivalent to $r \times p=\sim$ \$_.

```
$_ = "input data"
~ /at/
```

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## class RegexpError

Raised when given an invalid regexp expression.

```
Regexp.new("?")
```

raises the exception:

RegexpError: target of repeat operator is not

## In Files

re.c

## Parent

## StandardError

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## class Ripper

## In Files

parse.c

## Parent

## Object

## Constants

## Version

version of Ripper

## Public Class Methods

new(src, filename="(ripper)", lineno=1) $\rightarrow$ ripper
Create a new Ripper object. src must be a String, an IO, or an Object which has gets method.

This method does not starts parsing. See also \#parse and \#parse.

## Public Instance Methods

## ripper\#column $\rightarrow$ Integer

Return column number of current parsing line. This number starts from 0 .
ripper\#encoding $\rightarrow$ encoding
Return encoding of the source.
ripper\#end_seen? $\rightarrow$ Boolean
Return true if parsed source ended by +__END__+.
ripper\#error? $\rightarrow$ Boolean
Return true if parsed source has errors.
ripper\#filename $\rightarrow$ String
Return current parsing filename.
ripper\#lineno $\rightarrow$ 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 $\rightarrow$ true or false
Get yydebug.
yydebug = flag
Set yydebug.

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## class RubyVM

::RubyVM

## In Files

iseq.c
$\square$ 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 $\rightarrow$ Hash<br>stat(hsh) $\rightarrow$ hsh<br>stat(Symbol) $\rightarrow$ Numeric

Returns a Hash containing implementationdependent counters inside the VM.

This hash includes information about method/constant cache serials:

```
{
    :global_method_state=>251,
    :global_constant_state=>481,
    :class_serial=>9029
}
```

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 RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

# class RubyVM::Env 

::RubyVM::Env

## In Files

iseq.c

## Parent

Object
Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## class <br> 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

compile(source[, file[, path[, line[,
options][]]]) $\rightarrow$ iseq new(source[, file[, path[, line[, options]]]]) $\rightarrow$ iseq
Takes source, a String 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]) $\rightarrow$ iseq

Takes file, a String with the location of a Ruby source file, reads, parses and compiles the file, and returns iseq, the compiled InstructionSequence 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 $\rightarrow$ 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 ::new, ::compile and ::compile_file.
disasm(body) $\rightarrow$ str disassemble(body) $\rightarrow$ str
Takes body, a Method or Proc 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(:
```

$\square \square$

Produces:


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: redo st: 0000 ed: 0012 sp: 0000
| catch type: next st: 0000 ed: 0012 sp: 0000
|----------------------------------------------
local table (size: 2, argc: 0 [opts: 0, rest: -1,
[ 2] num
0 0 0 0 ~ t r a c e ~ 1 ~
0 0 0 2 \text { putobject 1}
0 0 0 4 ~ p u t o b j e c t ~ 2 , ~
0006 opt_plus <ic:1>
0008 dup
0 0 0 9 ~ s e t l o c a l ~ n u m , ~ 0 ~
0 0 1 2 ~ l e a v e
```

disasm(body) $\rightarrow$ str disassemble(body) $\rightarrow$ str
Takes body, a Method or Proc 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(:
```

4 」

## Produces:

```
== disasm: <RubyVM::InstructionSequence:hello@/tn
0000 trace 8
0 0 0 2 ~ t r a c e ~ 1 ~
0 0 0 4 ~ p u t s e l f ~ f
0 0 0 5 ~ p u t s t r i n g ~ " h e l l o , ~ w o r l d " ~
0 0 0 7 ~ s e n d ~ : p u t s , ~ 1 , ~ n i l , ~ 8 , ~ < i c : 0 > ~
0 0 1 3 ~ t r a c e
    16
```

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: redo st: 0000 ed: 0012 sp: 0000
| catch type: next st: 0000 ed: 0012 sp: 0000
lo---------------------------------------------
[ 2] num
0 0 0 0 ~ t r a c e ~ 1 ~
0 0 0 2 ~ p u t o b j e c t ~ 1 ~
0 0 0 4 ~ p u t o b j e c t ~ 2 ~
0006 opt_plus <ic:1>
0008 dup
0 0 0 9 ~ s e t l o c a l ~ n u m , ~ 0 ~
0 0 1 2 ~ l e a v e
```


compile(source[, file[, path[, line[, options]]]]]) $\rightarrow$ iseq new(source[, file[, path[, line[, options]]]]) $\rightarrow$ iseq
Takes source, a String 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=.


## of(p1)

Returns the instruction sequence containing the given proc or method.

For example, using irb:

```
# a proc
> p = proc { num = 1 + 2 }
> RubyVM::InstructionSequence.of(p)
> #=> <RubyVM::InstructionSequence:block in irb
# for a method
> def foo(bar); puts bar; end
> RubyVM::InstructionSequence.of(method(:foo))
> #=> <RubyVM::InstructionSequence:foo@(irb)>
```



Using ::compile_file:

```
#/tmp/iseq_of.rb
def hello
    puts "hello, world"
end
$a_global_proc = proc { str = 'a' + 'b' }
# in irb
> require '/tmp/iseq_of.rb'
# first the method hello
> RubyVM::InstructionSequence.of(method(:hello))
> #=> #<RubyVM::InstructionSequence:0x007fb73d7c
    then the global proc
> RubyVM::InstructionSequence.of($a_global_proc)
```



## 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:

```
# /tmp/method.rb
def hello
    puts "hello, world"
end
# in irb
> iseq = RubyVM::InstructionSequence.compile_fil¢
> iseq.absolute_path #=> /tmp/method.rb
```



## base_label()

Returns the base label of this instruction sequence.
For example, using irb:


Using ::compile_file:

```
# /tmp/method.rb
def hello
    puts "hello, world"
end
```

> iseq = RubyVM::InstructionSequence.compile_fil¢ > iseq.base_label \#=> <main>
disasm $\rightarrow$ str
disassemble $\rightarrow$ str
Returns the instruction sequence as a string in
human readable form.


## Produces:

| == disasm: <RubyVM: | InstructionSequence : <compiled |
| :--- | :--- |
| 0000 trace | 1 |
| 0002 putobject | 1 |
| 0004 putobject | 2 |
| 0006 opt_plus | [ic:1](ic:1) |
| 0008 leave |  |

disasm $\rightarrow$ str
disassemble $\rightarrow$ str
Returns the instruction sequence as a string in human readable form.


## Produces:



## eval $\rightarrow$ obj

Evaluates the instruction sequence and returns the result.

RubyVM::InstructionSequence.compile("1 + 2").eva-

## 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 label and path.

## 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:

```
iseq = RubyVM::InstructionSequence.compile('num
#=> <RubyVM::InstructionSequence:<compiled>@<com
iseq.label
```



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 TypeError 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:

to_a $\rightarrow$ ary
Returns an Array 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) |

## 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 Hash object containing parameter information.

More info about these values can be found in 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 RDoc 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 or 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 RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 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:

SecurityError: Insecure: Insecure operation

## In Files

error.c

## Parent

## Exception

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 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. Signal delivery semantics may also vary between systems; in particular signal delivery may not always be reliable.

## In Files

signal.c

## Public Class Methods

list $\rightarrow$ a_hash
Returns a list of signal names mapped to the corresponding underlying signal numbers.

signame(signo) $\rightarrow$ string
convert signal number to signal name

produces:

```
INT
```

trap( signal, command ) $\rightarrow$ obj

## 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:

## 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) $\rightarrow$ signal_exception new(sig_number [, name]) $\rightarrow$ signal_exception

Construct a new SignalException object. sig_name should be a known signal name.

## Public Instance Methods

signo $\rightarrow$ num
Returns a signal number.

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## class StandardError

The most standard error types are subclasses of StandardError. A rescue clause without an explicit Exception class will rescue all StandardErrors (and only those).

def foo<br>raise "Oups"<br>end<br>foo rescue "Hello"

Me110

On the other hand:

```
require 'does/not/exist' rescue "Hi"
```

raises the exception:

```
LoadError: no such file to load -- does/not/e
```


## In Files

error.c

## Parent

## Exception

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## class Stoplteration

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

result $\rightarrow$ value
Returns the return value of the iterator.

```
o = Object.new
```

```
def o.each
    yield 1
    yield 2
    yield 3
    1 0 0
end
e = o.to_enum
puts e.next
puts e.next
puts e.next
begin
    e.next
rescue StopIteration => ex
    puts ex.result
    #=> 100
end
```

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

## class String

A string object holds and manipulates an arbitrary sequence of bytes, typically representing characters. String 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
$\Rightarrow$ string.c
- transcode.c


## Parent

## Object

## Included Modules

Comparable

## Public Class Methods

```
new(str="") \(\rightarrow\) new_str
```

Returns a new string object containing a copy of str.

## try_convert(obj) $\rightarrow$ string or nil

Try to convert obj into a String, using \#to_str 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

## str \% arg $\rightarrow$ 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.

str * integer $\rightarrow$ new_str
Copy - Returns a new String containing integer copies of the receiver. integer must be greater than or equal to 0 .

```
"Ho! " * 3
"Ho! " * 0
```

str + other_str $\rightarrow$ new_str
Concatenation-Returns a new string containing other_str concatenated to str.

str << integer $\rightarrow$ str
concat(integer) $\rightarrow$ str
str << obj $\rightarrow$ str
(8) concat(obj) $\rightarrow$ 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.

```
a = "hello "
a << "world"
a.concat(33)
```

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 Comparable. The method String\#== does not use Comparable\#==.
"abcdef" <=> "abcde"
"abcdef" <=> "abcdef"
"abcdef" <=> "abcdefg"
"abcdef" <=> "ABCDEF"
str $==$ obj $\rightarrow$ true or false
str $===$ obj $\rightarrow$ true or false

## Equality

Returns whether str == obj, similar to Object\#==.
If obj is not an instance of String but responds to to_str, then the two strings are compared using case equality Object\#===.

Otherwise, returns similarly to \#eql?, comparing length and content.
str $==$ obj $\rightarrow$ true or false
str $===$ obj $\rightarrow$ true or false

## Equality

Returns whether str == obj, similar to Object\#==.
If obj is not an instance of String but responds to to_str, then the two strings are compared using case equality Object\#===.

Otherwise, returns similarly to \#eql?, comparing length and content.
str $=\sim$ obj $\rightarrow$ 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/
"cat o' 9 tails" =~ 9
```


## str[index] $\rightarrow$ new_str or nil

str[start, length] $\rightarrow$ new_str or nil
str[range] $\rightarrow$ new_str or nil
str[regexp] $\rightarrow$ new_str or nil
str[regexp, capture] $\rightarrow$ new_str or nil
str[match_str] $\rightarrow$ new_str or nil
slice(index) $\rightarrow$ new_str or nil
slice(start, length) $\rightarrow$ new_str or nil
slice(range) $\rightarrow$ new_str or nil
slice(regexp) $\rightarrow$ new_str or nil
slice(regexp, capture) $\rightarrow$ new_str or nil
slice(match_str) $\rightarrow$ 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[2..3]
a[-3, 2]
a[7..-2]
a[-4..-2]
a[-2..-4]
a[11, 0]
a[11]
    #=> nil
a[12, 0]
    #=> nil
a[12..-1]
a[/[aeiou](.)\1/]
a[/[aeiou](.)\1/, 0]
a[/[aeiou](.)\1/, 1]
a[/[aeiou](.)\1/, 2] ##> ni.l
```



## 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? $\rightarrow$ true or false

Returns true for a string which has only ASCII characters.

[^2]b $\rightarrow$ str
Returns a copied string whose encoding is ASCII8BIT.

## bytes $\rightarrow$ 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 $\rightarrow$ integer

Returns the length of str in bytes.

```
"\x80\u3042".bytesize
"hello".bytesize
```


## byteslice(fixnum) $\rightarrow$ new_str or nil byteslice(fixnum, fixnum) $\rightarrow$ new_str or nil 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.

```
"hello".byteslice(1)
"hello".byteslice(-1)
"hello".byteslice(1, 2)
"\x80\u3042".byteslice(1, 3) #=> "\u3042'
"\x03\u3042\xff".byteslice(1..3) ##> "\u3042"
```


## 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
```


## capitalize! $\rightarrow$ str or nil

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.

```
a = "hello"
a.capitalize! ##> "Hello'
a
a.capitalize! #=> nil
```


## casecmp(other_str) $\rightarrow-1,0,+1$ or nil

Case-insensitive version of String\#<=>.

```
"abcdef".casecmp("abcde")
"aBcDeF".casecmp("abcdef")
"abcdef".casecmp("abcdefg")
"abcdef".casecmp("ABCDEF")
```


## center(width, padstr=' ') $\rightarrow$ new_str

Centers str in width. If width is greater than the length of str, returns a new String of length width with str centered and padded with padstr; otherwise, returns str.

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.

## 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 $\backslash n, \backslash r$, and $\backslash r \backslash n$ ). 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('')
```

```
hello'
hello"
hello"
hello\n'
hello"
"hello
"he"
"hello"
```


## chomp!(separator=\$/) $\rightarrow$ str or nil

Modifies str in place as described for string\#chomp, returning str, or nil if no modifications were made.
chop $\rightarrow$ new_str
Returns a new String with the last character removed. If the string ends with $\backslash r \backslash n$, 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.

```
"string\r\n".chop
"string\n\r".chop
"string\n".chop
"string".chop
"x".chop.chop
```


## chop! $\rightarrow$ str or nil

Processes str as for String\#chop, returning str, or nil if $s t r$ is the empty string. See also string\#chomp!.
chr $\rightarrow$ string
Returns a one-character string at the beginning of the string.

```
a = "abcde"
a.chr
```

clear $\rightarrow$ string

Makes string empty.

```
a = "abcde"
```


## codepoints $\rightarrow$ 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 $\rightarrow$ str
concat(integer) $\rightarrow$ str
str << obj $\rightarrow$ str
concat(obj) $\rightarrow$ 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.

```
a = "hello "
a << "world"
a.concat(33)
```


## count([other_str]+) $\rightarrow$ 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 $\wedge$ 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"
```

```
a.count "lo", "o"
a.count "hello", "^l"
a.count "ej-m"
"hello^world".count "\\^aeiou"
"hello-world".count "a\\-eo"
c = "hello world\\r\\n"
c.count "\\"
c.count "\\A"
c.count "X-\\w"
```


## crypt(salt_str) $\rightarrow$ 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.

## delete([other_str]+) $\rightarrow$ 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" |

Performs a delete operation in place, returning str, or nil if $s t r$ was not modified.

## downcase $\rightarrow$ 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

## 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.

## dump $\rightarrow$ new_str

Produces a version of str with all non-printing characters replaced by \nnn notation and all special characters escaped.

```
"hello \n ''".dump
```


## each_byte \{|fixnum| block \} $\rightarrow$ str

each_byte $\rightarrow$ 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 \} $\rightarrow$ str

each_char $\rightarrow$ 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 ello
each_codepoint \{|integer| block \} $\rightarrow$ str
each_codepoint $\rightarrow$ 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:
1041011081081111593
each_line(separator=\$/) \{|substr| block \} $\rightarrow$,
str
each_line(separator=\$/) $\rightarrow$ an_enumerator
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"
"l"
"o\nworl"
"d"
Example three
"hello\n\n\n"
"world"
```


## empty? $\rightarrow$ true or false

Returns true if str has a length of zero.

```
"hello".empty? ##=> false
" ".empty? ##=> false
"".empty? ##> true
```

encode(encoding [, options] ) $\rightarrow$ str encode(dst_encoding, src_encoding [, options] ) $\rightarrow$ str encode([options]) $\rightarrow$ 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 Hash gives details for conversion and can have the following keys:
:invalid
If the value is : replace, encode replaces invalid byte sequences in str with the replacement character. The default is to raise the Encoding::InvalidByteSequenceError exception

## :undef

If the value is : replace, encode 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 Hash, a Proc, a Method, 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 encode replaces undefined characters with their (upper-case hexadecimal) numeric character references. ' $\&$ ', '<', and '>' are converted to "\&", "\<", and "\>", respectively. If the value is :attr, encode 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.

## encode!(encoding [, options] ) $\rightarrow$ str encode!(dst_encoding, src_encoding [, options] ) $\rightarrow$ 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 Hash gives details for conversion. See \#encode for details. Returns the string even if no changes were made.

## encoding $\rightarrow$ encoding

Returns the Encoding object that represents the encoding of obj.

## end_with?([suffixes]+) $\rightarrow$ true or false

Returns true if str ends with one of the suffixes given.

## eql?(other) $\rightarrow$ true or false

Two strings are equal if they have the same length and content.

## force_encoding(encoding) $\rightarrow$ str

Changes the encoding to encoding and returns self.

## getbyte(index) $\rightarrow 0$.. 255

returns the indexth byte as an integer.
gsub(pattern, replacement) $\rightarrow$ new_str gsub(pattern, hash) $\rightarrow$ new_str gsub(pattern) \{|match| block \} $\rightarrow$ new_str gsub(pattern) $\rightarrow$ 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. '<br>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 $\backslash \backslash d$, where $d$ is a group number, or $\backslash \backslash k<n>$, 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.


## gsub!(pattern, replacement) $\rightarrow$ str or nil

 gsub!(pattern) \{|match| block \} $\rightarrow$ str or nil gsub!(pattern) $\rightarrow$ an_enumeratorPerforms 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.

## hash $\rightarrow$ fixnum

Return a hash based on the string's length, content and encoding.

See also Object\#hash.
hex $\rightarrow$ integer
Treats leading characters from str as a string of hexadecimal digits (with an optional sign and an optional $0 x$ ) and returns the corresponding number. Zero is returned on error.

```
"0x0a".hex
"-1234".hex
"0".hex
"wombat".hex
```


## include? other_str $\rightarrow$ 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]) $\rightarrow$ fixnum or nil index(regexp [, offset]) $\rightarrow$ 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.

```
"hello".index('e')
"hello".index('lo')
"hello".index('a')
"hello".index(e)
"hello".index(/[aeiou]/, -3)
```

replace(other_str) $\rightarrow$ str

Replaces the contents and taintedness of str with the corresponding values in other_str.

```
s = "hello"
s.replace "world"
```

insert(index, other_str) $\rightarrow$ 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 aString so that it starts at the given index.


## inspect $\rightarrow$ string

Returns a printable version of str, surrounded by quote marks, with special characters escaped.

```
str = "hello"
str[3] = "\b"
str.inspect
```


## intern $\rightarrow$ symbol

to_sym $\rightarrow$ 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
```

length $\rightarrow$ integer
size $\rightarrow$ integer
Returns the character length of str.
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.
ljust(integer, padstr=' ') $\rightarrow$ 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.


Istrip $\rightarrow$ new_str
Returns a copy of str with leading whitespace removed. See also String\#rstrip and String\#strip.

```
" hello ".lstrip
"hello".lstrip
```


## Istrip! $\rightarrow$ self or nil

Removes leading whitespace from str, returning nil if no change was made. See also String\#rstrip! and String\#strip!.

```
    " hello ".lstrip
"hello".lstrip!
```


## match(pattern) $\rightarrow$ matchdata or nil

## match(pattern, pos) $\rightarrow$ 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.

```
'hello'.match('(.)\1')
'hello'.match('(.)\1')[0]
'hello'.match(/(.)\1/)[0]
'hello'.match('xx')
```

If a block is given, invoke the block with MatchData if match succeed, so that you can write

```
str.match(pat) {|m| ...}
```

instead of
if $m=s t r . m a t c h(p a t)$
end

The return value is a value from block execution in this case.
succ $\rightarrow$ new_str
(9) next $\rightarrow$ 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.

succ! $\rightarrow$ str
next! $\rightarrow$ str
Equivalent to String\#succ, but modifies the receiver in place.

## 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.


## ord $\rightarrow$ integer

Return the Integer ordinal of a one-character string.

```
"a".ord
```


## partition(sep) $\rightarrow$ [head, sep, tail]

 partition(regexp) $\rightarrow$ [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) $\rightarrow$ str

Prepend-Prepend the given string to str.

```
a = "world"
a.prepend("hello ") ##=> "hello world"
a
    he110 world"
```

replace(other_str) $\rightarrow$ str

Replaces the contents and taintedness of str with the corresponding values in other_str.

```
s = "hello"
s.replace "world"
```

reverse $\rightarrow$ new_str

Returns a new string with the characters from str in reverse order.

## reverse! $\rightarrow$ str

Reverses str in place.

## rindex(substring [, fixnum]) $\rightarrow$ fixnum or nil rindex(regexp [, fixnum]) $\rightarrow$ 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 searchcharacters beyond this point will not be considered.

```
"hello".rindex('e')
"hello".rindex('l')
"hello".rindex('a')
"hello".rindex(e)
"hello".rindex(/[aeiou]/, -2)
```

rjust(integer, padstr=' ') $\rightarrow$ 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) $\rightarrow$ [head, sep, tail] rpartition(regexp) $\rightarrow$ [head, match, tail] 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.

rstrip $\rightarrow$ new_str
Returns a copy of str with trailing whitespace removed. See also String\#lstrip and String\#strip.

| " hello ".rstrip <br> "hello".rstrip | $\begin{aligned} & : \#=>\text { nello } \\ & : \#=>\text { "hello" } \end{aligned}$ |
| :---: | :---: |

## rstrip! $\rightarrow$ self or nil

Removes trailing whitespace from str, returning nil if no change was made. See also String\#lstrip! and String\#strip!.

scan(pattern) $\rightarrow$ array
scan(pattern) \{|match, ...| block \} $\rightarrow$ 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 is itself an array containing one entry per group.

```
a = "cruel world"
a.scan(/\w+/)
a.scan(/.../)
```



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 $\rightarrow$ new_str

scrub(repl) $\rightarrow$ new_str
scrub\{|bytes|\} $\rightarrow$ 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! $\rightarrow$ str
scrub!(repl) $\rightarrow$ str
scrub! $\{\mid$ bytes| $\} \rightarrow$ 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\x81".scrub!("*") ##二> "abc\u3042*
```

setbyte(index, integer) $\rightarrow$ integer
modifies the indexth byte as integer.

## length $\rightarrow$ integer <br> size $\rightarrow$ integer

Returns the character length of str.
str[index] $\rightarrow$ new_str or nil
str[start, length] $\rightarrow$ new_str or nil
str[range] $\rightarrow$ new_str or nil
str[regexp] $\rightarrow$ new_str or nil
str[regexp, capture] $\rightarrow$ new_str or nil
str[match_str] $\rightarrow$ new_str or nil
slice(index) $\rightarrow$ new_str or nil
slice(start, length) $\rightarrow$ new_str or nil
slice(range) $\rightarrow$ new_str or nil
slice(regexp) $\rightarrow$ new_str or nil
slice(regexp, capture) $\rightarrow$ new_str or nil
slice(match_str) $\rightarrow$ 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.

| $\mathrm{a}=$ "hello there" |  |
| :---: | :---: |
| a[1] | \#=> "e" |
| $a[2,3]$ | \#=> "llo" |
| a[2. 3] | \#=> "11. |
| $a[-3,2]$ | \#=> "er" |
| a [7..-2] | \#=> "her" |
| $\mathrm{a}[-4 .,-2]$ | \#=> "her" |
| $\mathrm{a}[-2 . .4$ ] | => |
| $\mathrm{a}[11,0]$ | \#=> |
| a[11] | \#=> nill |
| $\mathrm{a}[12,0]$ | \#=> nil |
| $a[12 . .-1]$ | \#=> nil |
| a[/[aeiou] ( ) \1/] | \#=> "elli" |
| a[/[aeiou](.) \1/, 0] | \#=> "ell" |
| a[/[aeiou](.)\1/, 1] | \#\#> "I" |
| a[/[aeiou](.)\1/, 2] | \#=> nil |
| a[/(?<vowel>[aeiou])(?<non_vowel>[^aeiou])/, "nor <br> a[/(?<vowel>[aeiou])(?<non_vowel>[^aeiou])/, "vov |  |

## a["lo"] <br> a["bye"]

slice!(fixnum) $\rightarrow$ new_str or nil
slice!(fixnum, fixnum) $\rightarrow$ new_str or nil
slice!(range) $\rightarrow$ new_str or nil
slice!(regexp) $\rightarrow$ new_str or nil
slice!(other_str) $\rightarrow$ new_str or nil
Deletes the specified portion from str, and returns the portion deleted.

```
string = "this is a string"
string.slice!(2)
string.slice!(3..6)
string.slice!(/s.*t/)
string.slice!("r")
string
```


## 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 zerolength 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 Array is returned as the string is considered to have no fields to split.

| " now's the time".split | => |
| :---: | :---: |
| " now's the time".split(' ') | \#=> ["now's |
| " now's the time".split(/ /) | \#=> ["", "now's' |
| "1, 2.34, 56, 7".split(\%r ${ }^{\text {a }}$, $\left.\mathrm{s}^{*}\right\}$ ) | f=> ["1", "2.34' |
| "hello".split(//) | \#=> ["h", "e", |
| "hello".split(/), 3) | \#\#=> "n' |
| "hi mom".split(\%r\{\s*\}) | \#=> ["h", "i', |
| "mellow yellow".split("ello") |  |
| "1, 2, , 3, 4, , ". split (', ') | \#=> |
| "1, 2, 3, 4, '".split (',', 4) | \#=> |
| "1,2, 3, 4, '".split(',', -4) | \#=> |
| "".split(', ', -1) | \%=> [] |
| $4$ |  |

## squeeze([other_str]*) $\rightarrow$ 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]*) $\rightarrow$ str or nil

Squeezes str in place, returning either str, or nil if no changes were made.
start_with? ([prefixes]+) $\rightarrow$ true or false
Returns true if str starts with one of the prefixes given.


## strip $\rightarrow$ new_str

Returns a copy of str with leading and trailing whitespace removed.

```
" hello ".strip
"\tgoodbye\r\n".strip
```


## strip! $\rightarrow$ str or nil

Removes leading and trailing whitespace from str.
Returns nil if str was not altered.
sub(pattern, replacement) $\rightarrow$ new_str
sub(pattern, hash) $\rightarrow$ new_str
sub(pattern) \{|match| block \} $\rightarrow$ new_str
Returns a copy of str with the first occurrence of pattern replaced by 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. '<br>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 " $\backslash \mathrm{d}$ ", where $d$ is a group number, or " $\backslash k<n>"$, where $n$ is a group name. If it is a double-quoted string, both backreferences 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 String 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 "<br>'".

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.


## sub!(pattern, replacement) $\rightarrow$ str or nil

## 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 $\rightarrow$ new_str
next $\rightarrow$ 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.

"abcd". succ<br>"THX1138". succ<br>"<<koala>>".succ<br>"1999zzz".succ<br>"ZzZ9999".succ<br>"***".succ

succ! $\rightarrow$ str
next! $\rightarrow$ str
Equivalent to String\#succ, but modifies the receiver in place.

## sum( $\mathrm{n}=16$ ) $\rightarrow$ integer

Returns a basic $n$-bit checksum of the characters in
$s t r$, 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 $2 * * \mathrm{n}-1$. This is not a particularly good checksum.

## swapcase $\rightarrow$ 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.

```
"Hello". swapcase
"cYbEr_PuNk11". swapcase

\section*{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 ASCll region.

\section*{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

```

See Kernel.Complex.

\section*{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.
```

"123.45e1".to_f
"45.67 degrees".to_f
"thx1138".to_f

```

\section*{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 \(s t r, 0\) 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)

```

\section*{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'.\#to_r isn't the same as 0.3.to_r. The former is equivalent to '3/10'.\#to_r, but the latter isn't so.
\begin{tabular}{|c|c|c|}
\hline 2 '.to_r & \#=> & (2/1) \\
\hline '300/2'.to_r & \#=> & (150/1) \\
\hline '-9.2'.to_r & \#=> & (-46/5) \\
\hline '-9.2e2'.to_r & \#=> & (-920/1) \\
\hline '1_234_567'.to_r & \#-> & (1234567/1) \\
\hline '21 june 09'.to_r & \#\# & (21/1) \\
\hline '21/06/09'.to_r & \#=> & (7/2) \\
\hline 'bwv 1079'.to_r & \#\# & (0/1) \\
\hline
\end{tabular}

See Kernel.Rational.
```

t0_s -> str

```
to_str \(\rightarrow\) str
Returns the receiver.
to_s \(\rightarrow\) str
to_str \(\rightarrow\) str
Returns the receiver.
intern \(\rightarrow\) symbol
to_sym \(\rightarrow\) symbol

Returns the symbol corresponding to str, creating the symbol if it did not previously exist. See
Symbol\#id2name.
\begin{tabular}{|c|c|c|}
\hline "Koala".intern & \#=> & Koala \\
\hline \(s=1 \mathrm{cat}\) ',to_sym & \#二> & :cat \\
\hline \(\mathrm{s}==\) :cat & \#二> & true \\
\hline s = '@cat'.to_sym & \#=> & :@cat \\
\hline s == :@cat & \#=> & true \\
\hline
\end{tabular}

This can also be used to create symbols that cannot be represented using the : xxx notation.
```

'cat and dog'.to_sym \#\#> :"cat and dog"

```

\section*{tr(from_str, to_str) \(\rightarrow\) 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.
\begin{tabular}{|c|c|}
\hline "hello".tr('el', 'ip') & \#=> "hippo" \\
\hline "hello".tr \({ }^{\text {cheiou', '*') }}\) & , \(h * 11 * *\) \\
\hline "hello".tr('aeiou', 'AA*') & "hall \\
\hline
\end{tabular}

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.
```

"hello".tr('a-y', 'b-z')
"hello".tr('^aeiou', '*')

```

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:


\section*{tr!(from_str, to_str) \(\rightarrow\) str or nil}

Translates str in place, using the same rules as String\#tr. Returns str, or nil if no changes were made.
tr_s(from_str, to_str) \(\rightarrow\) new_str Processes a copy of str as described under String\#tr, then removes duplicate characters in regions that were affected by the translation.


\section*{tr_s!(from_str, to_str) \(\rightarrow\) str or nil}

Performs string\#tr_s processing on str in place, returning str, or nil if no changes were made.

\section*{unpack(format) \(\rightarrow\) 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 sSiIll 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.


This table summarizes the various formats and the Ruby classes returned by each.

\begin{tabular}{|c|c|c|}
\hline \[
\begin{aligned}
& \mathrm{S}<\mathrm{L}<\mathrm{Q}< \\
& \mathrm{S}<\mathrm{l}<\mathrm{q}< \\
& \mathrm{S}!<\mathrm{I}!< \\
& \mathrm{L}!<\mathrm{Q}!< \\
& \mathrm{S}!<\mathrm{i}!< \\
& \mathrm{l}!<\mathrm{q}!<
\end{aligned}
\] & Integer & ```
same as the directives l
little endian
(available since Ruby 1
"S<" is same as "v"
"L<" is same as "V"
``` \\
\hline n & Integer & 16-bit unsigned, networl \\
\hline N & Integer & 32-bit unsigned, networ \\
\hline v & Integer & 16-bit unsigned, VAX (li \\
\hline V & Integer & 32-bit unsigned, VAX (li \\
\hline U & Integer & UTF-8 character \\
\hline W & Integer & BER-compressed integer \\
\hline \multicolumn{3}{|l|}{Float} \\
\hline Directive & Returns & Meaning \\
\hline D, d & Float & double-precision, native \\
\hline F, f & Float & single-precision, native \\
\hline E & Float & double-precision, little \\
\hline e & Float & single-precision, littl¢ \\
\hline G & Float & double-precision, networ \\
\hline g & Float & single-precision, networ \\
\hline \multicolumn{3}{|l|}{String} \\
\hline Directive & Returns & Meaning \\
\hline A & String & arbitrary binary string \\
\hline a & String & arbitrary binary string \\
\hline Z & String & null-terminated string \\
\hline B & String & bit string (MSB first) \\
\hline b & String & bit string (LSB first) \\
\hline H & String & hex string (high nibble \\
\hline h & String & hex string (low nibble 1 \\
\hline u & String & UU-encoded string \\
\hline M & String & quoted-printable, MIME \(¢\) \\
\hline m & String & base64 encoded string ( \\
\hline & & base64 encoded string (F \\
\hline P & String & pointer to a structure \\
\hline \(p\) & String & pointer to a null-termir \\
\hline \multicolumn{3}{|l|}{Misc.} \\
\hline Directive & Returns & Meaning \\
\hline @ & & skip to the offset giver \\
\hline X & & skip backward one byte \\
\hline
\end{tabular}


\section*{upcase \(\rightarrow\) 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.
upcase! \(\rightarrow\) 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) \(\{\mid\) 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.


\section*{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.
```

"9".upto("11").to_a
"25".upto("5").to_a
"07".upto("11").to_a

```

\section*{valid_encoding? \(\rightarrow\) true or false}

Returns true for a string which encoded correctly.
```

"\xc2\xa1". force_encoding("UTF-8") . valid_encoding
"\xc2". force_encoding("UTF-8") . valid_encoding?
"\x80". force_encoding("UTF - 8") . valid_encoding?
4, +

```

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{class Struct}

\section*{In Files}
process.c
- struct.c

\section*{Parent}

\section*{Object}

\section*{Included Modules}
- Enumerable

\section*{Constants}

Tms

\section*{Public Class Methods}
new([class_name] [, member_name]+>) \(\rightarrow\) StructClass
new([class_name] [, member_name]+>)
\{|StructClass| block \} \(\rightarrow\) StructClass
new(value, ...) \(\rightarrow\) obj
StructClass[value, ...] \(\rightarrow\) 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 Struct, 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.


\section*{Public Instance Methods}

\section*{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, Ans
joejr = Customer.new("Joe Smith", "123 Maple, Ans
jane = Customer.new("Jane Doe", "456 Elm, Anytov
joe == joejr
joe == jane

```
struct[member] \(\rightarrow\) anObject
struct[index] \(\rightarrow\) anObject
Attribute Reference-Returns the value of the given
struct member or the member at the given index.
Raises NameError 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, Anytc
joe["name"]
joe[:name] \#=> "Joe Smith'
Smith
joe[0] \#=> "Joe Smith"
struct[name] = obj $\rightarrow$ obj
struct[index] = obj $\rightarrow$ obj
Attribute Assignment-Sets the value of the given struct member or the member at the given index.
Raises NameError if the name 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, Anytr
joe["name"] = "Luke"
joe[:zip] $=$ "90210"
joe.name \#=> "Luke"
joe.zip $\quad \#=>~ " 90210 "$

## each \{|obj| block \} $\rightarrow$ struct

each $\rightarrow$ an_enumerator
Yields the value of each struct member in order. If no block is given an enumerator is returned.

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

Produces:

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

s.each_pair \{|sym, obj| block \} $\rightarrow$ struct
each_pair $\rightarrow$ 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
```


## 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?).

## hash $\rightarrow$ fixnum

Returns a hash value based on this struct's contents (see Object\#hash).

See also Object\#hash.

## to_s $\rightarrow$ string

inspect $\rightarrow$ string
Describe the contents of this struct in a string.
Also aliased as: to_s
© length $\rightarrow$ fixnum
size $\rightarrow$ fixnum
Returns the number of struct members.

```
Customer = Struct.new(:name, :address, :zip)
joe = Customer.new("Joe Smith", "123 Maple, Anytc
joe.length
```


## members $\rightarrow$ array

Returns the struct members as an array of symbols:
Customer = Struct.new(:name, :address, :zip)
joe = Customer.new("Joe Smith", "123 Maple, Anyt
joe.members \#=> [:name, :address, :zip]
l

## select $\{|i|$ block $\} \rightarrow$ array

select $\rightarrow$ an_enumerator
Yields each member value from the struct to the block and returns an Array containing the member values from the struct for which the given block returns a true value (equivalent to Enumerable\#select).

```
Lots = Struct.new(:a, :b, :c, :d, :e, :f)
l = Lots.new(11, 22, 33, 44, 55, 66)
l.select {|v| (v % 2).zero? }
```


## length $\rightarrow$ fixnum <br> size $\rightarrow$ fixnum

Returns the number of struct members.


## to_a $\rightarrow$ array

values $\rightarrow$ array
Returns the values for this struct as an Array.

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



## to_h $\rightarrow$ hash

Returns a Hash containing the names and values for the struct's members.

```
Customer = Struct.new(:name, :address, :zip)
joe = Customer.new("Joe Smith", "123 Maple, Anytc
joe.to_h[:address]
```

```
to_s()
```

Alias for: inspect
to_a $\rightarrow$ array
values $\rightarrow$ array
Returns the values for this struct as an Array.

values_at(selector, ...) $\rightarrow$ an_array
Returns the struct member values for each selector as an Array. A selector may be either an Integer offset or a Range of offsets (as in Array\#values_at).

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.

```
module One
    class Fred
    end
    $f1 = :Fred
end
module Two
    Fred = 1
    $f2 = :Fred
end
def Fred()
end
$f3 = :Fred
$f1.object_id
$f2.object_id
$f3.object_id

\section*{In Files}
string.c

\section*{Parent}

\section*{Object}

\section*{Included Modules}

\section*{Comparable}

\section*{Public Class Methods}

\section*{all_symbols \(\rightarrow\) array}

Returns an array of all the symbols currently in Ruby's symbol table.


\section*{Public Instance Methods}
symbol <=> other_symbol \(\rightarrow-1,0,+1\) 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.


See String\#<=> for more information.

\section*{sym \(==\) obj \(\rightarrow\) true or false}

Equality—If sym and obj are exactly the same symbol, returns true.

\section*{sym \(==\) obj \(\rightarrow\) true or false}

Equality—If sym and obj are exactly the same symbol, returns true.
sym \(=\sim\) obj \(\rightarrow\) fixnum or nil
match(obj) \(\rightarrow\) fixnum or nil
Returns sym.to_s =~ obj.
sym[idx] \(\rightarrow\) char
sym[b, n] \(\rightarrow\) string
slice(idx) \(\rightarrow\) char
slice(b, n) \(\rightarrow\) string
Returns sym.to_s[].

\section*{capitalize \(\rightarrow\) symbol}

Same as sym.to_s.capitalize.intern.
casecmp(other) \(\rightarrow-1,0,+1\) or nil
Case-insensitive version of Symbol\#<=>.
downcase \(\rightarrow\) symbol
Same as sym.to_s.downcase.intern.

Returns that sym is :"" or not.

\section*{encoding \(\rightarrow\) encoding}

Returns the Encoding object that represents the encoding of sym.

\section*{id2name \(\rightarrow\) string}
to_s \(\rightarrow\) string
Returns the name or string corresponding to sym.
:fred.id2name

\section*{inspect \(\rightarrow\) string}

Returns the representation of sym as a symbol literal.
```

:fred.inspect

```
to_sym \(\rightarrow\) sym
intern \(\rightarrow\) sym
In general, to_sym returns the symbol corresponding
to an object. As sym is already a symbol, self is
returned in this case.
length \(\rightarrow\) integer
size \(\rightarrow\) integer
Same as sym.to_s.length.
sym \(=\sim\) obj \(\rightarrow\) fixnum or nil
match(obj) \(\rightarrow\) fixnum or nil
Returns sym.to_s =~ obj.

\section*{SUCC}

Same as sym.to_s.succ.intern.
length \(\rightarrow\) integer
size \(\rightarrow\) integer
Same as sym.to_s.length.
sym[idx] \(\rightarrow\) char
sym \([\mathrm{b}, \mathrm{n}] \rightarrow\) string
slice(idx) \(\rightarrow\) char
slice(b, n) \(\rightarrow\) string
Returns sym.to_s[].
succ
Same as sym.to_s.succ.intern.
swapcase \(\rightarrow\) symbol
Same as sym.to_s.swapcase.intern.
to_proc
Returns a Proc object which respond to the given method by sym.
(1..3).collect(\&:to_s)
id2name \(\rightarrow\) string
to_s \(\rightarrow\) string
Returns the name or string corresponding to sym.
:fred.id2name
to_sym \(\rightarrow\) sym
intern \(\rightarrow\) sym
In general, to_sym returns the symbol corresponding to an object. As sym is already a symbol, self is returned in this case.

\section*{upcase \(\rightarrow\) symbol}

Same as sym.to_s.upcase.intern.

Generated by RDoc 3.12.2. Generated with the Darkfish Rdoc Generator 3.

\section*{class SyntaxError}

Raised when encountering Ruby code with an invalid syntax.
```

eval("1+1=2")

```
raises the exception:

SyntaxError: (eval):1: syntax error, unexpect

\section*{In Files}
error.c

\section*{Parent}

ScriptError
Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{class SystemCallError}

SystemCallError is the base class for all lowlevel platform-dependent errors.

The errors available on the current platform are subclasses of SystemCallError and are defined in the Errno module.

File.open("does/not/exist")
raises the exception:

Errno::ENOENT: No such file or directory - do d

\section*{Parent}

\section*{StandardError}

\section*{Public Class Methods}
system_call_error === other \(\rightarrow\) 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) \(\rightarrow\) 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.

\section*{Public Instance Methods}

\section*{errno \(\rightarrow\) fixnum}

Return this SystemCallError's error number.

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{class SystemExit}

Raised by exit to initiate the termination of the script.

\section*{In Files}
error.c

\section*{Parent}

Exception

\section*{Public Class Methods}
new \(\rightarrow\) system_exit
new(status) \(\rightarrow\) system_exit
new(status, msg) \(\rightarrow\) system_exit
new(msg) \(\rightarrow\) 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.

\section*{Public Instance Methods}

\section*{status \(\rightarrow\) fixnum}

Return the status value associated with this system exit.

\section*{success? \(\rightarrow\) true or false}

Returns true if exiting successful, false if not.

Generated by RDoc 3.12.2. Generated with the Darkfish Rdoc Generator 3.

\section*{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

\section*{In Files}
proc.c

\section*{Parent}

\section*{Exception}

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{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 Thread 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

```

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 d
threads << Thread.new \{ 3.times \{ puts "Threal

After creating a few threads we wait for them all to finish consecutively.
threads.each \{ |thr| thr.join \}

\section*{Thread initialization}

In order to create new threads, Ruby provides \(\because \because\) new, \(\because:\) start, and \(\because:\) fork. A block must be provided with each of these methods, otherwise a ThreadError will be raised.

When subclassing the Thread class, the initialize method of your subclass will be ignored by \(::\) start and \(::\) fork. Otherwise, be sure to call super in your initialize method.

\section*{Thread termination}

For terminating threads, Ruby provides a variety of ways to do this.

The class method \(\because: \mathrm{kill}\), 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

\section*{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 status
thr \(=\) Thread.new \{ sleep \}
thr.status
thr.exit
thr.status

You can also use alive? to tell if the thread is running or sleeping, and stop? if the thread is dead or sleeping.

\section*{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.

\section*{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.current[:foo] = "bar"
    Fiber.new {
        p Thread.current[:foo]
        }.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:
Thread.new\{
Thread.current.thread_variable_set (:foo, 1)
p Thread.current.thread_variable_get( : foo)
Fiber.new\{
Thread.current.thread_variable_set (: foo,
p Thread.current.thread_variable_get (: foo
\}.resume
p Thread.current.thread_variable_get (: foo)
\}.join
4

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 thread_variable_set to create new thread-locals, and thread_variable_get to reference them.

There is also thread_variables to list all threadlocals, and thread_variable? to check if a given
thread-local exists.

\section*{Exception handling}

Any thread can raise an exception using the raise 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 abort_on_exception. 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 abort_on_exception= true or setting \$::DEBUG to true.

With the addition of the class method ::handle_interrupt, you can now handle exceptions asynchronously with threads.

\section*{Scheduling}

Ruby provides a few ways to support scheduling threads in your program.

The first way is by using the class method \(::\) stop, 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 ::pass, 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 priority, 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.

\section*{In Files}
- thread.c
\(-\mathrm{vm} . \mathrm{c}\)
- vm_trace.c

\section*{Parent}

\section*{Object}

\section*{Public Class Methods}

\section*{DEBUG \(\rightarrow\) num}

Returns the thread debug level. Available only if compiled with THREAD_DEBUG=-1.

\section*{DEBUG = num}

Sets the thread debug level. Available only if compiled with THREAD_DEBUG=-1.

\section*{abort_on_exception \(\rightarrow\) 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.

\section*{abort_on_exception= boolean \(\rightarrow\) true or 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=.

\section*{current \(\rightarrow\) thread}

Returns the currently executing thread.

\section*{Thread.current}

\section*{exit \(\rightarrow\) thread}

Terminates the currently running thread and schedules another thread to be run.

If this thread is already marked to be killed, ::exit returns the Thread.

If this is the main thread, or the last thread, exit the process.
start([args]*) \{|args| block \} \(\rightarrow\) thread
fork([args]*) \{|args| block \} \(\rightarrow\) thread
Basically the same as ::new. However, if class
Thread is subclassed, then calling start in that subclass will not invoke the subclass's initialize method.

\section*{handle_interrupt(hash) \(\{\ldots\} \rightarrow\) result of the block}

Changes asynchronous interrupt timing.
interrupt means asynchronous event and corresponding procedure by \#raise, \#kill, 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).

\section*{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.

\section*{Usage}

In this example, we can guard from \#raise exceptions.
Using the :never TimingSymbol the RuntimeError exception will always be ignored in the first block of the main thread. In the second ::handle_interrupt block we can purposefully handle RuntimeError exceptions.


While we are ignoring the RuntimeError exception, it's safe to write our resource allocation code. Then, the ensure block is where we can safely deallocate your resources.

\section*{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


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.

\section*{Stack control settings}

It's possible to stack multiple levels of
::handle_interrupt blocks in order to control more than one ExceptionClass and TimingSymbol at a time.
```

Thread.handle_interrupt(FooError => :never) {
Thread.handle_interrupt(BarError => :never) {
:FooError and BarError are prohibited
}
}

```

\section*{Inheritance with ExceptionClass}

All exceptions inherited from the ExceptionClass parameter will be considered.


\section*{kill(thread) \(\rightarrow\) thread}

Causes the given thread to exit, see also ::exit.

list \(\rightarrow\) array

Returns an array of Thread 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>

```

\section*{main \(\rightarrow\) thread}

Returns the main thread.
```

new $\{\ldots$ \} $\rightarrow$ thread

```
new(*args, \&proc) \(\rightarrow\) thread
new(*args) \{ |args| ... \} \(\rightarrow\) thread

Creates a new thread executing the given block.
Any args given to \(\because:\) new will be passed to the block:


A ThreadError exception is raised if ::new is called without a block.

If you're going to subclass Thread, be sure to call super in your initialize method, otherwise a ThreadError will be raised.
pass \(\rightarrow\) nil

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.

\section*{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.
```

def Thread.kick_interrupt_immediately
Thread.handle_interrupt(Object => :immediate)
Thread.pass
}
end

```

If error is given, then check only for error type deferred events.

\section*{Usage}



This example can also be written as the following, which you should use to avoid asynchronous interrupts.

start([args]*) \{|args| block \} \(\rightarrow\) thread
fork([args]*) \{|args| block \} \(\rightarrow\) thread
Basically the same as ::new. However, if class
Thread is subclassed, then calling start in that
subclass will not invoke the subclass's initialize method.
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

```


\section*{Public Instance Methods}

\section*{thr[sym] \(\rightarrow\) obj or nil}

Attribute Reference-Returns the value of a fiberlocal variable (current thread's root fiber if not explicitly inside a Fiber), using either a symbol or a string name. If the specified variable does not exist, returns nil.
```

[
Thread.new { Thread.current["name"] = "A" },
Thread.new { Thread.current[:name] = "B" },
Thread.new { Thread.current["name"] = "C" }
].each do |th|
th.join
puts "\#{th.inspect}: \#{th[:name]}"
end
This will produce:

```
```

\#<Thread:0x00000002a54220 dead>: A

```
#<Thread:0x00000002a54220 dead>: A
#<Thread:0x00000002a541a8 dead>: B
#<Thread:0x00000002a541a8 dead>: B
#<Thread:0x00000002a54130 dead>: C
```

\#<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

```

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 thread_variable_get and thread_variable_set.
thr[sym] \(=\) obj \(\rightarrow\) 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.
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.

\section*{abort_on_exception= boolean \(\rightarrow\) true or}
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=.
add_trace_func(proc) \(\rightarrow\) proc
Adds proc as a handler for tracing.
See \#set_trace_func and Kernel\#set_trace_func.

\section*{alive? \(\rightarrow\) true or false}

Returns true if thr is running or sleeping.


See also stop? and status.
backtrace \(\rightarrow\) array
Returns the current backtrace of the target thread.
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.

\section*{exit \(\rightarrow\) thr or nil}
kill \(\rightarrow\) thr or nil
terminate \(\rightarrow\) thr or nil
Terminates thr and schedules another thread to be run.

If this thread is already marked to be killed, exit
returns the Thread.
If this is the main thread, or the last thread, exits the process.

\section*{group \(\rightarrow\) thgrp or nil}

Returns the ThreadGroup which contains the given thread, or returns nil if thr is not a member of any group.

\section*{Thread.main.group}
\(\xrightarrow{\square}\) -
inspect \(\rightarrow\) string
Dump the name, id, and status of thr to a string.
join \(\rightarrow\) thr
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.
```

y = Thread.new { 4.times { sleep 0.1; puts 'tick.
puts "Waiting" until y.join(0.15)

```

This will produce:
```

tick
Waiting
tick
Waiting
tick...
tick...

```

\section*{key?(sym) \(\rightarrow\) true or false}

Returns true if the given string (or symbol) exists as a fiber-local variable.
```

me = Thread.current
me[:oliver] = "a"
me.key?(:oliver)
me.key?(:stanley)

```

\section*{keys \(\rightarrow\) array}

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
thr.keys

```

\section*{exit \(\rightarrow\) thr or nil}
kill \(\rightarrow\) thr or nil
terminate \(\rightarrow\) thr or nil
Terminates thr and schedules another thread to be run.

If this thread is already marked to be killed, exit returns the Thread.
If this is the main thread, or the last thread, exits the process.
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.

\section*{priority \(\rightarrow\) integer}

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

```

\section*{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.
```

count1 = count2 = 0
a = Thread.new do
loop { count1 += 1 }
end
a.priority = -1
b = Thread.new do
loop { count2 += 1 }
end
b.priority = -2
sleep 1
count1
622504
count2

```

\section*{raise}
```

raise(string)
raise(exception [, string [, array]])
Raises an exception from the given thread. The caller

```
does not have to be thr. See Kernel\#raise 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

```

\section*{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:
```

a
Got here
C

```

See also the instance method wakeup.

\section*{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

```
```

set_trace_func(proc) -> proc
set_trace_func(nil) }->\mathrm{ nil

```

Establishes proc on thr as the handler for tracing, or disables tracing if the parameter is nil.

See Kernel\#set_trace_func.

\section*{status \(\rightarrow\) string, false or nil}

Returns the status of thr.
\begin{tabular}{|l|}
\hline "sleep" \\
\hline \begin{tabular}{l} 
Returned if this thread is sleeping or waiting \\
on I/O \\
\hline "run" \\
\hline When this thread is executing \\
"aborting" \\
\hline If this thread is aborting \\
\hline false \\
\hline When this thread is terminated normally \\
\hline nil \\
\hline If terminated with an exception. \\
\hline
\end{tabular} \\
\hline
\end{tabular}

\footnotetext{
a = Thread.new \{ raise("die now") \}
b = Thread.new \{ Thread.stop \}
c = Thread.new \{ Thread.exit \}
d \(=\) Thread.new \{ sleep \}
d.kill
\#=> \#<Thread:0x401b3678
a.status
b.status
c.status
d.status
nil

Thread.current.status
"aborting"
}

See also the instance methods alive? and stop?

\section*{stop? \(\rightarrow\) true or false}

Returns true if thr is dead or sleeping.
```

a = Thread.new { Thread.stop }
b = Thread.current
a.stop?
b.stop?

```

See also alive? and status.

\section*{exit \(\rightarrow\) thr or nil}
kill \(\rightarrow\) thr or nil
terminate \(\rightarrow\) thr or nil
Terminates thr and schedules another thread to be run.

If this thread is already marked to be killed, exit returns the Thread.

If this is the main thread, or the last thread, exits the process.

\section*{thread_variable?(key) \(\rightarrow\) true or false}

Returns true if the given string (or symbol) exists as a thread-local variable.
```

me = Thread.current
me.thread_variable_set(:oliver, "a")
me.thread_variable?(:oliver)
me.thread_variable?(:stanley)

```

Note that these are not fiber local variables. Please see Thread\#[] and \#thread_variable_get for more details.

\section*{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:
```

Thread.new {
Thread.current.thread_variable_set("foo", "bar'
Thread.current["foo"] = "bar"
Fiber.new {
Fiber.yield [
Thread.current.thread_variable_get("foo"),
Thread.current["foo"],
]
}.resume
}.join.value

```

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.

\section*{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 \#thread_variable_get and Thread\#[] for more information.

\section*{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

```
```

Thread.current.thread_variable_set("dog", 'woot end
thr.join
thr.thread_variables \#=> [:dog, :cat]

```


Note that these are not fiber local variables. Please see Thread\#[] and \#thread_variable_get for more details.

\section*{value \(\rightarrow\) obj}

Waits for thr to complete, using join, and returns its value or raises the exception which terminated the thread.
```

a = Thread.new { 2 + 2 }
a.value
b = Thread.new { raise 'something went wrong' }
b.value \#=> RuntimeError: Something went wrong

```

\section*{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 run for more information.
```

c = Thread.new { Thread.stop; puts "hey!" }
sleep 0.1 while c.status!='sleep'
c.wakeup
c.join

```

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{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

\section*{In Files}
thread.c

\section*{Parent}

\section*{StandardError}

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{class ThreadGroup}

ThreadGroup provides a means of keeping track of a number of threads as a group.

A given Thread object can only belong to one ThreadGroup 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.

\section*{In Files}
thread.c

\section*{Parent}

\section*{Object}

\section*{Constants}

\section*{Default}

The default ThreadGroup created when Ruby starts; all Threads belong to it by default.

\section*{Public Instance Methods}

\section*{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.
```

puts "Initial group is \#{ThreadGroup::Default.lis
tg = ThreadGroup.new
t1 = Thread.new { sleep }
t2 = Thread.new { sleep }
puts "t1 is \#{t1}"
puts "t2 is \#{t2}"
tg.add(t1)
puts "Initial group now \#{ThreadGroup::Default.li
puts "tg group now \#{tg.list}"

```

This will produce:
```

Initial group is \#[Thread:0x401bdf4c](Thread:0x401bdf4c)
t1 is \#[Thread:0x401b3c90](Thread:0x401b3c90)
t2 is \#[Thread:0x401b3c18](Thread:0x401b3c18)
Initial group now \#[Thread:0x401b3c18](Thread:0x401b3c18)\#[Thread:0](Thread:0)
tg group now \#[Thread:0x401b3c90](Thread:0x401b3c90)

```
enclose \(\rightarrow\) thgrp

Prevents threads from being added to or removed from the receiving ThreadGroup.
New threads can still be started in an enclosed ThreadGroup.
```

ThreadGroup::Default.enclose
thr = Thread::new { Thread.stop }
tg = ThreadGroup::new
tg.add thr
\#=> ThreadError: can't move from the enclosed th

```

Returns true if the thgrp is enclosed. See also \#enclose.
list \(\rightarrow\) array
Returns an array of all existing Thread objects that belong to this group.


Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{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 Time class treats GMT (Greenwich Mean Time) and UTC (Coordinated Universal Time) 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, Time implementation uses a signed 63 bit integer, Bignum or Rational. The integer is a number of nanoseconds since the Epoch which can represent 1823-11-12 to 2116-02-20. When Bignum or Rational is used (before 1823, after 2116, under nanosecond), Time works slower as when integer is used.

\section*{Examples}

\section*{All of these examples were done using the EST timezone which is GMT-5.}

\section*{Creating a new Time instance}

You can create a new instance of Time with \(\because:\) new. This will use the current system time. :.now is an alias for this. You can also pass parts of the time to ::new 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:
\[
\begin{array}{ll}
\text { Time.new(2002) } & \#=>2002-01-0100: \\
\text { Time.new(2002, 10) } & \# \# 2002-10-0100: \\
\text { Time.new(2002, 10, 31) } \#=>2002-10-3100: \\
\text { Time.new } 2002, ~ 10, ~ 31, ~ 2, ~ 2, ~ 2, ~ "+02: 00 ") ~
\end{array}
\]

You can also use gm, local and utc to infer GMT, local and UTC timezones instead of using the current system setting.

You can also create a new time using ::at which takes the number of seconds (or fraction of seconds) since the Unix Epoch.

\section*{Working with an instance of Time}

Once you have an instance of Time 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?

What year was that again?
t.year

Was is daylight savings at the time?
```

t.dst?

```

What's the day a year later?


How many seconds was that since the Unix Epoch?

You can also do standard functions like compare two times.
```

t1 = Time.new(2010)
t2 = Time.new(2011)

| t1 == | t2 | \#=> |
| :---: | :---: | :---: |
| t1 == | t1 | \#=> |
| t1 < | t2 | \#=> |
| $\mathrm{t} 1>$ | t2 | \#=> |

Time.new(2010, 10, 31).between?(t1, t2) \#\#> tru
$\square$

```

\section*{In Files}
time.c

\section*{Parent}

\section*{Object}

\section*{Included Modules}
- Comparable

\section*{Public Class Methods}
at(time) \(\rightarrow\) time
at(seconds_with_frac) \(\rightarrow\) time
at(seconds, microseconds_with_frac) \(\rightarrow\)
time
Creates a new Time 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 Integer, Float, 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.

utc(year) \(\rightarrow\) time
utc(year, month) \(\rightarrow\) time
utc(year, month, day) \(\rightarrow\) time
utc(year, month, day, hour) \(\rightarrow\) time
utc(year, month, day, hour, min) \(\rightarrow\) time
utc(year, month, day, hour, min,
sec_with_frac) \(\rightarrow\) time
utc(year, month, day, hour, min, sec,
usec_with_frac) \(\rightarrow\) time
utc(sec, min, hour, day, month, year, dummy,
dummy, dummy, dummy) \(\rightarrow\) time
gm(year) \(\rightarrow\) time
gm(year, month) \(\rightarrow\) time
gm(year, month, day) \(\rightarrow\) time
gm(year, month, day, hour) \(\rightarrow\) time
gm(year, month, day, hour, min) \(\rightarrow\) time gm(year, month, day, hour, min, sec_with_frac) \(\rightarrow\) time gm(year, month, day, hour, min, sec, usec_with_frac) \(\rightarrow\) time gm(sec, min, hour, day, month, year, dummy, dummy, 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.

local(year) \(\rightarrow\) time
local(year, month) \(\rightarrow\) time
local(year, month, day) \(\rightarrow\) time
local(year, month, day, hour) \(\rightarrow\) time
local(year, month, day, hour, min) \(\rightarrow\) time
local(year, month, day, hour, min,
sec_with_frac) \(\rightarrow\) time
local(year, month, day, hour, min, sec,
usec_with_frac) \(\rightarrow\) time
local(sec, min, hour, day, month, year, dummy, dummy, isdst, dummy) \(\rightarrow\) time mktime(year) \(\rightarrow\) time mktime(year, month) \(\rightarrow\) time mktime(year, month, day) \(\rightarrow\) time mktime(year, month, day, hour) \(\rightarrow\) time mktime(year, month, day, hour, min) \(\rightarrow\) time mktime(year, month, day, hour, min, sec_with_frac) \(\rightarrow\) time
mktime(year, month, day, hour, min, sec, usec_with_frac) \(\rightarrow\) time
mktime(sec, min, hour, day, month, year, dummy, dummy, isdst, dummy) \(\rightarrow\) time Same as \(\because: g m\), but interprets the values in the local time zone.

\section*{Time.local(2000, "jan", 1, 20, 15, 1)}
local(year) \(\rightarrow\) time local(year, month) \(\rightarrow\) time local(year, month, day) \(\rightarrow\) time local(year, month, day, hour) \(\rightarrow\) time local(year, month, day, hour, min) \(\rightarrow\) time local(year, month, day, hour, min, sec_with_frac) \(\rightarrow\) time
local(year, month, day, hour, min, sec, usec_with_frac) \(\rightarrow\) time local(sec, min, hour, day, month, year, dummy, dummy, isdst, dummy) \(\rightarrow\) time
mktime(year) \(\rightarrow\) time
mktime(year, month) \(\rightarrow\) time
mktime(year, month, day) \(\rightarrow\) time
mktime(year, month, day, hour) \(\rightarrow\) time mktime(year, month, day, hour, min) \(\rightarrow\) time mktime(year, month, day, hour, min, sec_with_frac) \(\rightarrow\) time
mktime(year, month, day, hour, min, sec, usec_with_frac) \(\rightarrow\) time
mktime(sec, min, hour, day, month, year, dummy, dummy, isdst, dummy) \(\rightarrow\) time
Same as :.gm, but interprets the values in the local time zone.

Time.local(2000, "jan", 1, 20, 15, 1)
new \(\rightarrow\) time
new(year, month=nil, day=nil, hour=nil, min=nil, sec=nil, utc_offset=nil) \(\rightarrow\) time Returns a Time 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 = Time.new
b = Time.new

```

now \(\rightarrow\) time
Creates a new Time object for the current time. This is same as ::new without arguments.

utc(year) \(\rightarrow\) time
utc(year, month) \(\rightarrow\) time
utc(year, month, day) \(\rightarrow\) time
utc(year, month, day, hour) \(\rightarrow\) time
utc(year, month, day, hour, min) \(\rightarrow\) time
utc(year, month, day, hour, min,
sec_with_frac) \(\rightarrow\) time
utc(year, month, day, hour, min, sec,
usec_with_frac) \(\rightarrow\) time
utc(sec, min, hour, day, month, year, dummy,
dummy, dummy, dummy) \(\rightarrow\) time
gm(year) \(\rightarrow\) time
gm(year, month) \(\rightarrow\) time
gm(year, month, day) \(\rightarrow\) time
gm(year, month, day, hour) \(\rightarrow\) time
gm(year, month, day, hour, min) \(\rightarrow\) time
gm(year, month, day, hour, min,
sec_with_frac) \(\rightarrow\) time
gm(year, month, day, hour, min, sec, usec_with_frac) \(\rightarrow\) time
gm(sec, min, hour, day, month, year, dummy, dummy, 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.
\[
\begin{aligned}
& \text { Time.utc(2000,"jan", 1, 20, 15, 1) } \\
& \text { Time.gm(2000,"jan", 1, 20, 15, 1) }
\end{aligned}
\]

\section*{Public Instance Methods}
time + numeric \(\rightarrow\) time
Addition - Adds some number of seconds (possibly
fractional) to time and returns that value as a new Time object.

time - other_time \(\rightarrow\) float
time - numeric \(\rightarrow\) time
Difference - Returns a new Time object that represents the difference between time and other_time, or subtracts the given number of seconds in numeric from time.
\begin{tabular}{lllll}
\(\mathrm{t}=\) Time.now & \(\#\) & \(\#\) 2007-11-19 08:23:10 & -0600 \\
\(\mathrm{t} 2=\mathrm{t}+2592000\) & \(\#\) & \(\# 2007-12-19\) 08:23:10 & -0600 \\
\(\mathrm{t} 2-\mathrm{t}\) & \(\#\) & 2592000.0 & & \\
\(\mathrm{t} 2-2592000\) & \(\#\) & \\
\hline
\end{tabular}
time <=> other_time \(\rightarrow-1,0,+1\) 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.

asctime \(\rightarrow\) string
ctime \(\rightarrow\) string
Returns a canonical string representation of time.

asctime \(\rightarrow\) string
ctime \(\rightarrow\) string
Returns a canonical string representation of time.

day \(\rightarrow\) fixnum
mday \(\rightarrow\) fixnum
Returns the day of the month (1..n) for time.
```

t = Time.now
2007-11-19 08:27:03 -0600
t.day
t.mday

```
isdst \(\rightarrow\) true or false
dst? \(\rightarrow\) true or false

Returns true if time occurs during Daylight Saving
Time in its time zone.
CST6CDT
Time.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?

\section*{Asia/Tokyo}

Time.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?
"JST"
false false "JST" false false

\section*{eql?(other_time)}

Returns true if time and other_time are both Time
objects with the same seconds and fractional seconds.

\section*{friday? \(\rightarrow\) true or false}

Returns true if time represents Friday.


\section*{getgm \(\rightarrow\) new_time}
getutc \(\rightarrow\) new_time
Returns a new Time object representing time in UTC.

getlocal \(\rightarrow\) new_time
getlocal(utc_offset) \(\rightarrow\) new_time
Returns a new Time 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.
\begin{tabular}{|c|c|c|}
\hline \(\mathrm{t}=\) Time. \(\mathrm{utc}(2000,1,1,20,15,1)\) & \#=> & 2000-01-01 2 \\
\hline t.utc? & \#=> & true \\
\hline 1 = t.getlocal & \#\#> & 2000-01-01 \\
\hline l.utc? & \#=> & false \\
\hline t == l & \#=> & true \\
\hline j = t.getlocal("+09:00") & => & 2000-01-02 0 \\
\hline j.utc? & \#=> & false \\
\hline t == j & \# & true \\
\hline 1 & & , \\
\hline
\end{tabular}

\section*{getgm \(\rightarrow\) new_time}
getutc \(\rightarrow\) new_time
Returns a new Time object representing time in UTC.
\begin{tabular}{|c|c|}
\hline t = Time.local(2000,1,1,20,15,1) & \#\# \(=\) 2000-01-0: \\
\hline t.gmt? & \#=> false \\
\hline \(y=t . g e t g m\) & \#=> 2000-01.02 \\
\hline y.gmt? & \#=> true \\
\hline \(\mathrm{t}==\mathrm{y}\) & \#=> true \\
\hline 1 & - \\
\hline
\end{tabular}
utc? \(\rightarrow\) true or false
gmt? \(\rightarrow\) true or false
Returns true if time represents a time in UTC (GMT).
\begin{tabular}{|c|c|}
\hline \(\mathrm{t}=\) Time.now & \#=> 2007-11- \\
\hline t.utc? & \#=> false \\
\hline t = Time.gm(2000, "jan", 1, 20, 15, 1) & \#=> 2000-01. \\
\hline t.utc? & \#=> true \\
\hline \(\mathrm{t}=\) Time.now & \#=> 2007-11- \\
\hline t.gmt? & \#=> false \\
\hline \(\mathrm{t}=\) Time.gm(2000, \(1,1,20,15,1)\) & \#=> 2000-01. \\
\hline t.gmt? & \#\#> true \\
\hline
\end{tabular}

\section*{gmt_offset \(\rightarrow\) fixnum}
gmtoff \(\rightarrow\) fixnum

\section*{utc_offset \(\rightarrow\) fixnum}

Returns the offset in seconds between the timezone of time and UTC.


\section*{gmtime \(\rightarrow\) time}
utc \(\rightarrow\) time
Converts time to UTC (GMT), modifying the receiver.
```

t = Time.now
t.gmt?
t.gmtime
t.gmt?
t = Time.now
t.utc?
t.utc
t.utc?
\#=> 2007-11-19 08:18:31 -0600
\#=> false
\#=> 2007-11-19 14:18:31 UTC
\#=> true
\#=> 2007-11-19 08:18:51
-0600
\#=> false
\#\#> 2007-1.1-1.9 14:18:51. UTC
\#=> true

```

\section*{gmt_offset \(\rightarrow\) fixnum}
gmtoff \(\rightarrow\) fixnum

\section*{utc_offset \(\rightarrow\) fixnum}

Returns the offset in seconds between the timezone of time and UTC.
```

t = Time.gm(2000,1,1,20,15,1)
t.gmt_offset
l = t.getlocal
\#=> 0
\#=> 2000-01-01.
l.gmt_offset

hash $\rightarrow$ fixnum
Returns a hash code for this Time object.
See also Object\#hash.
hour $\rightarrow$ fixnum
Returns the hour of the day (0..23) for time.

```
t = Time.now
2007-11-19 08:26:20 -0600
```

t.hour
inspect $\rightarrow$ string
to_s $\rightarrow$ string

Returns a string representing time. Equivalent to calling strftime with the appropriate format string.

isdst $\rightarrow$ true or false
dst? $\rightarrow$ true or false
Returns true if time occurs during Daylight Saving Time in its time zone.

```
CSTGCDT:
Time.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
"CST"
false
false
"CDT"
true
```

```
Time.local(2000, 7, 1).dst?
Asia/Tokyo:
Time.local(2000, 1, 1).zone #=> "JST"
Time.local(2000, 1, 1).isdst #=> fallse
Time.local(2000, 1, 1).dst? #=> false
Time.local(2000, 7, 1).zone #=> "JST"
Time.local(2000, 7, 1).isdst #=> ffalse
Time.local(2000, 7, 1).dst? #=> fallse
```


## localtime $\rightarrow$ time

localtime(utc_offset) $\rightarrow$ 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.


## day $\rightarrow$ fixnum

## mday $\rightarrow$ fixnum

Returns the day of the month (1..n) for time.

| $\mathrm{t}=$ Time.now | \#=> 2007-11-19 08:27:03-0600 |
| :---: | :---: |
| t. day | \#=> 19 |
| t.mday | \#=> 19 |

min $\rightarrow$ fixnum
Returns the minute of the hour (0..59) for time.

```
t = Time.now
2007-11-19 08:25:51 -0600
```

t.min

## mon $\rightarrow$ fixnum

## month $\rightarrow$ fixnum

Returns the month of the year (1..12) for time.

```
t = Time.now
2007-11-19 08:27:30 -0600
t.mon
11
t.month
##> 11
```


## monday? $\rightarrow$ true or false

Returns true if time represents Monday.


## mon $\rightarrow$ fixnum

## month $\rightarrow$ fixnum

Returns the month of the year (1..12) for time.

```
t = Time.now
2007-11-19 08:27:30 -0600
t.mon
11
t.month
##> 11
```

nsec $\rightarrow$ int
tv_nsec $\rightarrow$ int
Returns the number of nanoseconds for time.


The lowest digits of to_f and 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.

## round([ndigits]) $\rightarrow$ new_time

Rounds sub seconds to a given precision in decimal digits ( 0 digits by default). It returns a new Time object. ndigits should be zero or positive integer.


Returns true if time represents Saturday.


## sec $\rightarrow$ 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.

```
t = Time.now
```

t.sec

## 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
^ 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 r\epsilon
        -0001, 0000, 1995, 2009, 14292, etc.
    %C - year / 100 (rounded down such as 20 in 20c
    %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-padd
    %I - Hour of the day, 12-hour clock, zero-padd
    %l - Hour of the day, 12-hour clock, blank-padd
    %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
            %3N millisecond (3 digits)
            %6N microsecond (6 digits)
            %9N nanosecond (9 digits)
            %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
        %:z - hour and minute offset from UTC
        %::z - hour, minute and second offset
    %Z - Abbreviated time zone name or similar infc
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 ir
the previous year.
\%G - The week-based year
\(\% \mathrm{~g}\) - 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 wes \(\% \mathrm{U}\) - Week number of the year. The week starts \%W - Week number of the year. The week starts
Seconds since the Epoch:
\%s - Number of seconds since 1970-01-01 00:00:c
Literal string:
\%n - Newline character (\n)
\%t - Tab character (\t)
\%\% - 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:



Various ISO 8601 formats:

| \%Y\%m\%d | => 20071119 | Ca |
| :---: | :---: | :---: |
| \%F | => 2007-11-19 | Ca |
| \%Y-\%m | => 2007-11 | Ca |
| \%Y | => 2007 | Cal |
| \%C | => 20 | Ca |
| \%Y\%j | => 2007323 | Or |
| \%Y-\%j | => 2007-323 | Or |
| \%GW\%V\%u | => 2007W471 | Wec |
| \%G-W\%V-\%u | => 2007-W47-1 | Wec |


| \%GW\%V |  | 2007W47 | We¢ |
| :---: | :---: | :---: | :---: |
| \%G-W\%V |  | 2007-W47 | Wee |
| \%H\%M\%S |  | 083748 | Lod |
| \%T |  | 08:37:48 | Lod |
| \%H\%M |  | 0837 | LOd |
| \%H:\%M |  | 08:37 | Lod |
| \%H |  | 08 | Lod |
| \%H\%M\%S , \%L |  | 083748,000 | LOC |
| \%T, \%L |  | 08:37:48,000 | Lod |
| \%H\%M\%S . \%L |  | 083748.000 | Lod |
| \%T . \%L |  | 08:37:48.000 | Lod |
| \%H\%M\%S\%z |  | 083748-0600 | Lod |
| \%T\%: z |  | 08:37:48-06:00 | Lod |
| \%Y\%m\%dT\%H\%M\%S\%z |  | 20071119T083748-0600 | Dat |
| \%FT\%T\%: z |  | 2007-11-19T08:37: 48-06:00 | D |
| \%Y\%j T\%H\%M\%S\%z |  | 2007323T083748-0600 | D |
| \%Y-\%jT\%T\% : z |  | 2007-323T08:37:48-06:00 | D |
| \%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\%j T\%H\%MZ |  | 2007323T0837Z | Orc |
| \%Y-\%jT\%RZ |  | 2007-323T08:37Z | Orc |
| \%GW\%V\%uT\%H\%M\%z |  | 2007W471T0837-0600 | Wee |
| \%G-W\%V-\%uT\%R\% : z |  | 2007-W47-1T08:37-06:00 | Wee |
| 4 |  |  | - |

## subsec $\rightarrow$ number

Returns the fraction for time.
The return value can be a rational number.

```
t = Time.now
                                ## 2009-03-26 22:33:12 +0900
"%10.9f" % t.to_f ### "1238074392.94056391.7"
t.subsec ###> (94056401/100000000)
4
```

The lowest digits of to_f and subsec are different because IEEE 754 double is not accurate enough to represent the rational number.

The more accurate value is returned by subsec.

## succ $\rightarrow$ new_time

Returns a new Time object, one second later than time. \#succ is obsolete since 1.9.2 for time is not a discrete value.


Use instead time + 1


## sunday? $\rightarrow$ true or false

Returns true if time represents Sunday.

thursday? $\rightarrow$ true or false
Returns true if time represents Thursday.


## to_a $\rightarrow$ 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.


## to_f $\rightarrow$ float

Returns the value of time as a floating point number of seconds since the Epoch.

```
t = Time.now
"%10.5f" % t.to_f
t.to_i
"1270968744.77658"
t.to_i
1270968744
```

Note that IEEE 754 double is not accurate enough to represent the number of nanoseconds since the Epoch.
to_i $\rightarrow$ int
tv_sec $\rightarrow$ int
Returns the value of time as an integer number of seconds since the Epoch.

```
t = Time.now
"%10.5f" % t.to_f
t.to_i
"1270968656.89607"
    1270968656
```


## to_r $\rightarrow$ 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 $\rightarrow$ string

## to_s $\rightarrow$ string

Returns a string representing time. Equivalent to
calling strftime with the appropriate format string.

tuesday? $\rightarrow$ true or false
Returns true if time represents Tuesday.

| $\mathrm{t}=$ Time.local(1991, 2, 19) <br> p t.tuesday? | $\# \Rightarrow 1991-02-19$ |
| :--- | :--- |
|  |  |

nsec $\rightarrow$ int
tv_nsec $\rightarrow$ int
Returns the number of nanoseconds for time.


The lowest digits of to_f and 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.
to_i $\rightarrow$ int
tv_sec $\rightarrow$ int
Returns the value of time as an integer number of seconds since the Epoch.

```
t = Time.now
"%10.5f" % t.to_f #=> "1270968656.89607'
t.to_i
1270968656
```

usec $\rightarrow$ int
tv_usec $\rightarrow$ int
Returns the number of microseconds for time.

usec $\rightarrow$ int
tv_usec $\rightarrow$ int
Returns the number of microseconds for time.

gmtime $\rightarrow$ time
utc $\rightarrow$ time
Converts time to UTC (GMT), modifying the receiver.

```
t = Time.now
t.gmt?
t.gmtime
#=> false
t.gmt? ##> true
t = Time.now
#=> 2007-11-19 08:18:51 -0600
```

t.utc?
t.utc

2007-11-19 14:18:51 UTC
t.utc?
\# => true

## utc? $\rightarrow$ true or false

gmt? $\rightarrow$ true or false
Returns true if time represents a time in UTC (GMT).

| $\mathrm{t}=$ Time.now | \#=> 2007-11- |
| :---: | :---: |
| t.utc? | \#=> false |
| t = Time.gm(2000, "jan", 1, 20, 15, 1) | \#=> 2000-01. |
| t.utc? | \#=> true |
| $\mathrm{t}=$ Time.now | \#=> 2007-11-: |
| t.gmt? | \#=> false |
| $\mathrm{t}=$ Time.gm(2000, 1, 1, 20, 15, 1) | \#=> 2000-01. |
| t.gmt? | \#\#> |

## gmt_offset $\rightarrow$ fixnum

gmtoff $\rightarrow$ fixnum
utc_offset $\rightarrow$ fixnum
Returns the offset in seconds between the timezone of time and UTC.


## wday $\rightarrow$ fixnum

Returns an integer representing the day of the week, $0 . .6$, with Sunday $==0$.

```
t = Time.now
##> 2007-11-20 02:35:35
-0600
t.wday
t.sunday?
```

t.monday?
t.tuesday?
t.wednesday?
\#=> false
t.thursday?
\#=> false
t.friday?
\#=> false
t.saturday? \#\#> false

```

\section*{wednesday? \(\rightarrow\) true or false}

Returns true if time represents Wednesday.

yday \(\rightarrow\) fixnum
Returns an integer representing the day of the year, 1.. 366 .
```

t = Time.now \#\#> 2007-11-19 08:32:31 -0600
t.yday
\#=> 323

```

8ear \(\rightarrow\) fixnum
Returns the year for time (including the century).
```

t = Time.now \#\#> 2007-11-19 08:27:51 -0600
t.year \#=> 2007

```

\section*{zone \(\rightarrow\) 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
t = Time.local(2000, "jan", 1, 20, 15, 1)
t.zone

```

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{class TracePoint}

\section*{A class that provides the functionality of} Kernel\#set_trace_func in a nice Object-Oriented API.

\section*{Example}

We can use TracePoint to gather information specifically for exceptions:


\section*{Events}

If you don't specify the type of events you want to listen for, TracePoint 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

\section*{In Files}
vm_trace.c

\section*{Parent}

Object

\section*{Public Class Methods}

\section*{new(*events) \{ |obj| block \} \(\rightarrow\) obj}

Returns a new TracePoint object, not enabled by default.

Next, in order to activate the trace, you must use \#enable
```

trace = TracePoint.new(:call) do |tp|
p [tp.lineno, tp.defined_class, tp.method_id,
end
\#=> \#[TracePoint:disabled](TracePoint:disabled)
trace.enable
\#\#> false
puts "Hello, TracePoint!"
[48, IRB::Notifier::AbstractNotifier,
printf
4)

```

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 ThreadError is raised.

If the trace method isn't included in the given events filter, a RuntimeError is raised.
```

TracePoint.trace(:line) do |tp|
p tp.raised_exception
end
RuntimeError: 'raised_exception' not support
4|, !

```

If the trace method is called outside block, a RuntimeError is raised.
```

TracePoint.trace(:line) do |tp|
\$tp = tp
end
\$tp.line \#=> access from outside (RuntimeError)

```

Access from other threads is also forbidden.
```

stat $\rightarrow$ 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.
```

trace(*events) { |obj| block } -> obj

```

A convenience method for ::new, that activates the trace automatically.


\section*{Public Instance Methods}

\section*{binding()}

Return the generated binding object from event

\section*{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

```
```

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
end.enable do
C.foo
end

```

\section*{disable \(\rightarrow\) true or false}
disable \{ block \} \(\rightarrow\) obj
Deactivates the trace
Return true if trace was enabled. Return false if trace was disabled.
```

trace.enabled? \#\#> true
trace.disable
trace.enabled?
\#=> false (previous status
\#=> false
trace.disable
\#=> false

```

If a block is given, the trace will only be disable within the scope of the block.
```

trace.enabled?
\#=> true
trace.disable do
trace.enabled?
\# only disabled for this block
end
trace.enabled?
\#=> true

```

Note: You cannot access event hooks within the block.
```

trace.disable { p tp.lineno }
\#=> RuntimeError: access from outside

```

\section*{enable \(\rightarrow\) true or false}

\section*{enable \{ block \} \(\rightarrow\) obj}

Activates the trace
Return true if trace was enabled. Return false if trace was disabled.
```

trace.enabled? \#=> false
trace.enable \#\#> false (previous state)
trace is enabled
trace.enabled? \#=> true
trace.enable \#=> true (previous state)
trace is still enabled

```

If a block is given, the trace will only be enabled within the scope of the block.
```

trace.enabled?
\#=> false
trace.enable do
trace.enabled?
\# only enabled for this block

```
trace. enabled?

Note: You cannot access event hooks within the block.
trace.enable \(\{p\) tp.lineno \(\}\)
enabled? \(\rightarrow\) true or false
The current status of the trace
event()
Type of event
See Events at TracePoint for more information.
inspect \(\rightarrow\) 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

\section*{path()}

Path of the file being run
raised_exception()

Value from exception raised on the : raise event

\section*{return_value()}

Return value from : return, c_return, and b_return event

\section*{self()}

Return the trace object during event
Same as \#binding:
trace.binding.eval('self')

Generated by RDoc 3.12.2. Generated with the Darkfish Rdoc Generator 3.

\section*{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.

\section*{In Files}
- object.c

\section*{Parent}

\section*{Object}

\section*{Public Instance Methods}
true \& obj \(\rightarrow\) true or false
And—Returns false if obj is nil or false, true otherwise.
true ^ \({ }^{\text {obj }} \rightarrow\) ! obj
Exclusive Or-Returns true if obj is nil or false, false otherwise.
inspect()
Alias for: to_s

\section*{to_s \(\rightarrow\) "true"}

The string representation of true is "true".
Also aliased as: inspect
true | obj \(\rightarrow\) 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 RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{class TypeError}

Raised when encountering an object that is not of the expected type.
```

[1, 2, 3].first("two")

```
raises the exception:

TypeError: no implicit conversion of String i

In Files
error.c

\section*{Parent}

\section*{StandardError}

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{class UnboundMethod}

Ruby supports two forms of objectified methods. Class 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

```

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.
```

class Test
def test
:original
end
end
um = Test.instance_method(:test)
class Test
def test
:modified
end
end
t = Test.new
t.test
\#=> :modified
um.bind(t).call
\#=> :original

```

\section*{In Files}
\(\perp\) proc.c

\section*{Parent}

\section*{Object}

\section*{Public Instance Methods}
eql?(other_meth) \(\rightarrow\) true or false
meth \(==\) other_meth \(\rightarrow\) 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.

\section*{arity \(\rightarrow\) 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.


\section*{bind(obj) \(\rightarrow\) 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
|

```

\section*{clone \(\rightarrow\) new_method}

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"

```

\section*{eql?(other_meth) \(\rightarrow\) true or false} meth \(==\) other_meth \(\rightarrow\) 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.
hash \(\rightarrow\) integer
Returns a hash value corresponding to the method object.
See also Object\#hash.
to_s \(\rightarrow\) string
inspect \(\rightarrow\) string
Returns the name of the underlying method.

name \(\rightarrow\) symbol
Returns the name of the method.

\section*{original_name \(\rightarrow\) symbol}

Returns the original name of the method.

\section*{owner \(\rightarrow\) class_or_module}

Returns the class or module that defines the method.

\section*{parameters \(\rightarrow\) array}

Returns the parameter information of this method.

\section*{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)

\section*{super_method()}

Returns a Method of superclass, which would be called when super is used.
to_s \(\rightarrow\) string
inspect \(\rightarrow\) string
Returns the name of the underlying method.


Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{class UncaughtThrowError}

Raised when throw is called with a tag which does not have corresponding catch block.
```

throw "foo", "bar"

```
raises the exception:

\section*{UncaughtThrowError: uncaught throw "foo"}

\section*{In Files}
vm_eval.c

\section*{Parent}

\section*{ArgError}

\section*{Public Class Methods}

\section*{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:

\section*{Public Instance Methods}
tag \(\rightarrow\) obj
Return the tag object which was called for.
to_s \(\rightarrow\) string
Returns formatted message with the inspected tag.
value \(\rightarrow\) obj
Return the return value which was called for.

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{class ZeroDivisionError}

Raised when attempting to divide an integer by 0.
\(42 / 0\)
\(\#=>\) ZeroDivisionError: divided by 0

Note that only division by an exact 0 will raise the exception:


\section*{In Files}
numeric.c

\section*{Parent}

\section*{StandardError}

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\section*{class fatal}
fatal is an Exception that is raised when ruby has encountered a fatal error and must exit. You are not able to rescue fatal.

\section*{In Files}
- error.c

\section*{Parent}

Exception
Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.

\title{
Ruby 2.2.4 Core API Reference API Reference
}

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

Generated by RDoc 3.12.2.
Generated with the Darkfish Rdoc Generator 3.```


[^0]:    (10..15).each \{|n| print n, ' ' \}

[^1]:    八p\{^Ll\}/.match("A")

[^2]:    "abc".force_encoding("UTF-8").ascii_only?

