Gura Library Reference

Yutaka Saito

June 17th, 2017

Contents

Chapter 1

About This Reference

This reference explains about functions and classes that are shipped with Gura interpreter. Refer to Gura Language Manual if you want information about syntax and specifications of Gura language itself.

Chapter 2

Explanatory Note

Functions in this reference are described in a generic expression. For example, if there is a reference described like `func(num:number)`, it means that `func` function takes one argument named `num` with value type of `number`. You can call it like `func(3)`.

If an argument is optional, the argument name is followed by a symbol `?`. For example: `func(num?:number)`. You can call it as `func(2)` or can omit the arugument like `func()`.

If the the arument name has * symbol followed, the arument takes zero or more values. For a function that has a generic expression `func(args*:number)`, it can be called like `func()`, `func(3)`, `func(3, 4)`, `func(3, 4, 2)`, and so on.

If the the arument name has `+` symbol followed, the arument takes one or more values. For a function that has a generic expression `func(args+:number)`, it can be called like `func(3)`, `func(3, 4)`, `func(3, 4, 2)`, and so on. In difference with *, it must take at least one value.

An argument may have a default value. The default value is described with `=>` operator like `func(num:number => 4)`. In such a case, if `num` is omitted, the default value `4` shall be used.

Chapter 3

Predefined Variables

Variable	Type	Explanation
`*`	`iterator`	An iterator instance equivalent with "`0..`".
`-`	`nil`	Value of `nil`.
`@rem`	`nil`	Value of `nil`.
`__name__`	`string`	If the current script is a main one that the interpreter launches, this variable is set to `'__main__'`. If it is imported by another as a module, this variable is set to that module name.
`false`	`boolean`	Value of false.
`nil`	`nil`	Value of `nil`.
`root`	`environment`	Top level scope.
`true`	`boolean`	Value of true.

Chapter 4

Built-in Function

4.1 Formatting and Printing of Text

format(format:string, values*):map

Converts `values` into string depending on formatter specifications in `format` and returns the result in string. For a detail information about formatter specications, refer to the document of `printf()` function.

print(values*):map:void

Prints out `values` to standard output.

printf(format:string, values*):map:void

Prints out `values` to standard output according to formatter specifiers in `format`. The format specifier has a format of `%[flags][width][.precision]specifier`.

The `specifier` takes one of the following characters:

- `d`, `i` .. decimal integer number with a sign mark
- `u` .. decimal integer number wihout a sign mark
- `b` .. binary integer number without a sign mark
- `o` .. octal integer number without a sign mark
- `x` .. hexadecimal integer number in lower character without a sign mark
- `X` .. hexadecimal integer number in upper character without a sign mark
- `e` .. floating number in exponential form
- `E` .. floating number in exponential form (in upper character)
- `f` .. floating number in decimal form
- `F` .. floating number in decimal form (in upper character)
- `g` .. better form between `e` and `f`
- `G` .. better form between `E` and `F`
- `s` .. string
- `c` .. character

The `flags` takes one of the following characters.

- `+` .. Appends a character "`+`" before a positive number.
- `-` .. Adjust a string to left.
- `[SPC]` .. Appends a space character before a positive number.
- `#` .. Appends a prefix before a numbers "`0b`" for a binary, "`0`" for an octal and "`0x`" for a hexadecimal number.
- `0` .. Fills lacking columns with "`0`" instead of space characters.

The `width` is a decimal number that specifies a minimum character. If the width of the corresponding field is less than this number, the lacking part will be filled with space characters or "`0`". If the width is equal to or more than this number, there's nothing to be processed. If an asterisk character "*" is specified for `width`, the minimum character width will be retrieved from the argument list.

The `width` it a character width that appears on a console, and it takes into account each character width based on the specification of East Asian Width. This means that a kanji-character occupies two characters in width.

The `precision` is a decimal number and has different effects depending on `specifier`.

For specifiers that formats integer numbers, it specifies a minimum character width and fills `0` for the lacking column. Format specifiers "`%03d`" and "`%.3d`" have the same effect. When it works in combination with `width`, `precision` fills `0` in the lacking space before `width` does padding. An example is shown below:

```
printf('%5.3d', 23) .. prints "  023"
```

For specifiers `e`, `f` and `g`, it specifies a digit number after a decimal point. Examples are shown below:

```
printf('%.3f', 1 / 3) .. prints "0.333"
printf('%.5f', 1 / 3) .. prints "0.33333"
```

It has no effect with other specifiers.

`println(values*):map:void`

Prints out `values` and an end-of-line character to the standard output.

4.2 Repetition

`cross (`expr+) {block}`

Executes the `block` while evaluating all the combinations of results from `expr` that has format "`var in iteratable`". You can specify one or more such `expr`s as arguments and they are counted up from the one on the right side. Iterators and lists are the most popular iteratables, but even any objects that are cable of generating iterators can be specified as such.

It returns the last evaluated value in the block as its own result, but, if one of `:list`, `:xlist`, `:set`, `:xset` or `:iter` is specified, it returns a list or evaluated value or an iterator. The rule is as follows:

- `:list` .. returns a list of result values
- `:xlist` .. returns a list of result values eliminating `nil`
- `:set` .. returns a list of unique values of results
- `:xset` .. returns a list of unique values of results eliminating `nil`
- `:iter` .. returns an iterator that executes the block
- `:xiter` .. returns an iterator that executes the block, skipping `nil`

Block parameter format is `|idx:number, i0:number, i1:number, ..|` where `idx` indicates an index of the whole loop and each of `i0`, `i1` .. indicates an index of each corresponding iterable.

Example:

```
cross (ch in ['A', 'B', 'C'], i in 1..4) {
    printf('%s-%d ', ch, i)
}
// prints "A-1 A-2 A-3 A-4 B-1 B-2 B-3 B-4 C-1 C-2 C-3 C-4 "
```

`for (`expr+) {block}`

Executes the `block` while evaulating iteration command `expr` that has a format "`var in iteratable`". For `var`, an identifier or a list of identifiers is specified. For `iterable`, you can spedify iterators and lists as well as any objects that are cable of generating iterators.

You can specify one or more `expr` in the argument list. In such a case, it continues to repeat until the shortest iterable among them reaches at its end.

It returns the last evaluated value in the block as its own result, but, if one of `:list`, `:xlist`, `:set`, `:xset` or `:iter` is specified, it returns a list or evaluated value or an iterator. The rule is as follows:

- `:list` .. returns a list of result values
- `:xlist` .. returns a list of result values eliminating `nil`
- `:set` .. returns a list of unique values of results
- `:xset` .. returns a list of unique values of results eliminating `nil`
- `:iter` .. returns an iterator that executes the block
- `:xiter` .. returns an iterator that executes the block, skipping `nil`

Block parameter format is `|idx:number|` where `idx` indicates an index of the loop.

Example:

Example:

```
for (ch in ['A', 'B', 'C'], i in 1..4) {
    printf('%s-%d ', ch, i)
}
// prints "A-1 B-2 C-3"
```

`repeat (n?:number) {block}`

Executes the block for `n` times. If `n` is omitted, it repeats the block execution forever.

It returns the last evaluated value in the block as its own result, but, if one of `:list`, `:xlist`, `:set`, `:xset` or `:iter` is specified, it returns a list or evaluated value or an iterator. The rule is as follows:

- `:list` .. returns a list of result values
- `:xlist` .. returns a list of result values eliminating `nil`
- `:set` .. returns a list of unique values of results
- `:xset` .. returns a list of unique values of results eliminating `nil`
- `:iter` .. returns an iterator that executes the block
- `:xiter` .. returns an iterator that executes the block, skipping `nil`

Block parameter format is `|idx:number|` where `idx` indicates an index of the loop.

`while (`cond) {block}`

Executes the block while the evaluation result of `cond` is true.

It returns the last evaluated value in the block as its own result, but, if one of `:list`, `:xlist`, `:set`, `:xset` or `:iter` is specified, it returns a list or evaluated value or an iterator. The rule is as follows:

- `:list` .. returns a list of result values
- `:xlist` .. returns a list of result values eliminating `nil`
- `:set` .. returns a list of unique values of results
- `:xset` .. returns a list of unique values of results eliminating `nil`
- `:iter` .. returns an iterator that executes the block
- `:xiter` .. returns an iterator that executes the block, skipping `nil`

Block parameter format is `|idx:number|` where `idx` indicates an index of the loop.

`break(value?):symbol_func:void`

Exits from an inside of a loop that is formed with repeating functions like `repeat()`, `while()`, `for()` and `cross()`, as well as other functions generating an iterator.

After this function is called, the current loop value would be set to `value` given in the function's argument. If the argument is omitted, that would be set to `nil`.

However, when the loop function is called with one of the attributes, `:list`, `:xlist`, `:set`, `:xset`, `:iter` and `:xiter`, the argument value of `break()` is NOT included as an element in the list or iterator.

`continue(value?):symbol_func:void`

Cancels the current turn of a loop and continues on to the next. This function can be used in a loop that is formed with repeating functions like `repeat()`, `while()`, `for()` and `cross()`, as well as other functions generating an iterator.

After this function is called, the current loop value would be set to `value` given in the function's argument. If the argument is omitted, that would be set to `nil`.

If the loop function is specified with one of the attributes `:list`, `:xlist`, `:set`, `:xset`, `:iter` and `:xiter`, the argument value of `continue()` is included as an element in the list or iterator.

4.3 Value Generator

`consts(value, num?:number) {block?}`

Creates an iterator that generates the same value specified by the argument `value`.

The argument `num` specifies the number of elements to be generated. If omitted, it would generate the value infinitely.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

Below is an example to create an iterator that returns constant values:

```
x = consts('hello', 10)
// x generates 'hello' for 10 times
```

`dim(n+:number) {block?}`

Returns a list that contains `n` values of `nil`. If you pass multiple numbers for `n`, it would create a nested list.

Below is an example to create a one-dimentional list:

```
x = dim(3)
// x is [nil, nil, nil]
```

Below is an example to create a two-dimentional list:

```
x = dim(3, 2)
// x is [[nil, nil], [nil, nil], [nil, nil]]
```

The optional `block` should return values for each element and takes block parameters: `|i0:number, i1:number, ..|` where the arguments `i0` and `i1` take indices of the loops.

Below is an example to create a one-dimentional list containing a string:

```
x = dim(3) {'Hi'}
// x is ['Hi', 'Hi', 'Hi']
```

Below is an example to create a two-dimentional list that consists of strings showing indices.

```
x = dim(3, 2) {|i, j| format('%d-%d', i, j) }
// x is [['0-0', '0-1'], ['1-0', '1-1'], ['2-0', '2-1']]
```

`interval(begin:number, end:number, samples:number):map:[open,open_l,open_r] {block?}`

Creates an iterator that generates a sequence of numbers by specifying the beginning and ending numbers, and the number of samples between them.

In default, it creates a sequence that contains the beginning and ending numbers. Following attributes would generate the following numbers:

- `:open` .. Numbers in range of `(begin, end)` that doesn't contain either `begin` or `end`.
- `:open_l` .. Numbers in range of `(begin, end]` that doesn't contain `begin`.
- `:open_r` .. Numbers in range of `[begin, end)` that doesn't contain `end`.

`range(num:number, num_end?:number, step?:number):map {block?}`

Creates an iterator that generates a sequence of integer numbers.

This function can be called in three formats that generate following numbers:

- `range(num)` .. Numbers between `0` and `(num - 1)`.
- `range(num, num_end)` .. Numbers between `num` and `(num_end - 1)`.
- `range(num, num_end, step)` .. Numbers between `num` and `(num_end - 1)` incremented by `step`.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

Below are examples:

```
x = range(10)
// x generates 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

x = range(3, 10)
// x generates 3, 4, 5, 6, 7, 8, 9

x = range(3, 10, 2)
// x generates 3, 5, 7, 9
```

4.4 Branch and Flow Control

`if (`cond):leader {block}`

Specify an "if" block within a statement of `if-elsif-else`.

If the evaluation result of `cond` is determined as true, the block would be executed, and its evaluation result would become the returned value of the function.

Otherwise, if the function is followed by a trailer `elsif` or `else`, that would be evaluated. If no trailer exists, the function returns `nil` value.

`elsif (`cond):leader:trailer {block}`

Specify an "elsif" block within a statement of `if-elsif-else`.

If the evaluation result of `cond` is determined as true, the block would be executed, and its evaluation result would become the returned value of the function.

Otherwise, if the function is followed by a trailer `elsif` or `else`, that would be evaluated. If no trailer exists, the function returns `nil` value.

`else():trailer {block}`

Specify an "else" block within a statement of `if-elsif-else` or `try-catch-else-finally`.

`end(dummy*):end_marker:symbol_func:trailer:void`

Specify an end of a sequence.

This function is supposed to be used as a block terminator in an embedded script of a template.

`switch() {block}`

Form a switch block that contains `case()` and `default()` function calls. It calls these functions sequentially and exits the execution when one of the conditions is evaluated as true.

`case(`cond) {block}`

Specify an case block within a switch block. After evaluating an expr object cond, the block shall be executed if it has a value of true.

`default() {block}`

Specify a default block within a switch block. If all the preceding condition of case block are not evaluated as true, this block shall be executed.

`return(value?):symbol_func:void`

Skips the remaining procedure of the current function and returns to the context that calls it.

If it takes an argument, the value is treated as a result of the function. Otherwise, the returned value would be `nil`.

4.5 Exception Handling

`try():leader {block}`

Specify a try block of a statement of try-catch-else-finally. It catches signals that occur in the block and executes a corresponding `catch()` or `else()` function that follow after it.

`catch(errors*:error):leader:trailer {block}`

Specify an catch block of a statement of try-catch-else-finally. It can take multiple numbers of arguments of error objects to handle. If there's no error objects specified, it handles all the errors that are not handled in the preceding `catch()` function calls. Block parameter format: `|error:error|` `error` is an error object that contains information of the handled error.

`finally():finalizer:trailer {block}`

`raise(error:error, msg:string => 'error', value?)`

Raises an error signal with a specified error object, a message string and an additional value.

4.6 Data Converter

`chr(code:number):map`

Converts a UTF-32 code into a string.

`hex(num:number, digits?:number):map:[upper]`

Converts a number into a hexadecimal string. Argument `digits` specifies a minimum columns of the converted result and fills `0` in the lacking space.

In default, it uses lower-case characters in its conversion, while it uses upper-case ones when `:upper` attribute is specified.

`int(value):map`

Converts a value into an integer number like below:

- For a number value, it would be converted into an integer number.
- For a compex value, its absolute number would be converted into an integer number.
- For a string value, it would be parsed as an integer number. An error occurs if it has an invalid format.
- For other values, an error occurs.

`ord(str:string):map`

Converts the first character of a string into a number of UTF-32 code. If the string contains more than one characters, it simply neglects trailing ones.

`tonumber(value):map:[nil,raise,strict,zero]`

Converts a string value into a number by a lexical parsing. If the value is not a string, it first tries to convert the value into a string.

If the string starts with a sequence of characters that can be parsed as a number literal, it's not a failure even when it contains other characters following them. Specifying an attribute `:strict` doesn't allow such a case and fails the process.

If it fails the conversion, it would return `nil` value. Attributes described below are prepared to customize the behaviour in the case of a failure.

- `:raise` .. raises an error
- `:zero` .. returns zero value
- `:nil` .. returns `nil` value (default)

`tostring(value):map`

Converts a value into a string.

`tosymbol(str:string):map`

Converts a string into a symbol.

4.7 Class Operations

`class(superclass?:class) {block?}`

Creates a `class` that includes methods and properties described in the content of the `block`. The detail information on how to describe the block content for this function is written in "Gura Language Manual".

Below is an example to create a class named `Person`:

```
Person = class {
    __init__(name:string, age:number) = {
        this.name = name
        this.age = age
    }
    Print() = {
        printf('name:%s age:%d\n', this.name, this.age)
    }
}

person = Person('Smith', 26)
person.Print()
```

If the argument `superclass`, which is expected to be a constructor function of a super class, is specified, the created class would inherit methods and properties from the specified class.

`classref(type+:expr):map {block?}`

Looks up a class by an expression of a type name.

`struct(`args+):nonamed:[loose] {block?}`

Creates a `class` for a structure that contains properties specified by `args`. It can optionally take a block which declares methods and properties just like `class()` function does.

An element in `args` is an expression that has the same format with one in the argument list of a function's declaration. Each variable name becomes a member name in the created instance.

Below is an example to create a structure named `Person`:

```
Person = struct(name:string, age:number)
person = Person('Smith', 26)
printf('name:%s age:%d\n', person.name, person.age)
```

If `:loose` attribute is speicied, the generated constructor would take all the arguments as optional. Omitted variables are set to `nil`

`super(obj):map {block?}`

Returns a reference to `obj` through which you can call methods of the super class.

Example:

```
A = class {
    func() = {}
}

B = class(A) {
    func() = {}
}

b = B()
b.func()         // B#func() is called.
super(b).func()  // A#func() is called.
```

4.8 Scope Operations

`local(`syms+)`

Declares symbols of variable that are supposed to be accessed locally in a block.

`locals(module?:module) {block?}`

Returns an environment object that belongs to a specified module. If the argument `module` is omitted, it returns an `environment` object of the current scope.

`outers() {block?}`

Returns an environment object that accesses to an outer scope.

`public():void {block}`

Declares symbols as public ones that are accessible from outer scopes.

If you want to make `foo` and `bar` accessible, call this function like below:

```
public { foo, bar }
```

`scope(target?) {block}`

Evaluates block with a local scope.

4.9 Module Operations

`import(‘module, ‘alias?):[binary,mixin_type,overwrite] {block?}`

Imports a module and creates a variable that represents the imported module. It also returns a value that is a reference to the module.

It searches module files in directories specified by a variable `sys.path`.

There are three format to call this function like follow:

- `import(foo)` .. imports `foo` module and creates a module object named `foo`
- `import(foo, bar)` .. imports `foo` module and creates a module object named `bar`
- `import(foo) {symbol1, symbol2, symbol3}` .. imports `foo` and mixes up the module’s properties `symbol1`, `symbol2` and `symbol3` in the current scope.

In the third format, you can specify an asterisk character to mixes up all the symbols defined in the module like below:

```
import(foo) {*}
```

If a specified symbol conflicts with what already exists in the current scope, it will cause an error. Specifying the attribute `:overwrite` will avoid such an error and allow overwriting of symbols.

If the argument `module` is prefixed by a minus operator like `-foo`, it will not create a variable that represents the imported module.

If the argument `module` is prefixed by an and operator like `&foo`, the trailing expression will be evaluated and its result, which must be a string, is treated as a module name to be imported. Below is a sample to import `foo` module through a variable that contains that name:

```
var = 'foo'
import(&var)
```

`module() {block}`

Creates a module that contains functions and variables defined in the block and returns it as a module object. This can be used to realize a namespace.

4.10 Value Type Information

`isbinary(value)`

Returns `true` if the `value` is an instance of `binary`, and `false` otherwise.

`isboolean(value)`

Returns `true` if the `value` is an instance of `boolean`, and `false` otherwise.

`isclass(value)`

Returns `true` if the `value` is an instance of `class`, and `false` otherwise.

`iscomplex(value)`

Returns `true` if the `value` is an instance of `complex`, and `false` otherwise.

`isdatetime(value)`

Returns `true` if the `value` is an instance of `datetime`, and `false` otherwise.

`isdict(value)`

Returns `true` if the `value` is an instance of `dict`, and `false` otherwise.

`isenvironment(value)`

Returns `true` if the `value` is an instance of `environment`, and `false` otherwise.

`iserror(value)`

Returns `true` if the `value` is an instance of `error`, and `false` otherwise.

`isexpr(value)`

Returns `true` if the `value` is an instance of `expr`, and `false` otherwise.

`isfunction(value)`

Returns `true` if the `value` is an instance of `function`, and `false` otherwise.

`isiterator(value)`

Returns `true` if the `value` is an instance of `iterator`, and `false` otherwise.

`islist(value)`

Returns `true` if the `value` is an instance of `list`, and `false` otherwise.

`ismodule(value)`

Returns `true` if the `value` is an instance of `module`, and `false` otherwise.

`isnil(value)`

Returns `true` if the `value` is an instance of `nil`, and `false` otherwise.

`isnumber(value)`

Returns `true` if the `value` is an instance of `number`, and `false` otherwise.

`isrational(value)`

Returns `true` if the `value` is an instance of `rational`, and `false` otherwise.

`issemaphore(value)`

Returns `true` if the `value` is an instance of `semaphore`, and `false` otherwise.

`isstring(value)`

Returns `true` if the `value` is an instance of `string`, and `false` otherwise.

`issymbol(value)`

Returns `true` if the `value` is an instance of `symbol`, and `false` otherwise.

istimedelta(value)

Returns `true` if the `value` is an instance of `timedelta`, and `false` otherwise.

isuri(value)

Returns `true` if the `value` is an instance of `uri`, and `false` otherwise.

isdefined(`identifier)

Returns `true` if `identifier` is defined, and `false` otherwise.

isinstance(value, type+:expr):map

Returns `true` if `value` is an instance of `type` or its descendant, and `false` otherwise.

istype(value, type+:expr):map

Returns `true` if `value` is of the type of `type`, and `false` otherwise.

typename(`value)

Returns a type name of the value.

undef(`identifier+):[raise]

Undefines `identifier` in the current scope.

4.11 Data Processing

choose(index:number, values+):map

Picks up a value placed at `index` in the argument list `values`.

Sample:

```
choose(0, 'apple', 'orange', 'banana') // returns 'apple'
choose(2, 'apple', 'orange', 'banana') // returns 'banana'
```

cond(flag:boolean, value1:nomap, value2?:nomap):map

Returns `value1` if `flag` is determined as true, and `value2` otherwise. If the argument `value2` is omitted, it will return `nil` when `flag` is determined as false.

This function behaves in a similar way with `if` function when it's called like below:

```
if (flag) { value1 } else { value2 }
```

Notice that they have the following differences:

- Function `cond()` always evaluates arguments `value1` and `value2` no matter what `flag` value is, while function `if()` doesn't evaluate `value1` expression when `flag` is determined as `false`.
- Function `cond()` works with implicit mapping, which means that the argument `flag` may be a list or an iterator that are to be processed with the implicit mapping.

The arguments `value1` and `value2` are not processed by the implicit mapping, so you can specify a list or an iterator for them as selected items.

`conds(flag:boolean, value1, value2?):map`

Returns `value1` if `flag` is determined as true, and `value2` otherwise. If argument `value2` is omitted, it will return `nil` when `flag` is determined as false.

This function behaves in a similar way with `if` function when it's called like below:

```
if (flag) { value1 } else { value2 }
```

Notice that they have the following differences:

- Function `conds()` always evaluates arguments `value1` and `value2` no matter what `flag` value is, while function `if()` doesn't evaluate `value1` expression when `flag` is determined as false.
- Function `conds()` works with implicit mapping, which means that the arguments `flag`, `value1` and `value2` may be lists or iterators that are to be processed with the implicig mapping.

If you want to specify a list or an iterator for `value1` and `value2` as selected values, use `cond()` function instead.

`max(values+):map`

Returns the maximum value among the given arguments.

`min(values+):map`

Returns the minimum value among the given arguments.

4.12 Random

Random numbers are generated using SIMD-oriented Fast Mersenne Twister (SFMT) library.

`rand(range?:number) {block?}`

Returns a random number between `0` and `(range - 1)`. If argument `range` is not specified, it generates random numbers in a range of [0, 1).

`rand@normal(mu?:number, sigma?:number) {block?}`

Returns a normal distribution random number with a mean value of `mu` and a standard deviation of `sigma`. The default values for `mu` and `sigma` are `0` and `1` respectively.

`rands(range?:number, num?:number) {block?}`

Creates an iterator that returns random numbers between `0` and `(range - 1)`.

If argument `range` is not specified, it generates random numbers in a range of [0, 1).

In default, the created iterator infinitely generates random numbers. The argument `num` specifies how many elements should be generated.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

Below is an example to create a create that generates random numbers:

```
x = rands(100)
// x is an infinite iterator to generates random numbers between 0 and 99
```

`rands@normal(mu?:number, sigma?:number, num?:number) {block?}`

Creates an iterator that returns normal distribution random numbers with a mean value of `mu` and a standard deviation of `sigma`. The default values for `mu` and `sigma` are `0` and `1` respectively.

If argument `range` is not specified, it generates random numbers in a range of [0, 1).

In default, the created iterator infinitely generates random numbers. The argument `num` specifies how many elements should be generated.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

Below is an example to create a create that generates random numbers:

```
x = randns(100)
// x is an infinite iterator to generates random numbers between 0 and 99
```

`randseed(seed:number):void`

Initializes random seed with a specified number.

4.13 Property Listing

`dir(obj?):[noesc]`

Returns a symbol list of variables and functions that are assigned in the environment of `obj`.

In default, when the `obj` is an instance of a class, it also searches symbols assigned in the class that it belongs to and its parent classes. Specifying attribute `:noesc` avoids that behavior.

`dirtype(obj?):[noesc]`

Returns a symbol list of value types that are assigned in the environment of `obj`.

In default, when the `obj` is an instance of a class, it also searches symbols assigned in the class that it belongs to and its parent classes. Specifying attribute `:noesc` inhibits avoids behavior.

Chapter 5

Built-in Class

5.1 argument Class

The `argument` class provides measures to access argument information that is passed to a function. One of its purposes is to check if an attribute is specified in the function call. It also provides a method to control a leader-trailer sequence, a mechanism that flow controls such as `if-elsif-else` and `try-catch` utilize.

There's no constructor to realize an instance of `argument` class. Its instance is implicitly created when a function is called, and you can refer to it by a variable named `__arg__`.

Below is an example to use `argument` class:

```
func(v0, v1, v2):[attr1,attr2] = {
    printf('arg#%d %s\n', 0.., __arg__.values)
    printf('attr1:%s attr2:%s\n', __arg__.isset(`attr1), __arg__.isset(`attr2))
}
```

5.1.1 Property

An `argument` instance has the following properties:

Property	Type	R/W	Explanation
`function`	`function`	R	The `function` instance that has created the argument.
`values`	`list`	R	A list of argument values.

5.1.2 Method

`argument#finalize_trailer():void`

Signals finalizing status to trailers after the current function.

`argument#isset(symbol:symbol)`

Returns `true` if the function is called with an attribute that matches the specified symbol.

`argument#quit_trailer():void`

Cancels evaluation of following trailers.

Example:

```
f(flag:boolean) = {
    !flag && __arg__.quit_trailer()
}

f(true) println('printed')
f(false) println('not printed')
```

5.2 array Class

An instance of the `array` class stores multiple numeric values in a seamless binary sequence. It can directly be passed to functions in C libraries without any modification that expect arrays of `char`, `short`, `int`, `float`, `double` and so on.

There are several `array` classes that deal with different element types as shown below:

Class Name	Element Type
array@int8	Int8
array@uint8	Uint8
array@int16	Int16
array@uint16	Uint16
array@int32	Int32
array@uint32	Uint32
array@int64	Int64
array@uint64	Uint64
array@half	Half
array@float	Float
array@double	Double
array@complex	Complex

In the specification described here, the class name is is represented as `array@T` where `T` means its element type.

Most of methods in `array` class are implemented in `arrayutil` module while the class itself is provided by the intepreter. This is because array features cost much code size and it would be preferable to reduce the size of the intepreter body by separating the implementation of array methods. So, you have to import `arrayutil` module before using the `array` class in your program.

5.2.1 Property

A `array` instance has the following properties:

Property	Type	R/W	Explanation
`T`	`array`	R	Return an array with its row and column being tranposed.
`elembytes`	`number`	R	Returns the size of each element in bytes.
`elemtype`	`symbol`	R	Returns the typename of the elements as a ‘symbol‘ such as “ ‘boolean“, “ ‘int8“, “ ‘uint8“, “ ‘int16“, “ ‘uint16“, “ ‘int32“, “ ‘uint32“, “ ‘int64“, “ ‘uint64“, “ ‘half“, “ ‘float“, “ ‘double“ and “ ‘complex“.
`memoryid`	`string`	R	Returns the id of memory.
`ndim`	`number`	R	Returns the number of dimensions.
`p`	`pointer`	R	Returns the pointer through which you can inspect and modify the content of the array as a binary data.
`shape`	`number`	R	Returns a list of sizes of each dimension.
`size`	`number`	R	Returns the total number of elements.

5.2.2 Constructor

`array(src?, elemtype?:symbol) {block?}`

Creates an `array` instance that contains `double` type of elements from a `list` or an `iterator` specified as an argument `src` or from elements described in the block. The followings are examples to create an `array` instance:

```
array([[0, 1, 2], [3, 4, 5]])
array {{0, 1, 2}, {3, 4, 5}}
```

Specifying the argument `elemtype` would create an array of element type other than `double`. The following examples create an `array` instance of `int32` elements:

```
array([[0, 1, 2], [3, 4, 5]], elemtype => `int32)
array(elemtype => `int32) {{0, 1, 2}, {3, 4, 5}}
```

Available element types are:

```
`int8, `uint8, `int16, `uint16, `int32, `uint32, `int64, `uint64
`half, `float, `double, `complex
```

Functions named `array@T` where `T` gets an element type name are also provided to create an `array` instance of a specific type more easily. Using these functions could simplify the code above like this:

```
array@int32 ([[0, 1, 2], [3, 4, 5]])
array@int32 {{0, 1, 2}, {3, 4, 5}}
```

`array.identity(n:number, elemtype?:symbol):static:map {block?}`

Creates an array of `double` type of elements that represents an identity matrix with specified size `n`.

Example:

```
x = array.identity(3)
// x is array@double {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}}
```

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

Specifying the argument `elemtype` would create an array of element type other than `double`. The followings examples create an `array` instance of `int32` elements:

```
array.identity(3, elemtype => `int32)
```

Available element types are:

```
`int8, `uint8, `int16, `uint16, `int32, `uint32, `int64, `uint64
`half, `float, `double, `complex
```

Methods named `array@T.identity` where `T` gets an element type name are also provided to create an `array` instance of a specific type more easily. Using these functions could simplify the code above like this:

```
array@int32.identity(3)
```

`array.interval(begin:number, end:number, samples:number, elemtype?:symbol):static:map:[open,open_l,open_r]`

Creates a one-dimentional array with `double` type of element that contains a sequence of numbers according to the beginning, ending numbers and the number of samples between them.

In default, it creates a sequence with a range of `[begin, end]` meaning that the sequence contains the beginning and ending numbers. Following attributes would control the range:

- `:open` .. Numbers in range of `(begin, end)` that doesn't contain either `begin` or `end`.
- `:open_l` .. Numbers in range of `(begin, end]` that doesn't contain `begin`.
- `:open_r` .. Numbers in range of `[begin, end)` that doesn't contain `end`.

Example:

```
x = array.interval(0, 3, 7)
// x is array@double {0, 0.5, 1, 1.5, 2, 2.5, 3}
```

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

Specifying the argument `elemtype` would create an array of element type other than `double`. The followings examples create an `array` instance of `float` elements:

```
array.interval(0, 3, 7, elemtype => `float)
```

Available element types are:

```
`int8, `uint8, `int16, `uint16, `int32, `uint32, `int64, `uint64
`half, `float, `double, `complex
```

Methods named `array@T.interval` where `T` gets an element type name are also provided to create an `array` instance of a specific type more easily. Using these functions could simplify the code above like this:

```
array@float.interval(0, 3, 7)
```

`array.ones(dims[]:number, elemtype?:symbol):static:map {block?}`

Creates an array of `double` type of elements, which are initialized with one. The argument `dims` specifies the dimension of the created array.

Example:

```
x = array.ones([3, 4])
// x is array@double {{1, 1, 1, 1}, {1, 1, 1, 1}, {1, 1, 1, 1}}
```

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

Specifying the argument `elemtype` would create an array of element type other than `double`. The followings examples create an `array` instance of `int32` elements:

```
array.ones([3, 4], elemtype => `int32)
```

Available element types are:

```
`int8, `uint8, `int16, `uint16, `int32, `uint32, `int64, `uint64
`half, `float, `double, `complex
```

Methods named `array@T.ones` where `T` gets an element type name are also provided to create an `array` instance of a specific type more easily. Using these functions could simplify the code above like this:

```
array@int32.ones([3, 4])
```

`array.rands(dims[]:number, range?:number, elemtype?:symbol):static:map {block?}`

Creates an array of `double` type of elements which are initialized with random numbers. The argument `dims` specifies the dimension of the created array. When the argument `range` is specified, the generated are all integer numbers that are ranged within `[0, range)`. Otherwise, the generated are real numbers ranged within `[0, 1)`.

Example:

```
x = array.rands([3, 4], 10)
// x is like array@double {{6, 7, 6, 9}, {3, 3, 6, 0}, {7, 9, 5, 5}}
```

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

Specifying the argument `elemtype` would create an array of element type other than `double`. The followings examples create an `array` instance of `int32` elements:

```
array.rands([3, 4], 10, elemtype => `int32)
```

Available element types are:

```
`int8, `uint8, `int16, `uint16, `int32, `uint32, `int64, `uint64
`half, `float, `double, `complex
```

Methods named `array@T.rands` where `T` gets an element type name are also provided to create an `array` instance of a specific type more easily. Using these functions could simplify the code above like this:

```
array@int32.rands([3, 4], 10)
```

`array.rands@normal(dims[]:number, mu?:number, sigma?:number, elemtype?:symbol):static:map {block?}`

Creates an array of `double` type of elements which are initialized with normal distribution random numbers. The argument `dims` specifies the dimension of the created array. The arguments `mu` and `sigma` supply parameters for normal distribution where `mu` specifies the mean value and `sigma` the standard deviation. In default, the `mu` is zero and `sigma` one.

Example:

```
x = array.rands@normal([3, 4], 1, 3)
```

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

Specifying the argument `elemtype` would create an array of element type other than `double`. The followings examples create an `array` instance of `float` elements:

```
array.rands@normal([3, 4], 1, 3, elemtype => `float)
```

Available element types are:

```
`int8, `uint8, `int16, `uint16, `int32, `uint32, `int64, `uint64
`half, `float, `double, `complex
```

Methods named `array@T.rands@normal` where `T` gets an element type name are also provided to create an `array` instance of a specific type more easily. Using these functions could simplify the code above like this:

```
array@int32.rands@normal([3, 4], 1, 3)
```

`array.range(num:number, num_end?:number, step?:number, elemtype?:symbol):static:map {block?}`

Creates an array of `double` type of elements that are initialized with a sequence of integer numbers.

This function can generate three types of sequence like below:

- `array.range(num)` .. between `0` and `(num - 1)`.
- `array.range(num, num_end)` .. between `num` and `(num_end - 1)`.
- `array.range(num, num_end, step)` .. between `num` and `(num_end - 1)` incremented by `step`.

Example:

```
x = array.range(5)
// x is array@double {0, 1, 2, 3, 4}
x = array.range(2, 5)
// x is array@double {2, 3, 4}
x = array.range(2, 10, 2)
// x is array@double {2, 4, 6, 8}
```

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

Specifying the argument `elemtype` would create an array of element type other than `double`. The followings examples create an `array` instance of `int32` elements:

```
array.range(5, elemtype => `int32)
```

Available element types are:

```
`int8, `uint8, `int16, `uint16, `int32, `uint32, `int64, `uint64
`half, `float, `double, `complex
```

Methods named `array@T.range` where `T` gets an element type name are also provided to create an `array` instance of a specific type more easily. Using these functions could simplify the code above like this:

```
array@int32.range(5)
```

`array.rotation(angle:number, xtrans?:number, ytrans?:number, elemtype?:symbol):static:map:[deg] {block?}`

Creates an `array` of `double` type of elements representing a matrix to rotate a 2-D coord.

The rotation `angle` is specified in radian value. If the attribute `:deg` is specified, the `angle` is specified in degree value.

If one or both of `xtrans` and `ytrans` are specified, it would create an array that works as translation as well as rotation.

Example:

```
x = array.rotation(math.pi / 6)
// x is a matrix to rotate by pi/6.
x = array.rotation(30):deg
// x is a matrix to rotate by 30 degree, which is pi/6 in radian.
x = array.rotation(math.pi / 6, 3, 5)
// x is a matrix to rotate by pi/6 and translate by [3, 5].
```

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

Specifying the argument `elemtype` would create an array of element type other than `double`. The followings examples create an `array` instance of `float` elements:

```
array.rotation(math.pi / 6, elemtype => ‘float)
```

Available element types are:

```
‘int8, ‘uint8, ‘int16, ‘uint16, ‘int32, ‘uint32, ‘int64, ‘uint64
‘half, ‘float, ‘double, ‘complex
```

Methods named `array@T.rotation` where `T` gets an element type name are also provided to create an `array` instance of a specific type more easily. Using these functions could simplify the code above like this:

```
array@float.rotation(math.pi / 6)
```

`array.rotation@x(angle:number, xtrans?:number, ytrans?:number, ztrans?:number, elemtype?:symbol):static:ma`

Creates an `array` of `double` type of elements representing a matrix to rotate a 3-D coord around x-axis.

The rotation `angle` is specified in radian value. If the attribute `:deg` is specified, the `angle` is specified in degree value.

If one or more of `xtrans`, `ytrans` and `ztrans` are specified, it would create an array that works as translation as well as rotation.

Example:

```
x = array.rotation@x(math.pi / 6)
// x is a matrix to rotate by pi/6.
x = array.rotation@x(30):deg
// x is a matrix to rotate by 30 degree, which is pi/6 in radian.
x = array.rotation@x(math.pi / 6, 3, -2, 5)
// x is a matrix to rotate by pi/6 and translate by [3, -2, 5].
```

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

Specifying the argument `elemtype` would create an array of element type other than `double`. The followings examples create an `array` instance of `float` elements:

```
array.rotation@x(math.pi / 6, elemtype => `float)
```

Available element types are:

```
`int8, `uint8, `int16, `uint16, `int32, `uint32, `int64, `uint64
`half, `float, `double, `complex
```

Methods named `array@T.rotation@x` where `T` gets an element type name are also provided to create an `array` instance of a specific type more easily. Using these functions could simplify the code above like this:

```
array@float.rotation@x(math.pi / 6)
```

`array.rotation@y(angle:number, xtrans?:number, ytrans?:number, ztrans?:number, elemtype?:symbol):static:ma`

Creates an `array` of `double` type of elements representing a matrix to rotate a 3-D coord around y-axis.

The rotation `angle` is specified in radian value. If the attribute `:deg` is specified, the `angle` is specified in degree value.

If one or more of `xtrans`, `ytrans` and `ztrans` are specified, it would create an array that works as translation as well as rotation.

Example:

```
x = array.rotation@y(math.pi / 6)
// x is a matrix to rotate by pi/6.
x = array.rotation@y(30):deg
// x is a matrix to rotate by 30 degree, which is pi/6 in radian.
x = array.rotation@y(math.pi / 6, 3, -2, 5)
// x is a matrix to rotate by pi/6 and translate by [3, -2, 5].
```

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

Specifying the argument `elemtype` would create an array of element type other than `double`. The followings examples create an `array` instance of `float` elements:

```
array.rotation@y(math.pi / 6, elemtype => `float)
```

Available element types are:

```
`int8, `uint8, `int16, `uint16, `int32, `uint32, `int64, `uint64
`half, `float, `double, `complex
```

Methods named `array@T.rotation@y` where `T` gets an element type name are also provided to create an `array` instance of a specific type more easily. Using these functions could simplify the code above like this:

```
array@float.rotation@y(math.pi / 6)
```

`array.rotation@z(angle:number, xtrans?:number, ytrans?:number, ztrans?:number, elemtype?:symbol):static:ma`

Creates an `array` of `double` type of elements representing a matrix to rotate a 3-D coord around z-axis.

The rotation `angle` is specified in radian value. If the attribute `:deg` is specified, the `angle` is specified in degree value.

If one or more of `xtrans`, `ytrans` and `ztrans` are specified, it would create an array that works as translation as well as rotation.

Example:

```
x = array.rotation@z(math.pi / 6)
// x is a matrix to rotate by pi/6.
x = array.rotation@z(30):deg
// x is a matrix to rotate by 30 degree, which is pi/6 in radian.
x = array.rotation@z(math.pi / 6, 3, -2, 5)
// x is a matrix to rotate by pi/6 and translate by [3, -2, 5].
```

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

Specifying the argument `elemtype` would create an array of element type other than `double`. The followings examples create an `array` instance of `float` elements:

```
array.rotation@z(math.pi / 6, elemtype => `float)
```

Available element types are:

```
`int8, `uint8, `int16, `uint16, `int32, `uint32, `int64, `uint64
`half, `float, `double, `complex
```

Methods named `array@T.rotation@z` where `T` gets an element type name are also provided to create an `array` instance of a specific type more easily. Using these functions could simplify the code above like this:

```
array@float.rotation@z(math.pi / 6)
```

`array.scaling(xscale:number, yscale:number, zscale?:number, elemtype?:symbol):static:map {block?}`

Creates an `array` of `double` type of elements representing a matrix to scale a coord.

It creates a matrix that works on a 2-D coord if `xscale` and `yscale` are specified while it creates a matrix on a 3-D coord if `xscale`, `yscale` and `zscale` are specified.

Example:

```
x = array.scaling(3, 4)
// x is a matrix to scale a 2-D coord by [3, 4].
x = array.scaling(3, 4, -2)
// x is a matrix to scale a 3-D coord by [3, 4, -2].
```

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

Specifying the argument `elemtype` would create an array of element type other than `double`. The followings examples create an `array` instance of `float` elements:

```
array.scaling(3, 4, elemtype => `float)
```

Available element types are:

```
`int8, `uint8, `int16, `uint16, `int32, `uint32, `int64, `uint64
`half, `float, `double, `complex
```

Methods named `array@T.scaling` where `T` gets an element type name are also provided to create an `array` instance of a specific type more easily. Using these functions could simplify the code above like this:

```
array@float.scaling(3, 4
```

`array.translation(xtrans:number, ytrans:number, ztrans?:number, elemtype?:symbol):static:map {block?}`

Creates an `array` of `double` type of elements representing a matrix to translate a coord.

It creates a matrix that works on a 2-D coord if `xtrans` and `ytrans` are specified while it creates a matrix on a 3-D coord if `xtrans`, `ytrans` and `ztrans` are specified.

Example:

```
x = array.translation(3, 4)
// x is a matrix to translate a 2-D coord by [3, 4].
x = array.translation(3, 4, -2)
// x is a matrix to translate a 3-D coord by [3, 4, -2].
```

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

Specifying the argument `elemtype` would create an array of element type other than `double`. The followings examples create an `array` instance of `float` elements:

```
array.translation(3, 4, elemtype => `float)
```

Available element types are:

```
‘int8, ‘uint8, ‘int16, ‘uint16, ‘int32, ‘uint32, ‘int64, ‘uint64
‘half, ‘float, ‘double, ‘complex
```

Methods named `array@T.translation` where `T` gets an element type name are also provided to create an `array` instance of a specific type more easily. Using these functions could simplify the code above like this:

```
array@float.translation(3, 4
```

array.zeros(dims[]:number, elemtype?:symbol):static:map {block?}

Creates an array of `double` type of elements, which are initialized with zero. The argument `dims` specifies the dimension of the created array.

Example:

```
x = array.zeros([3, 4])
// x is array@double {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}
```

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

Specifying the argument `elemtype` would create an array of element type other than `double`. The followings examples create an `array` instance of `int32` elements:

```
array.zeros([3, 4], elemtype => ‘int32)
```

Available element types are:

```
‘int8, ‘uint8, ‘int16, ‘uint16, ‘int32, ‘uint32, ‘int64, ‘uint64
‘half, ‘float, ‘double, ‘complex
```

Methods named `array@T.zeros` where `T` gets an element type name are also provided to create an `array` instance of a specific type more easily. Using these functions could simplify the code above like this:

```
array@int32.zeros([3, 4])
```

5.2.3 Method

array#argmax(axis?:number):map:[last_index] {block?}

Returns index of a maximum number of elements in the target `array`.

array#argmin(axis?:number):map:[last_index] {block?}

Returns index of a minimum number of elements in the target `array`.

`array.dot(a:array, b:array):static:map {block?}`

Calculates a dot product between two arrays that have one or two dimensions.

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

`array#dump(stream?:stream):void:[upper]`

Prints out a binary dump of the array's content.

`array#each():[flat] {block?}`

Creates an iterator that iterates each element in the array.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

The block parameter is `|elem:number, idx:number|` where `elem` is the element value.

`array#elemcast(elemtype:symbol) {block?}`

Cast value type of contained elements.

Available symbols for `elemtype` are as follows:

- `` `boolean ``
- `` `int8 ``
- `` `uint8 ``
- `` `int16 ``
- `` `uint16 ``
- `` `int32 ``
- `` `uint32 ``
- `` `int64 ``
- `` `uint64 ``
- `` `half ``
- `` `float ``

- `‘double`
- `‘complex`

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

`array#fill(value:number):map:void`

Fills array with a specified value.

`array#flatten() {block?}`

Returns an `array` instance as a result that has flattened elements in the target `array`.

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

`array#head(n:number):map {block?}`

Returns an `array` instance as a result that has extracted `n` elements from the beginning of the target `array`.

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

`array#invert(eps?:number):map {block?}`

Returns an `array` instance as a result that has elements of inverted matrix of the target `array`. If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

`array#iselemsame(array:array)`

Returns `true` if the target `array` instance has the same elements with the specified `array`.

`array#issquare()`

Returns `true` if the target `array` consists square matrices.

`array#max(axis?:number):map:[index,last_index] {block?}`

Finds a maximum number of elements in the target `array`.

`array#mean(axis?:number):map {block?}`

Calculates an mean value of elements in the array.

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

`array#min(axis?:number):map:[index,last_index] {block?}`

Finds a minimum number of elements in the target `array`.

`array#offset(n:number):map {block?}`

Returns an `array` instance as a result that has extracted elements of the target `array` after skipping its first `n` elements.

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

`array#paste(offset:number, src:array):map`

Pastes elements of `src` into the target `array` instance.

The argument `offset` specifies the posision where elements are pasted in

`array#reshape(dims[]:number:nil) {block?}`

Returns an `array` instance as a result that has reshaped the target `array` according to a list of dimension size specified by `dims`.

Below are examples:

```
x = array(1..24)
a = x.reshape([6, 4])    // a is an array of 6x4.
b = x.reshape([2, 3, 4]) // b is an array of 2x3x4.
```

A value of `nil` in the list of dimension means it would be calculated from the whole size and other dimension sizes. Only one `nil` is allowed to exist.

```
x = array(1..24)
b = x.reshape([2, nil, 4]) // b is an array of 2x3x4.
```

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

`array#roundoff(threshold?:number) {block?}`

Returns an `array` instance as a result that has rounded off elements less than `threshold` to zero in the target `array`. The default value for `threshold` is `1.0e-6` when omitted.

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

`array#std(axis?:number):map:[p] {block?}`

Calculates a standard deviation value of elements in the target `array`.

In default, it calculates an unbiased estimation of standard deviation in which the summation of squared deviations is divided by `(n - 1)`. Specifying `:p` attributes will calculate a population variance that divides that summation by `n`.

`array#sum(axis?:number):map {block?}`

Calculates a summation value of elements in the target `array`.

`array#tail(n:number):map {block?}`

Returns an `array` instance as a result that has extracted `n` elements from the bottom of the target `array`.

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

`array#transpose(axes[]?:number) {block?}`

Creates an array instance that transposes axes of the original array according to the specified argument `axes`.

If the argument is not specified, two axes at the lowest rank, which correspond to row and column for a matrix, would be transposed.

If `block` is specified, it would be evaluated with a block parameter `|array:array|`, where `array` is the created instance. In this case, the block's result would become the function's returned value.

`array#var(axis?:number):map:[p] {block?}`

Calculates a variation value of elements in the target `array`.

In default, it calculates an unbiased estimation of standard deviation in which the summation of squared deviations is divided by `(n - 1)`. Specifying `:p` attributes will calculate a population variance that divides that summation by `n`.

5.3 audio Class

The `audio` class provides measures to work on audio data.

5.3.1 Method

`audio#each(channel:number, offset?:number):map {block?}`

`audio#get(channel:number, offset:number):map`

`audio#put(channel:number, offset:number, data:number):map:reduce`

`audio#sinewave(channel:number, freq:number, len:number, amplitude?:number):map:reduce`

`audio#store(channel:number, offset:number, data:iterator):reduce`

5.4 binary Class

The `binary` class provides measures to work on binary data that is a byte sequence without any format.

You can create a `binary` instance by calling `binary()` function.

You can also create the instance by specifying `b` prefix before a string literal. For example, the code below creates a `binary` instance that contains a sequence `0x41, 0x42, 0xfe, 0x03, 0x43, 0x44`.

```
b'AB\xfe\x03CD'
```

5.4.1 Property

A `binary` instance has the following properties:

Property	Type	R/W	Explanation
`p`	`pointer`	R	Returns a `pointer` instance that accesses the binary. This result is equivalent to that of calling the method `binary#pointer()`
`size`	`number`	R	Returns the binary size in bytes.
`writable`	`boolean`	R	Indicates if the content of the binary object is writable.

5.4.2 Constructor

`binary(buff*) {block?}`

Creates a `binary` instance after combining `string` or `binary` specified by the arguments `buff`. If no argument is specified for `buff`, an empty `binary` instance would be created.

If `block` is specified, it would be evaluated with a block parameter `|bin:binary|`, where `bin` is the created instance. In this case, the block's result would become the function's returned value.

5.4.3 Method

`binary.alloc(bytes:number, data?:number):static:map {block?}`

Creates a `binary` instance that has the specified size of buffer. If the argument `data`, which should have a number between 0 and 255, is specified, the buffer would be initialized with the value.

If `block` is specified, it would be evaluated with a block parameter `|bin:binary|`, where `bin` is the created instance. In this case, the block's result would become the function's returned value.

`binary#dump(stream?:stream:w):void:[upper]`

Prints a hexadecimal dump from the content of the `binary` to the standard output. If the argument `stream` is specified, the result would be output to the stream.

In default, hexadecimal digit are printed with lower-case characters. Specifying an attribute `:upper` would output them with upper-case characters instead.

Example:

```
>>> b'A quick brown fox jumps over the lazy dog.'.dump():upper
41 20 71 75 69 63 6B 20 62 72 6F 77 6E 20 66 6F  A quick brown fo
78 20 6A 75 6D 70 73 20 6F 76 65 72 20 74 68 65  x jumps over the
20 6C 61 7A 79 20 64 6F 67 2E                     lazy dog.
```

`binary#pointer(offset?:number):map {block?}`

Returns a `pointer` instance that has an initial offset specified by the argument `offset`. If the argument is omitted, it would return a `pointer` instance that points to the top of the binary.

If `block` is specified, it would be evaluated with a block parameter `|p:pointer|`, where `p` is the created instance. In this case, the block's result would become the function's returned value.

`binary#reader() {block?}`

Creates a `stream` instance with which you can read data from the binary by `stream#read()` method. If `block` is specified, it would be evaluated with a block parameter `|s:stream|`, where `s` is the created instance. In this case, the block's result would become the function's returned value.

`binary#writer() {block?}`

Creates a `stream` instance with which you can append data to the binary by `stream#write()` method. If `block` is specified, it would be evaluated with a block parameter `|s:stream|`, where `s` is the created instance. In this case, the block's result would become the function's returned value.

5.5 boolean Class

The `boolean` class represents a boolean data type that is used in logical operations including NOT, AND, OR and XOR.

The `boolean` type provides two values: `true` and `false`. The other types of values can also be calculated in logical operations according to the following general rule:

- The `nil` value is evaluated as `false` value.
- Other values are evaluated as `true`.

Note that the number `0` is treated as `true` in logical operations.

5.6 codec Class

The `codec` class has features to decoding/encoding character codes stored in `string` and `binary`. Following measures are provided:

- Decode .. Converts specific character codes stored in `binary` into UTF-8 code and generages `string` containing the result. It can also delete a CR code (`0x0d`) before a LF code (`0x0d`) at each end of line so that lines in the result are joined with LF code.
- Encode .. Converts UTF-8 character codes stored in `string` into specific codes and generates `binary` containing the result. It can also add a CR code (`0x0d`) before a LF code (`0x0a`) at each end of line so that lines in the result are joined with CR-LF sequence.

You can utilize these functions using `codec` class's methods `codec#decode()` and `codec#encode()` as well as using `stream` class's method to read/write text by specifying `codec` instance when creating its instance.

The actual functions for encoding and decoding are provided by sub modules under `codecs` module. Each module provides following codecs:

- `codecs.basic` .. `us-ascii`, `utf-8`, `utf-16`, `utf-16le`, `utf-16be`
- `codecs.iso8859`.. `iso-8859-1`, `iso-8859-2`, `iso-8859-3`, `iso-8859-4`, `iso-8859-5`, `iso-8859-6`, `iso-8859-7`, `iso-8859-8`, `iso-8859-9`, `iso-8859-10`, `iso-8859-11`, `iso-8859-13`, `iso-8859-14`, `iso-8859-15`, `iso-8859-16`

- `codecs.korean` .. `cp949`, `euc-kr`
- `codecs.chinese` .. `cp936`, `gb2312`, `gbk`, `cp950`, `big5`
- `codecs.japanese` .. `euc-jp`, `cp932`, `shift_jis`, `ms_kanji`, `jis`, `iso-2022-jp`

Importing other codec modules would expand available codecs. You can call `codecs.dir()` to get a list of codec names currently available.

5.6.1 Predefined Variable

Variable	Type	Explanation
`codec.bom@utf8`	`binary`	BOM for UTF-8: `b'\xef\xbb\xbf'`
`codec.bom@utf16le`	`binary`	BOM for UTF-16 little endian: `b'\xff\xfe'`
`codec.bom@utf16be`	`binary`	BOM for UTF-16 big endian: `b'\xfe\xff'`
`codec.bom@utf32le`	`binary`	BOM for UTF-32 little endian: `b'\xff\xfe\x00\x00'`
`codec.bom@utf32be`	`binary`	BOM for UTF-32 big endian: `b'\x00\x00\xfe\xff'`

5.6.2 Constructor

`codec(encoding:string) {block?}`

Creates a `codec` instance of the specified encoding name. You can call `codecs.dir()` to get a list of available encoding names.

If `block` is specified, it would be evaluated with a block parameter `|codec:codec|`, where `codec` is the created instance. In this case, the block's result would become the function's returned value.

5.6.3 Method

`codec#addcr(flag?:boolean):reduce`

The codec's encoder has a feature to add a CR code (`0x0d`) before a LF code (`0x0a`) so that the lines are joined with CR-LF codes in the encoded result. This method enables or disables the feature.

- To enable it, call the method with the argument `flag` set to `true` or without any argument.
- To disable it, call the method with the argument `flag` set to `false`.

`codec#decode(buff:binary):map`

Decodes a binary `buff` and returns the decoded result as `string`.

`codec#delcr(flag?:boolean):reduce`

The codec's decoder has a feature to delete a CR code (`0x0d`) before a LF code (`0x0a`) so that the lines are joined with LF code in the decoded result. This method enables or disables the feature.

- To enable it, call the method with the argument `flag` set to `true` or without any argument.
- To disable it, call the method with the argument `flag` set to `false`.

`codec#encode(str:string):map`

Encodes a string `str` and returns the encoded result as `binary`.

5.6.4 Cast Operation

A function that expects a `codec` instance in its argument can also take a value of `string` that is recognized as a codec name.

With the above casting feature, you can call a function `f(codec:codec)` that takes a `codec` instance in its argument as below:

- `f(codec('utf-16'))` .. The most explicit way.
- `f('utf-16')` .. Implicit casting: from `string` to `codec`.

5.7 color Class

An instance of the `color` class represents a color data that consists of red, green, blue and alpha elements.

You can create a `color` instance by calling `color()` function.

There are class variables as shown below:

5.7.1 Predefined Variable

Variable	Type	Explanation
`color.names`	`string[]`	A list of color names that can be passed to `color()` function.
`color.zero`	`color`	Color instance equivalent with `color(0, 0, 0, 0)`
`color.black`	`color`	Color instance equivalent with `color(0, 0, 0, 255)`
`color.maroon`	`color`	Color instance equivalent with `color(128, 0, 0, 255)`
`color.green`	`color`	Color instance equivalent with `color(0, 128, 0, 255)`
`color.olive`	`color`	Color instance equivalent with `color(128, 128, 0, 255)`
`color.navy`	`color`	Color instance equivalent with `color(0, 0, 128, 255)`
`color.purple`	`color`	Color instance equivalent with `color(128, 0, 128, 255)`
`color.teal`	`color`	Color instance equivalent with `color(0, 128, 128, 255)`
`color.gray`	`color`	Color instance equivalent with `color(128, 128, 128, 255)`
`color.silver`	`color`	Color instance equivalent with `color(192, 192, 192, 255)`
`color.red`	`color`	Color instance equivalent with `color(255, 0, 0, 255)`
`color.lime`	`color`	Color instance equivalent with `color(0, 255, 0, 255)`
`color.yellow`	`color`	Color instance equivalent with `color(255, 255, 0, 255)`
`color.blue`	`color`	Color instance equivalent with `color(0, 0, 255, 255)`
`color.fuchsia`	`color`	Color instance equivalent with `color(255, 0, 255, 255)`
`color.aqua`	`color`	Color instance equivalent with `color(0, 255, 255, 255)`
`color.white`	`color`	Color instance equivalent with `color(255, 255, 255, 255)`

5.7.2 Property

A `color` instance has the following properties:

Property	Type	R/W	Explanation
a	number	R/W	Value of the alpha element.
b	number	R/W	Value of the blue element.
g	number	R/W	Value of the green element.
r	number	R/W	Value of the red element.

5.7.3 Cast Operation

A function that expects a `color` instance in its argument can also take a value of `symbol`, `string` and `list` as below:

- `symbol` .. Recognized as a color name to look up the color table.
- `string` .. Recognized as a color name to look up the color table.
- `list` .. Expected to contain elements in a format `[red, green, blue]` or `[red, green, blue, alpha]`.

With the above casting feature, you can call a function `f(c:color)` that takes a `color` instance in its argument as below:

- `f(color(‘purple))` .. The most explicit way.
- `f(‘purple)` .. Implicit casting: from `symbol` to `color`.
- `f(’purple’)` .. Implicit casting: from `string` to `color`.
- `f([128, 0, 128])` .. Implicit casting: from `list` to `color`.

5.7.4 Constructor

`color(args+):map {block?}`

Creates a `color` instance.

If `block` is specified, it would be evaluated with a block parameter `|c:color|`, where `c` is the created instance. In this case, the block's result would become the function's returned value.

There are two forms to call this function as below:

- `color(name:string, a?:number)` .. Creates an instance from color name and an optional alpha element. Predefined variable `color.names` is a list that contains available color names. A string in a format of `’#rrggbb’` that is used in HTML documents is also acceptable as a color name.
- `color(r:number, g?:number, b?:number, a?:number)` .. Creates an instance from RGB elements and an optional alpha element.

5.7.5 Method

`color#getgray()`

Calculates a gray scale from RGB elements in the `color` instance.

This is computed by a formula: `gray = 0.299 * red + 0.587 * blue + 0.114 * blue`.

`color#html()`

Returns a color string in a format of `'#rrggbb'` that is used in HTML documents.

`color#list():[alpha]`

Returns a list of RGB elements in a form `[r, g, b]`.

Specifying `:alpha` attribute would add the alpha element to the list.

5.8 complex Class

The `complex` class provides measures to calculate complex numbers.

You can create a `complex` instance by following ways:

- Calls `complex()` function with a real and imaginary part of numbers. e.g., `complex(2, 3)`
- Calls `complex.polar()` function with an absolute value and an argument in radius. e.g., `complex.polar(5, math.pi / 6)`
- Appending `j` suffix after a number literal would create an imaginal part of a complex numbrer. e.g., `2 + 3j`

5.8.1 Constructor

`complex(real:number, imag?:number):map {block?}`

Creates a `complex` instance with a real part `real` and an imaginary part `imag`.

If the argument `imag` is omitted, the imaginary part would be set to zero.

If `block` is specified, it would be evaluated with a block parameter `|n:complex|`, where `n` is the created instance. In this case, the block's result would become the function's returned value.

5.8.2 Method

`complex.polar(abs:number, arg:number):static:map:[deg] {block?}`

Creates a `complex` instance with an absolute number `abs` and an angle `arg` in polar coords.

The argument `arg` is specified in a unit of radian. You can give it a degree value by calling the function with `:deg` attribute.

If `block` is specified, it would be evaluated with a block parameter `|n:complex|`, where `n` is the created instance. In this case, the block's result would become the function's returned value.

`complex.roundoff(threshold:number => 1e-10) {block?}`

Returns a complex number with real and imaginary parts being rounded off.

The argument `threshold` specifies the threshold value for the round-off.

If `block` is specified, it would be evaluated with a block parameter `|n:complex|`, where `n` is the created instance. In this case, the block's result would become the function's returned value.

5.9 datetime Class

The `datetime` class provides measures to handle date and time information.

You can create a `datetime` instance by calling following functions:

- `datetime()` .. Creates an intance from specified date and time.
- `datetime.now()` .. Creates an instance with its date and time fields set as the current one.
- `datetime.today()` .. Creates an instance with its date field set as the current one. Its time fields, `hour`, `min`, `sec` and `usec`, are set to zero.

You can calculate a `datetime` with a `timedelta` to put its date and time values forward and backward.

5.9.1 Predefined Variable

Variable	Type	Explanation
`datetime.Sunday`	`number`	Assigned with number 0 that represents Sunday.
`datetime.Monday`	`number`	Assigned with number 1 that represents Monday.
`datetime.Tuesday`	`number`	Assigned with number 2 that represents Tuesday.
`datetime.Wednesday`	`number`	Assigned with number 3 that represents Wednesday.
`datetime.Thursday`	`number`	Assigned with number 4 that represents Thursday.
`datetime.Friday`	`number`	Assigned with number 5 that represents Friday.
`datetime.Saturday`	`number`	Assigned with number 6 that represents Saturday.

5.9.2 Property

A `datetime` instance has the following properties:

Property	Type	R/W	Explanation
`year`	`number`	R/W	Chritian year.
`month`	`number`	R/W	Month starting from 1. Numbers from 1 to 12 correspond to January to December.
`day`	`number`	R/W	Day in a month starting from 1.
`hour`	`number`	R/W	Hour in a day between 0 and 23.
`min`	`number`	R/W	Minute in an hour between 0 and 59.
`sec`	`number`	R/W	Second in a minute between 0 and 59.
`usec`	`number`	R/W	Millisecond in a second between 0 and 999.
`wday`	`number`	R	Week number starting from 0. Number from 0 to 6 corresponds to Sunday to Saturday.
`week`	`number`	R	Week symbol that takes one of the followings: `‘sunday`, `‘monday`, `‘tuesday`, `‘wednesday`, `‘thursday`, `‘friday`, `‘saturday`
`yday`	`number`	R	Day in a year starting from 1.
`unixtime`	`number`	R	Seconds passed from 00:00:00 on January 1st in 1970 in UTC.

5.9.3 Constructor

`datetime(year:number, month:number, day:number, hour:number => 0, min:number => 0, sec:number => 0, usec:r`

Creates an instance of `datetime` class based on the specified arguments.

Explanations of the arguments are shown below:

- `year` .. Christian year.
- `month` .. Month starting from 1. Numbers from 1 to 12 correspond to January to December.
- `day` .. Day in a month starting from 1.
- `hour` .. Hour in a day between 0 and 23.
- `min` .. Minute in an hour between 0 and 59.
- `sec` .. Second in a minute between 0 and 59.
- `usec` .. Millisecond in a second between 0 and 999.
- `minsoff` .. Timezone offset in minutes.

In default, the instance has a timezone offset based on the current system settings.

If `block` is specified, it would be evaluated with a block parameter `|dt:datetime|`, where `dt` is the created instance. In this case, the block's result would become the function's returned value.

5.9.4 Method

`datetime#clrtzoff():reduce`

Eliminates timezone offset information from the instance.

`datetime#format(format => 'w3c)`

Returns a string of the datetime properties based on the specified format. For the argument `format`, you can specify either a string of user-specific format or a symbol of predefined style.

A string of user-specific format contains following specifiers:

- `%d` .. day of month
- `%H` .. hour in 24-hour format
- `%I` .. hour in 12-hour format
- `%m` .. month
- `%M` .. minute
- `%S` .. second
- `%w` .. week number starting from 0 for Sunday.
- `%y` .. lower two digits of year
- `%Y` .. four digits of year

Below are the symbols of predefined styles:

- `w3c .. W3C style. eg) '2015-01-01T12:34:56+09:00'
- `http .. a style used in HTTP protocol. eg) 'Thu, 01 Jan 2015 12:34:56 +0900'
- `asctime .. a style used by the C function asctime(). eg) 'Thu Jan 1 12:34:56 +0900 2015'

datetime.isleap(year:number):static:map

Returns true if the specified year is a leap one.

datetime.monthdays(year:number, month:number):static:map {block?}

Returns a number of days that exists in the specified month.

If block is specified, it would be evaluated with a block parameter |n:number|, where n is the created instance. In this case, the block's result would become the function's returned value.

datetime.now():static:[utc] {block?}

Creates a datetime instance of the current time.

In default, the timezone offset is set to one in the system setting. Specifying :utc attribute would set the offset to 0.

If block is specified, it would be evaluated with a block parameter |dt:datetime|, where dt is the created instance. In this case, the block's result would become the function's returned value.

datetime.parse(str:string):static:map {block?}

Parses a string that describs date and time information and returns the datetime instance.

It is capable of parsing the following style:

- RFC1123 style. eg) 'Sat, 06 Nov 2010 08:49:37 GMT'
- RFC1036 style. eg) 'Saturday, 06-Nov-10 08:49:37 GMT'
- C's asctime() style. eg) 'Sat Nov 6 08:49:37 2010', 'Sat Nov 6 08:49:37 +0000 2010'
- W3C style. eg) '2010-11-06T08:49:37Z'

If block is specified, it would be evaluated with a block parameter |dt:datetime|, where dt is the created instance. In this case, the block's result would become the function's returned value.

datetime#settzoff(mins:number):reduce

Sets timezone offset in minutes.

datetime.time(hour:number => 0, minute:number => 0, sec:number => 0, usec:number => 0):static:map {block?}

Creates a datetime instance from time information. The date inforomation is set as 1st of January in the Christian year of 0.

If block is specified, it would be evaluated with a block parameter |dt:datetime|, where dt is the created instance. In this case, the block's result would become the function's returned value.

datetime.today():static:[utc] {block?}

Creates a `datetime` instance of today. All the time information are cleared to 0.

In default, the timezone offset is set to one in the system setting. Specifying `:utc` attribute would set the offset to 0.

If `block` is specified, it would be evaluated with a block parameter `|dt:datetime|`, where `dt` is the created instance. In this case, the block's result would become the function's returned value.

`datetime#utc()`

Calculates UTC time of the target `datetime` instance. An error occurs if the instance has no timezone offset

`datetime.weekday(year:number, month:number, day:number):static:map`

Returns a week number for the specified date, which starts from 0 for Sunday.

5.10 declaration Class

The `declaration` class provides information about argument's declaration defined in a function. You can get an iterator of `declaration` instances with the following measures that the `function` class provides:

- A property value: `function#decls`
- An instance method: `function.getdecls()`

Below is an example to print argument names declared in a function.

```
f(a, b, c, d) = {}
println(f.decls:*name)
```

5.10.1 Property

A `declaration` instance has the following properties:

Property	Type	R/W	Explanation
`symbol`	`symbol`	R	The name of the declaration in symbol.
`name`	`string`	R	The name of the declaration in string.
`default`	`expr`	R	The expression that provides a default value.

5.10.2 Method

`declaration#istype(type+:expr):map`

Return `true` if the declaration is defined as a type that is specified in the arguments.

The argument `type` has following formats:

- a single symbol.
- a sequence of symbols joined by a dot.

In the second format, a symbol on the left side indicates a container such as a module and a class.

Below is an example to check if the declaration is defined as `number` type.

```
decl.istype(`number)
```

Below is an example to check if the declaration is defined as `re.match` type, which is a type named `match` defined in `re` module.

```
decl.istype(`re.match)
```

You can also specify a type by describing factors in separate arguments like below:

```
decl.istype(`re, `match)
```

5.11 dict Class

The `dict` class provides measures to handle dictionary data that can seek values by indexing with their associated keys. You can specify values of `string`, `number` and `symbol` as a dictionary key.

You can create a `dict` instance by following measures:

- Calls `dict()` constructor.
- Calls a function named `%` that is an alias of `dict()` constructor.

Below are examples to create a `dict` instance:

```
dict {'first' => 1, 'second' => 2, 'third' => 3}
dict {{'first', 1}, {'second', 2}, {'third', 3}}
dict {'first', 1, 'second', 2, 'third', 3}

dict(['first' => 1, 'second' => 2, 'third' => 3])
dict([['first', 1], ['second', 2], ['third', 3]])
dict(['first', 1, 'second', 2, 'third', 3])

%{'first' => 1, 'second' => 2, 'third' => 3}
%{{'first', 1}, {'second', 2}, {'third', 3}}
%{'first', 1, 'second', 2, 'third', 3}
```

You can specify different type of values for keys in the same dictionary. In this case, values of different types are just recognized as different values.

Index Access

You can read and write element values in a `dict` with an indexer by giving it a key value which type is `string`, `number` or `symbol`. Below is an example:

```
x = %{'first' => 1, 'second' => 2, 'third' => 3}

println(x['second']) // prints `2`
x['third'] = 33      // replaces `3` with `33`
```

5.11.1 Constructor

dict(elems?):[icase] {block?}

Creates a `dict` instance.

It takes a list of key-value pairs in an argument as shown below:

```
d = dict([['apple', 100], ['grape', 200], ['banana', 80]])
```

Or, you can use a block to describe them like below:

```
d = dict {
    ['apple', 100], ['grape', 200], ['banana', 80]
}
```

You can specify values of `number`, `string` or `symbol` as dictionary keys.

You can also use the operator `=>` to create a key-value pair like below:

```
d = dict(['apple' => 100, 'grape' => 200, 'banana' => 80])
```

Below is an example using a block:

```
d = dict {
    'apple' => 100, 'grape' => 200, 'banana' => 80
}
```

The symbol `%` is an alias of the function `dict()`.

```
d = %{
    'apple' => 100, 'grape' => 200, 'banana' => 80
}
```

In default, if keys contain alphabet characters, different cases are distinguished. Appending the attribute `:icase` would ignore cases in them.

5.11.2 Method

dict#append(elems?):reduce:[overwrite,strict,timid] {block?}

Adds multiple key-value pairs. It takes a list of key-value pairs in an argument or in a block that has the same format with one for the function `dict()`.

If the specified key already exists in the dictionary, it would be overwritten. This behavior can be customized with the following attributes:

- `:overwrite` .. overwrite the existing one (default)
- `:strict` .. raises an error

- `:timid` .. keep the existing one

`dict#clear()`

Clears all the key-value pairs in the dictionary.

`dict#erase(key):map`

Erases a key-value pair that mathces the provided `key`.

The `key` is either `number`, `string` or `symbol`.

`dict#get(key, default?):map:[raise]`

Seeks a value that is associated with the specified `key`.

The method would return `nil` as its default value when the specified key doesn't exist in the dictionary. It would use different value if the argument `default` is specified.

Since the `default` value is also processed with implicit mapping, you have to apply `object#nomap()` method to it if you want to specify a list or an iterator as a default value.

When the attribute `:raise` is specified, an error occurs in the case of the key's absence.

Another measure to get a value associated with a key is to use an index operator. The following two codes have the same effect.

- `v = d['foo']`
- `v = d.get('foo'):raise`

`dict#haskey(key):map`

Returns `true` if the specified `key` exists in the dictionary.

`dict#items() {block?}`

Returns an iterator of key-value pairs in the dictionary.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`dict#keys() {block?}`

Returns an iterator of keys in the dictionary.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`dict#len()`

Returns the number of key-value pairs in the dictionary.

`dict#put(key, value):map:reduce:[overwrite,strict,timid]`

Adds a new key-value pair.

If the specified key already exists in the dictionary, it would be overwritten. This behavior can be customized with the following attributes:

- `:overwrite` .. overwrite the existing one (default)
- `:strict` .. raises an error
- `:timid` .. keep the existing one

Another measure to add a key-value pair is to use an index operator. The following two codes have the same effect.

- `d['foo'] = 3`
- `d.put('foo', 3)`

`dict#values() {block?}`

Returns an iterator of values in the dictionary.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.

- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

5.12 directory Class

The `directory` class handles information necessary to seek directory structure in a path. Its instance usually works with functions in `path` module: `path.dir()` and `path.walk()`.

Though the instance can be created by `directory()` function, you don't have to use it in many cases because a casting from `string` to `directory` instance works implicitly in a function call.

5.12.1 Constructor

`directory(pathname:string):map {block?}`

Creates a `directory` instance from the specified path name.

5.13 environment Class

The `environment` class provides measures to operate variables in an environment, which is a fundamental mechanism to store variables.

5.13.1 Method

`environment#getprop!(symbol:symbol):map`

`environment#lookup(symbol:symbol, escalate:boolean => true):map`

Looks up a specified symbol in the environment and returns the associated value. In default, if the symbol is not defined in the environment, it will be searched in environments outside of the current one. Set escalate flag to false in order to disable such an escalation behaviour. Returns false when the symbol could not be found.

`environment#setprop!(symbol:symbol, value):map`

5.14 error Class

The `error` class provides measures to access error information.

There is no measures to create an `error` instance. They're instantiated and passed to a block of `catch()` function when an error occurs within a `try` block in a `try-catch` sequence.

In the following code, `e` is an instance that contains information about an error that has occured in the `try` block.

```
try {
    // any jobs
} catch {|e:error|
    // ...
}
```

5.14.1 Predefined Variable

Variable	Explanation
`error.ArgumentError`	Argument error.
`error.ArithmeticError`	Arithmetic error.
`error.AttributeError`	Invalid attribute is specified.
`error.CodecError`	An error that is related to codec process.
`error.CommandError`	An error that could happen in command line.
`error.DeclarationError`	An error in a function\'s declarations.
`error.FormatError`	
`error.IOError`	
`error.ImportError`	
`error.IndexError`	
`error.IteratorError`	
`error.KeyError`	
`error.MemberAccessError`	
`error.MemoryError`	
`error.NameError`	
`error.NotImplementedError`	An error that could occur when a called function has no implemented body but an entry.
`error.OutOfRange`	Index number is out of range.
`error.ResourceError`	Resource error.
`error.RuntimeError`	Runtime error.
`error.SyntaxError`	Syntax error.
`error.SystemError`	System error.
`error.TypeError`	Type error.
`error.ValueError`	Invalid value is specified.
`error.ZeroDivisionError`	Zero-division occured in division or modulo operations.

5.14.2 Property

An `error` instance has the following properties:

Property	Type	R/W	Explanation
`source`	`string`	R	The name of the file that causes this error.
`lineno`	`number`	R	The number of line where the expression that causes this error starts.
`linenobtm`	`number`	R	The number of line where the expression that causes this error ends.
`postext`	`string`	R	A text that consists of a source name and a line number.
`text`	`string`	R	An error message. If an attribute ':lineno' is specified, it would contain a line number.
`trace`	`expr[]`	R	Stack trace.

5.15 expr Class

The `expr` class provides inromation about the language's syntax expression.

5.15.1 Property

An `expr` instance has the following properties:

Property	Type	R/W	Explanation
`attrfront`	`symbol[]`	R	Exists in "identifier" and "caller".
`attrs`	`symbol[]`	R	Exists in "identifier" and "caller".
`attrsopt`	`symbol[]`	R	Exists in "identifier" and "caller".
`block`	`expr`	R	Exists in "caller".
`blockparam`	`iterator`	R	Exists in "block".
`body`	`string`	R	Exists in "suffixed".
`car`	`expr`	R	Exists in "compound".
`cdr`	`iterator`	R	Exists in "compound".
`child`	`expr`	R	Exists in "unary".
`children`	`iterator`	R	Exists in "collector".
`left`	`expr`	R	Exists in "binary".
`lineno`	`number`	R	
`linenobtm`	`number`	R	
`operator`	`operator`	R	Exists in "unaryop", "binaryop" and "assign".
`postext`	`string`	R	
`right`	`expr`	R	Exists in "binary".
`source`	`string`	R	
`suffix`	`symbol`	R	Exists in "suffixed".
`symbol`	`symbol`	R	Exists in "identifier".
`trailer`	`expr`	R	Exists in "caller".
`typename`	`string`	R	
`typesym`	`symbol`	R	
`value`	`any`	R	Exists in "value".

5.15.2 Constructor

`expr(src:stream:r):map {block?}`

Parses a Gura script from the stream `src` and creates an `expr` instance.

If `block` is specified, it would be evaluated with a block parameter `|expr:expr|`, where `expr` is the created instance. In this case, the block's result would become the function's returned value.

5.15.3 Method

`expr#eval(env?:environment)`

Evaluates the `expr` instance.

If the argument `env` is specified, that environment is used for evaluation. If omitted, the current scope is used.

`expr.parse(script:string):static:map {block?}`

Parses a Gura script in the string `script` and creates an `expr` instance.

If `block` is specified, it will be evaluated with block parameter in a format of `|expr:expr|` where `expr` is the created instance.

`expr#textize(style?:symbol, indent?:string)`

Composes a script text from a content of `expr`.

Argument `style` specifies the text style output, which takes the following symbols:

- `‘crammed` .. Puts all the text in one line and removes volatile spaces.
- `‘oneline` .. Puts all the text in one line.
- `‘brief` .. Omits content of blocks and long strings with "`..`".
- `‘fancy` .. Prints in the most readable style. This is the default.

The argument `indent` specifies a string used for indentation. Its default is a sequence of four spaces.

`expr#tofunction(‘args*)`

Converts the `expr` into a function.

If the `expr` is a block that has a block parameter, that would be used as an argument list of the created function. Otherwise, the argument `args` declares the argument list.

It would be an error if `args` is specified and a block parameter exists as well.

`expr#unquote()`

Returns `expr` instance that has removed quote operator from the original `expr`.

`expr#write(dst:stream:w, style?:symbol, indent?:string)`

Outputs a script that describes the expression to the specified `stream`.

Argument `style` specifies the text style output, which takes the following symbols:

- `‘crammed` .. Puts all the text in one line and removes volatile spaces.
- `‘oneline` .. Puts all the text in one line.
- `‘brief` .. Omits content of blocks and long strings with "`..`".
- `‘fancy` .. Prints in the most readable style. This is the default.

The argument `indent` specifies a string used for indentation. Its default is a sequence of four spaces.

`expr#isunary()`

Returns `true` if expr is an expression of unary.

`expr#isunaryop()`

Returns `true` if expr is an expression of unaryop.

`expr#isquote()`

Returns `true` if expr is an expression of quote.

`expr#isbinary()`

Returns `true` if expr is an expression of binary.

`expr#isbinaryop()`

Returns `true` if expr is an expression of binaryop.

`expr#isassign()`

Returns `true` if expr is an expression of assign.

`expr#ismember()`

Returns `true` if expr is an expression of member.

`expr#iscollector()`

Returns `true` if expr is an expression of collector.

`expr#isroot()`

Returns `true` if expr is an expression of root.

`expr#isblock()`

Returns `true` if expr is an expression of block.

`expr#islister()`

Returns `true` if expr is an expression of lister.

`expr#isiterer()`

Returns `true` if expr is an expression of iterer.

`expr#iscompound()`

Returns `true` if expr is an expression of compound.

`expr#isindexer()`

Returns `true` if expr is an expression of indexer.

`expr#iscaller()`

Returns `true` if expr is an expression of caller.

`expr#isvalue()`

Returns `true` if expr is an expression of value.

`expr#isidentifier()`

Returns `true` if expr is an expression of identifier.

`expr#issuffixed()`

Returns `true` if expr is an expression of suffixed.

5.16 formatter Class

The `formatter` class provides information about a format specifier.

The function `printf()` has the following declaration:

```
printf('name: %s, age: %d\n', name, age)
```

The first argument is a string containing format specifiers like `%s` and `%d` that determine the manner on how the correspoding values `name` and `age` should be formatted. In the formatting mechanism, when the specifiers `%s` and `%d` appear, it would call methods `name.__format_s__()` and `age.__format_s__()` respectively which are format handlers responsible of formatting these values. In general, a format handler has a format like `__format_X__(fmt:formatter)` where `X` is the symbol of the specifier and `fmt` is a `formatter` instance that carries information about the associated specifier such as minimum width and a padding character. The handler must return a `string` as its result.

The table below summarizes associations between specifiers and the method name of their format handlers:

Specifier	Method Name
%d	__format_d__
%u	__format_u__
%b	__format_b__
%o	__format_o__
%x	__format_x__
%e	__format_e__
%f	__format_f__
%g	__format_g__
%s	__format_s__
%c	__format_c__

This feature is supposed to be used when you want your original class's instance properly formatted in `printf`. Below is an example to implement a format handler for the specifier `%d`:

```
A = class {

    // any implementations

    __format_d__(fmt:format) = {
        // returns a string for %d specifier.
    }
}

a = A()
printf('%d', a) // a.__format_d__() is called
```

5.16.1 Method

`formatter#getminwidth()`

Returns an expected minimum width for the field.

For example, with `'%3d'`, this method would return `3`.

`formatter#getpadding()`

Returns a string containing a padding character, a space or `'0'`.

In default, a space is used for padding. For example, with `'%3d'`, this method would return `' '`.

When a character `'0'` appears after `'%'`, that becomes the padding character. For example, with `'%03d'`, this method would return `'0'`.

`formatter#getplusmode()`

Returns a symbol that indicates an expected action when a positive number appears.

- `` `none `` .. No character ahead of the number.
- `` `space `` .. A space should be inserted.
- `` `plus `` .. A plus character should be inserted.

`formatter#getprecision()`

Returns an expected precision for the field.

For example, with `'%.3d'`, this method would return `3`.

`formatter#isleftalign()`

Returns `true` if the field is expected to be aligned on left.

For example, with `'%-3d'`, this method would return `true`.

`formatter#issharp()`

Returns `true` if the specifier sequence includes `'#'` flag, which means some literal prefixes such as `0x` are expected to be appended at the top.

For example, with `'%#x'`, this method would return `true`.

`formatter#isuppercase()`

Returns `true` if alphabet characters are expected to be shown in upper case.

Upper case characters are requested when a specifier such as `'%X'`, `'%E'` and `'%G'` is specified.

5.17 function Class

The `function` class provides measure to inspect information about the instance.

All the functions are instances of `function` class, so an implementation of a function means a realization of a `function` instance. You can also create the instance using `function()` constructor. The following two codes have the same result:

```
f(a:number, b:number, c:number) = {
    (a + b + c) / 3
}

f = function(a:number, b:number, c:number) {
    (a + b + c) / 3
}
```

Using `function()`, you can use variables prefixed by a dollar character so that they are automatically added to the argument list. In such a case, the variables are added to the argument list in the same order as they appear in the function body. The code below creates a function with a declaration `f($a, $b, $c)`.

```
f = function {
    ($a + $b + $c) / 3
}
```

You can use `&` as an alias of `function()` as shown below:

```
f = &{
    ($a + $b + $c) / 3
}
```

5.17.1 Property

A `function` instance has the following properties:

Property	**Type**	**R/W**	**Explanation**
`decls`	`iterator`	R	iterator of `declaration` instances that provide information about argument declaration the function defines.
`expr`	`expr`	R/W	an expression of the function.
`format`	`string`	R	a string showing a declared format of the function.
`fullname`	`string`	R	a full name of the function that is prefixed by a name of the module or the class it belongs to.
`name`	`string`	R	a name of the function in `string`.
`symbol`	`symbol`	R/W	a name of the function in `symbol`.

5.17.2 Operator

You can print a function's help from the interactive prompt using the unary operator " ". Below is an example to print the help of `printf()` function:

```
>>> ~printf
```

5.17.3 Constructor

`function('args*) {block}`

Creates a `function` instance with an argument list of `args` and a procedure body provided by `block`.

Following two codes have the same effect with each other.

- `f = function(a, b, c) { /* any job */ }`
- `f(a, b, c) = { /* any job */ }`

5.17.4 Method

function.getdecls(func:function):static:map

Creates an iterator of declaration instances that provide information about argument declaration that the function instance func defines.

This class method returns the same information as the property function#decls.

function.getexpr(func:function):static:map

Returns an expression of the function instance func.

It would return nil if the function is implemented with binary programs, not scripts.

This class method returns the same information as the property function#expr.

function.getformat(func:function):static:map

Returns a string showing a declared format of the function instance func.

This class method returns the same information as the property function#format.

function.getfullname(func:function):static:map

Returns a full name of the function instance func, which is prefixed by a name of the module or the class the instance belongs to.

This class method returns the same information as the property function#fullname.

function.getname(func:function):static:map

Returns a name of the function instance func in string type.

This class method returns the same information as the property function#name.

function.getsymbol(func:function):static:map

Returns a name of the function instance func in symbol type.

This class method returns the same information as the property function#symbol.

function#mathdiff(var?:symbol):reduce

Returns a function instance that computes derivation of the target function, which is expected to contain only mathematical procedures. An error occurs if the target function has any elements that have nothing to do with mathematics.

In default, it differentiates the target function with respect to its first argument. Below is an example:

```
>>> f(x) = math.sin(x)
>>> g = f.mathdiff()    // g is a function to compute math.cos(x)
```

Specify a symbol to argument var when you want to differentiate with respect to another variable.

You can check the result of derivation by seeing property function#expr like below:

```
>>> g.expr
'math.cos(x)
```

5.18 help Class

The `help` class provides measures to access help information associated with a `function` instance.

You can get a `help` instance from a `function` instance or a `class` by calling `help@function()` or `help@class()` respectively.

5.18.1 Property

A `help` instance has the following properties:

Property	Type	R/W	Explanation
`title`	`string`	R	The title of the help.
`format`	`string`	R	A name of the syntax format in which the help text is described such as `'markdown'`.
`lang`	`symbol`	R	A symbol of the natural language in which the help text is written. For example, `` `en `` for English and `` `ja `` for Japanese.
`text`	`string`	R	The help text.

5.18.2 Method

`help.text@iterator(lang:symbol):static {block?}`

Returns a help text for functions that return an iterator.

The argument `lang` is a symbol that specifies the language in which the text is written, e.g. `` `en `` for English and `` `ja `` for Japanese.

If `block` is specified, it would be evaluated with a block parameter `|str:string|`, where `str` is the created instance. In this case, the block's result would become the function's returned value.

`help.text@block(lang:symbol, varname:string, typename:string):static {block?}`

Returns a help text that for functions that take a block .

The argument `lang` is a symbol that specifies the language in which the text is written, e.g. `` `en `` for English and `` `ja `` for Japanese.

In the text, variable names would be replaced by `varname` and type names by `typename`.

If `block` is specified, it would be evaluated with a block parameter `|str:string|`, where `str` is the created instance. In this case, the block's result would become the function's returned value.

`help.presenter(format:string):static:void {block}`

Registers a presentation procedure with a name specified by the argument `format`.

The procedure is written in the block that takes block parameters: `|help:help|`.

5.19 image Class

The `image` class provides following measures to handle graphic image data:

- Reads image data from a file.
- Writes image data to a file.
- Apply some modifications on image data including rotation, resize and color conversion.

Acceptable image data formats can be extended by importing modules. Below is a table to show image formats and name of modules that handle them. The string listed in "imagetype" column shows a name that is used by functions `image()`, `image#read()` and `image#write()` to explicitly specify the image data format in a process of reading and writing files.

Image Format	Module Name	imagetype
BMP	`bmp`	`'bmp'`
GIF	`gif`	`'gif'`
JPEG	`jpeg`	`'jpeg'`
Microsoft Icon	`msico`	`'msico'`
PNG	`png`	`'png'`
PPM	`ppm`	`'ppm'`
TIFF	`tiff`	`'tiff'`

5.19.1 Property

An `image` instance has the following properties:

Property	Type	R/W	Explanation
`format`	`symbol`	R	Takes one of the following symbols indicating what elements are stored in the memory: `` `rgb `` .. red, green and blue `` `rgba `` .. red, green, blue and alpha
`width`	`number`	R	Image width.
`height`	`number`	R	Image height.
`palette`	`palette`	R/W	A `palette` instance associated with this image. If there is no palette associated, this property returns `nil`.

5.19.2 Constructor

`image(args+):map {block?}`

Returns an image instance with specified characteristics. There are three forms to call the function as below:

- `image(format:symbol)` .. Creates an image instance of the specified format without buffer allocated.
- `image(format:symbol, width:number, height:number, color?:color)` .. Allocates an image buffer with the specified size and fills it with the color.
- `image(stream:stream, format?:symbol, imagetype?:string)` .. Reads image data from the stream and allocates necessary buffer in which the read data is stored.

The argument `format` specifies what elements are stored in the memory and takes one of the following symbols:

- `` `rgb `` .. red, green and blue
- `` `rgba `` .. red, green, blue and alpha

In the third form, the format of the image data is determined by the byte sequence of the stream data and its file name.

You can also explicitly specify the image data format by the argument `imagetype`.

5.19.3 Method

`image#allocbuff(width:number, height:number, color?:color):reduce`

Allocates a specified size of buffer in the `image` instance that is supposed to has no buffer allocated.

The allocated buffer will be filled with `color`. If omitted, it will be filled with zero value.

An error occurs in following cases:

- It fails to allocate necessary buffer.
- The `image` instance already has allocated buffer.

`image#blur(radius:number, sigma?:number) {block?}`

Returns a new image that blurs the original image with the given parameters.

If `block` is specified, it would be evaluated with a block parameter `|img:image|`, where `img` is the created instance. In this case, the block's result would become the function's returned value.

`image#clear():reduce`

Fills the buffer in the `image` instance with zero value.

This has the same effect with calling `image#fill()` with `color.zero`.

This method returns the reference to the target instance itself.

`image#crop(x:number, y:number, width?:number, height?:number):map {block?}`

Returns a new image instance of the extracted area of the source image.

The extracted area is specified by the following arguments:

- `x` .. The left position.
- `y` .. The top position.
- `width` .. The width. If it's omitted or specified with `nil`, the whole area on the right of `x` will be extracted.
- `height` .. The height. If it's omitted or specified with `nil`, the whole area on the bottom of `y` will be extracted.

If `block` is specified, it would be evaluated with a block parameter `|img:image|`, where `img` is the created instance. In this case, the block's result would become the function's returned value.

`image#delpalette():reduce`

Deletes a `palette` instance that belongs to the `image`.

This method returns the reference to the target instance itself.

`image#extract(x:number, y:number, width:number, height:number, element:symbol, dst):reduce`

Extracts the element values within the specified area of the image, and store them into a list. The argument `x` and `y` specifies the left-top position, and `width`, and `height` does the size of the area.

The argument `element` takes the following symbol that specifies which element should be extracted:

- `` `r `` .. red
- `` `g `` .. green
- `` `b `` .. blue
- `` `a `` .. alpha

The argument `dst` specifies the variable into which the extracted data is stored, which must be a list that has enough space to store the data.

This method returns the reference to the target instance itself.

`image#fill(color:color):reduce`

Fills the whole image with the specified color.

This method returns the reference to the target instance itself.

`image#fillrect(x:number, y:number, width:number, height:number, color:color):map:reduce`

Fills the specified area with the specified color. The argument `x` and `y` specifies the left-top position, and `width`, and `height` does the size of the area.

This method returns the reference to the target instance itself.

`image#flip(orient:symbol):map {block?}`

Returns a new `image` instance that flips the source image horizontally or vertically. You can specify the following symbol to the `orient` argument.

- `` `horz `` .. flips horizontally.
- `` `vert `` .. flips vertically.
- `` `both `` .. flips both horizontally and vertically. This has the same effect with rotating the image 180 degrees.

If `block` is specified, it would be evaluated with a block parameter `|img:image|`, where `img` is the created instance. In this case, the block's result would become the function's returned value.

`image#getpixel(x:number, y:number):map {block?}`

Returns a color of a pixel data at the specified position.

If `block` is specified, it would be evaluated with a block parameter `|c:color|`, where `c` is the created instance. In this case, the block's result would become the function's returned value.

`image#grayscale() {block?}`

Returns a new image instance that converts the source image into gray scale.

If `block` is specified, it would be evaluated with a block parameter `|img:image|`, where `img` is the created instance. In this case, the block's result would become the function's returned value.

`image#mapcolorlevel(map@r:array@uint8, map@g?:array@uint8, map@b?:array@uint8) {block?}`

Returns a new image that converts color levels according to the given table.

Each of the arguments `map@r`, `map@g` and `map@b` is an instance of array@uchar containing 256 numbers that range between 0 and 255 and corresponds to elements red, green and blue respectively. An element value in the source image becomes an index of the list and the indexed value will be stored as a converted element value.

If you want to apply a mapping table to all the elements, call the method with a single argument like `image#mapcolorlevel(map)`.

If `block` is specified, it would be evaluated with a block parameter `|img:image|`, where `img` is the created instance. In this case, the block's result would become the function's returned value.

`image#paste(x:number, y:number, src:image, width?:number, height?:number, xoffset:number => 0, yoffset:nu`

Pastes the source image `src` onto the target image instance at the specified position.

The argument `width`, `height`, `xoffset` and `yoffset` specify the source image's area to be pasted. If they're omitted, the whole image will be pasted.

The argument `a` specifies the alpha value that is put on the target image.

This method returns the reference to the target instance itself.

`image#putpixel(x:number, y:number, color:color):map:reduce`

Puts a color on the specified position.

This method returns the reference to the target instance itself.

`image#size()`

Returns the image size as a list `[width, height]`.

`image#store(x:number, y:number, width:number, height:number, element:symbol, src):reduce`

`image#read(stream:stream:r, imagetype?:string):map:reduce`

Reads image data from a stream.

The format of the image data is determined by the byte sequence of the stream data and its file name.

You can also explicitly specify the image data format by the argument `imagetype`.

This method returns the reference to the target instance itself.

`image#reducecolor(palette?:palette) {block?}`

Creates an image that reduces colors in the original image with a set of colors in the given palette. The specified palette would be associated with the created image.

If no argument is specified, the associated palette would be used. In this case, an error occurs if there's no palette associated.

If `block` is specified, it would be evaluated with a block parameter `|img:image|`, where `img` is the created instance. In this case, the block's result would become the function's returned value.

`image#replacecolor(colorOrg:color, color:color, tolerance?:number):reduce`

Replaces pixels that have a color matching `colorOrg` with the `color`.

The argument `tolerance` specifies an acceptable distance for the matching. If omitted, only an exact match is acceptable.

This method returns the reference to the target instance itself.

`image#resize(width?:number, height?:number):map:[box,ratio] {block?}`

Resizes the image to the size specified by `width` and `height` and returns the result.

- When both `width` and `height` are specified, the image would be resized to the size.
- When `width` is specified and `height` is omitted or `nil`, the resized height would be calculated from the width so that they keep the same ratio as the original.
- When `width` is `nil` and `height` is specified, the resized width would be calculated from the height so that they keep the same ratio as the original.

The following attributes are acceptable:

- `:box` .. When only `width` is specified, the same value is set to `height`.
- `:ratio` .. Treats values of `width` and `height` as magnifying ration instead of pixel size.

If `block` is specified, it would be evaluated with a block parameter `|img:image|`, where `img` is the created instance. In this case, the block's result would become the function's returned value.

`image#rotate(rotate:number, background?:color):map {block?}`

Creates an image that rotates the original image by the specified angle.

The argument `angle` specifies the rotation angle in degree unit, and positive numbers for counterclockwise direction and negative for clockwise direction.

The created instance has a size that exactly fits the rotated image. The argument `background` specifies the color of pixels to fill the empty area that appears after rotation. If omitted, the color that has all elements set to zero is used for filling.

If `block` is specified, it would be evaluated with a block parameter `|img:image|`, where `img` is the created instance. In this case, the block's result would become the function's returned value.

`image#scan(x?:number, y?:number, width?:number, height?:number, scandir?:symbol) {block?}`

Returns an iterator that scans pixels in the image.

The arguments `x`, `y`, `width` and `height` specify the image area to scan. The argument `scandir` specifies the scan direction and takes one of the following symbol:

Symbol	Start Pos	Direction
`left_top_horz	left-top	horizontal
`left_top_vert	left-top	vertical
`left_bottom_horz	left-bottom	horizontal
`left_bottom_vert	left-bottom	vertical
`right_top_horz	right-top	horizontal
`right_top_vert	right-top	vertical
`right_bottom_horz	right-bottom	horizontal
`right_bottom_vert	right-bottom	vertical

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`image#setalpha(a:number, color?:color, tolerance?:number):reduce`

Replaces the alpha element of all the pixels in the image with the value specified by `a`.

If the argument `color` is specified, alpha element of pixels that match with that color would be replaced. The argument `tolerance` specifies the distance within which the color is determined as matched.

This method returns the reference to the target instance itself.

`image#thumbnail(width?:number, height?:number):map:[box] {block?}`

Resizes the image so that it fits within a rectangular area specified by `width` and `height` and returns the result.

If `block` is specified, it would be evaluated with a block parameter `|img:image|`, where `img` is the created instance. In this case, the block's result would become the function's returned value.

`image#write(stream:stream:w, imagetype?:string):map:reduce`

Writes image data to a stream.

The format of the image data is determined by the stream's file name.

You can also explicitly specify the image data format by the argument `imagetype`.

This method returns the reference to the target instance itself.

5.20 list/iterator Class

The `list` class provides measures to handle a list structure, which stores values on memory that can be accessed by indexer.

The `iterator` class provides measures to operate an iterator, which iterates values that come from containers and streams.

5.20.1 List-specific Features

Creating List

There are several ways to create a list.

```
[3, 1, 4, 1, 5, 9]
@{3, 1, 4, 1, 5, 9}
```

Index Access

You can read and write element values in a list with an indexer by giving it an index number starting from zero. Below is an example:

```
x = [`A, `B, `C, `D, `E, `F]

println(x[2]) // prints `C
x[4] = `e     // replaces `E with `e
```

Function to Create list Instance

`list(value+)`

Creates a new list from given values in its argument list. If a given value is a list or an iteartor, elements it contains are added to the created list.

`xlist(value+)`

Creates a new list from given values except for `nil` in its argument list. If a given value is a list or an iteartor, elements it contains are added to the created list.

`set(iter+:iterator):[and,or,xor]`

Creates a new list that contains unique values from given iterators in its argument list.

In default, all the elements in each iterators are added to the created list. Specifying the following attributes would apply a filtering condition.

- `:and` .. Elements that exist in all the iterators are added.
- `:or` .. All the elements are added. This is the default behavior.
- `:xor` .. Elements that exist in only one iterator are added.

`xset(iter+:iterator):[and,or,xor]`

Creates a new list that contains unique values except for `nil` from given iterators in its argument list.

In default, all the elements in each iterators are added to the created list. Specifying the following attributes would apply a filtering condition.

- `:and` .. Elements that exist in all the iterators are added.
- `:or` .. All the elements are added. This is the default behavior.
- `:xor` .. Elements that exist in only one iterator are added.

Method Specific to list Class

`list#add(elem+):reduce`

Add specified items to the list.

`list#append(elem+):reduce`

Adds specified items to the list. If the item is a list or an iterator, each element in such an item is added to the list.

`list#clear():reduce`

Clear the content of the list.

`list#combination(n:number) {block?}`

Creates an iterator that generates lists that contain elements picked up from the original list in a combination manner.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`list#erase(idx*:number):reduce`

Erases elements at the specified indices.

`list#first()`

Returns a first value in the list. An error occurs when the list is empty.

`list#get(index:number):flat:map`

Returns a value stored at the specified index in the list. An error occurs when the index is out of range.

`list#insert(idx:number, elem+):reduce`

Insert specified items to the list from the selected index.

`list#isempty()`

Return true if the list is empty.

`list#last()`

Returns a last value in the list. An error occurs when the list is empty.

`list#permutation(n?:number) {block?}`

Creates an iterator that generates lists that contain elements picked up from the original list in a permutation manner.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`list#put(index:number, value:nomap):map:reduce`

Stores a value at the specified index in the list. An error occurs when the index is out of range.

`list#shift():[raise]`

Shifts the elements of the list. If the content of the list is [1, 2, 3, 4], it becomes [2, 3, 4] after calling this method. In default, no error occurs even when the list is empty. To raise an error for executing this method on an empty list, specify :raise attribute.

`list#shuffle():reduce`

Shuffle the order of the list content based on random numbers.

`list.zip(values+):static {block?}`

Creates an iterator generating lists that bind given argument values. When the value is a list or an iterator, each item in it would be zipped.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

5.20.2 Iterator-specific Features

Function to Create iterator Instance

`iterator(value+) {block?}`

Creates an iterator that combines iterators given in the argument.

If an argument is not an iterator, that would be added as an element.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

Method Specific to iterator Class

`iterator#delay(delay:number) {block?}`

Creates an iterator that returns each element with an interval time specified by the argument `delay` in seconds.

`iterator#finite():reduce`

Marks the iterator as a finite one by clearing its infinite flag.

This method returns the target instance itself.

`iterator#infinite():reduce`

Marks the iterator as an infinite one by setting its infinite flag.

This method returns the target instance itself.

`iterator#isinfinite()`

Returns `true` if the iterator is infinite one.

The trait of iterator's infinity is used to avoid an endless process by evaluating an infinite iterator. An attempt to evaluate an infinite iterator such as creation of a list from it would occur an error.

`iterator#next()`

Returns a next element of the iterator. This operation updates the iterator's internal status.

`iterator#repeater()`

Makes the iterator behave as a "repeater". This would allow the iterator be evaulated when it appears as an element of another "repeater" iterator.

Below is an example:

```
x = repeat(3):iter {
    ['apple', 'orange', 'grape'].each()
}
println(x)
// Just prints iterator instance three times
// since x can't evaluate the internal iterator.

x = repeat(3):iter {
    ['apple', 'orange', 'grape'].each().repeater()
}
println(x)
// Prints 'apple', 'orange' and  'grape' three times
// after evaluating the internal iterator.
```

5.20.3 Method Common to Both list and iterator Classes

`iterable#after(criteria) {block?}`

Creates an iterator that picks up elements that appear at positions after the criteria is evaluated to be `true`.

You can specify a function, a list or an iterator as the criteria.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.

- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`iterable#align(n:number, value?) {block?}`

Creates an iterator that returns the specified number of elements in the source iterator. If the number is larger than the length of the source iterator, the lacking part is filled with `value`. If the argument `value` is omitted, `nil` is used for the filling.

Below is an example to specify a number less than the source length:

```
x = [`A, `B, `C, `D, `E, `F].align(3)
// x generates `A, `B, `C.
```

Below is an example to specify a number that exceeds the source length:

```
x = [`A, `B, `C, `D, `E, `F].align(8)
// x generates `A, `B, `C, `D, `E, `F, nil, nil.
```

`iterable#and()`

Calculates a logical AND result of all the values in the iterable.

`iterable#argmax():[indices,last_index]`

Returns a position index where the maximum value is found at first.

The following attributes modify the behavior:

- `:last_index` .. returns an index where the maximum value is found at last.
- `:indices` .. returns a list of all indices where the maximum value is found.

Calling of methods `iterable#argmas()` and `iteraboe#max()` have the same effect when attributes are specified as follows:

- `iterable.argmax()` and `iterable.max():index`
- `iterable.argmax():last_index` and `iterable.max():last_index`
- `iterable.argmax():indices` and `iterable.max():indices`

`iterable#argmin():[indices,last_index]`

Returns a position index where the minimum value is found at first.

The following attributes modify the behavior:

- `:last_index` .. returns an index where the minimum value is found at last.
- `:indices` .. returns a list of all indices where the minimum value is found.

Calling of methods `iterable#argmas()` and `iteraboe#max()` have the same effect when attributes are specified as follows:

- `iterable.argmax()` and `iterable.max():index`
- `iterable.argmax():last_index` and `iterable.max():last_index`
- `iterable.argmax():indices` and `iterable.max():indices`

`iterable#before(criteria) {block?}`

Creates an iterator that extracts elements in the iterable before criteria is evaluated as true. You can specify a function object, a list or an iterator as the criteria.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`iterable#contains(value)`

Returns `true` if the specified value appears in the iterable.

`iterable#count(criteria?)`

Returns a number of elements that matches the given criteria which is a single-argument function or a value.

When a function is applied, it counts the number of true after evaluating element value with the function. If a value is applied, it counts the number of elements that are equal to the value.

`iterable#cycle(n?:number) {block?}`

Creates an iterator that iterates elements in the source iterator cyclically.

The argument `n` specifies the number of elements the created iterator returns. If omitted, it would iterates elements infinitely.

Below is an example:

```
x = [`A, `B, `C, `D, `E].cycle()
// x generates `A, `B, `C, `D, `E, `A, `B, `C, `D, `E, `A, `B, ..
```

`iterable#each() {block?}`

Creates an iterator that iterates each element in the list.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`iterable#filter(criteria?) {block?}`

Creates an iterable that filters values in the source iterable by a criteria.

A criteria can be an iterable or a function instance.

- When the criteria is an iterable, the created iterator would scan the source and the criteria iterable simultaneously and would return a value of the source when the corresponding criteria value is evaluated as `true`.
- When the criteria is a function instance, the created iterator would give it a value of the source as an argument and would return the value when the function has returned `true`.

Below is an example to use an iterable as its criteria:

```
x = [3, 1, 4, 1, 5, 9]
y = filter(x > 3)
// (x > 3) makes a list [false, false, true, false, true, true]
// y generates 4, 5, 9
```

Below is an example to use a function as its criteria:

```
x = [3, 1, 4, 1, 5, 9]
y = filter(&{$x > 3})
// y generates 4, 5, 9
```

`iterable#find(criteria?):[index]`

`iterable#flatten():[bfs,dfs] {block?}`

Creates an iterator that searches items recursively if they are lists or iterators.

Specifying an attribute could customize searching order as below:

- `:dfs` .. Searches in depth-first order. This is the default behavior.
- `:bfs` .. Searches in breadth-first order.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

Below is an example:

```
x = [[`A, `B, `C], [`D, `E, [`F, `G, `H], `I, `J], `K, `L]

y = x.flattten():dfs
// y generates `A, `B, `C, `D, `E, `F, `G, `H, `I, `J, `K, `L

y = x.flatten():bfs
// y generates `K, `L, `A, `B, `C, `D, `E, `I, `J, `F, `G, `H
```

`iterable#fold(n:number, nstep?:number):map:[iteritem,neat] {block?}`

Creates an iterator that packs `n` elements of the source iterator into a list and returns it as its element.

The argument `nstep` specifies the shift amount to the next packing.If omitted, the next packing is shifted by `n` elements.

Specifying the attribute `:iteritem` returns an iterator as its element instead of a list

If the last packing doesn't satisfy `n` elements, its list would be shorter than `n`. When specifying the attribute `:neat`, such an immature list would be eliminated.

Following is an example to fold elements by 3:

```
x = [`A, `B, `C, `D, `E, `F, `G, `H].fold(3)
// x generates [`A, `B, `C], [`D, `E, `F], [`G, `H].
```

Following is an example to fold elements by 3 with a step of 2:

```
x = [`A, `B, `C, `D, `E, `F, `G, `H].fold(3, 2)
// x generates [`A, `B, `C], [`C, `D, `E], [`E, `F, `G], [`G, `H].
```

`iterable#format(format:string):map {block?}`

Creates an iterator that converts element values in the source iterable into strings depending on formatter specifier in `format`.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`iterable#head(n:number):map {block?}`

Creates an iterator that takes the first `n` elements from the source iterable.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`iterable#join(sep?:string):map`

Joins all the elements in the iterable as strings while inserting the specified separator `sep` and returns the result.

If an element is not a `string` value, it would be converted to a `string` before being joined.

`iterable#joinb()`

Joins all the `binary` values in the iterable and returns the result.

`iterable#len()`

Returns the length of the iterable.

`iterable#map(func:function) {block?}`

Creates an iterator that generates element values after applying the specfied function on them. The function must take one argument.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`iterable#max():[index,indices,last_index]`

Returns the maximum value in the iterable.

It would return a position index where the maximum value is found when one of the following attributes is specified:

- `:index` .. an index of the maximum value.
- `:last_index` .. an index where the maximum value is found at last.
- `:indices` .. a list of all indices where the maximum value is found.

`iterable#mean()`

Calculates an average of elements in the iterable.

It can work on an iterable with elements of type that supports addition and division operators. Below is a list of acceptable value types:

- `number`
- `complex`
- `rational`

`iterable#min():[index,indices,last_index]`

Returns the minimum value in the iterable.

It would return a position index where the minimum value is found when one of the following attributes is specified:

- `:index` .. an index of the minimum value.
- `:last_index` .. an index where the minimum value is found at last.
- `:indices` .. a list of all indices where the minimum value is found.

`iterable#nilto(replace) {block?}`

Creates an iterator that converts `nil` in the source iterable to the specified value.

`iterable#offset(n:number) {block?}`

Creates an iterator that returns skips the first `n` elements in the source iterable.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

Below is an example:

```
x = [`A, `B, `C, `D, `E, `F, `G, `H].offset(3)
// x generates `D, `E, `F, `G, `H
```

`iterable#or()`

Calculates a logical OR result of all the values in the iterable.

`iterable#pack(format:string) {block?}`

Creates a `binary` instance that has packed elements in the iterable according to specifiers in the `format`.

A specifier has a format of "`nX`" where `X` is a format character that represents a packing format and `n` is a number of packing size. The number can be omitted, and it would be treated as `1` in that case.

Following format characters would take a `number` value from the argument list and pack them into a binary sequence.

- `b` .. A one-byte signed number.
- `B` .. A one-byte unsigned number.
- `h` .. A two-byte signed number.

- `H` .. A two-byte unsigned number.
- `i` .. A four-byte signed number.
- `I` .. A four-byte unsigned number.
- `l` .. A four-byte signed number.
- `L` .. A four-byte unsigned number.
- `q` .. A eight-byte signed number.
- `Q` .. A eight-byte unsigned number.
- `f` .. A float-typed number occupying four bytes.
- `d` .. A double-typed number occupying eight bytes.

As for them, the packing size `n` means the number of values to be packed.

Following format characters would take a `string` value from the argument list and pack them into a binary sequence.

- `s` .. Packs a sequence of UTF-8 codes in the string. The packing size `n` means the size of the room in bytes where the character codes are to be packed. Only the sequence within the allocated room would be packed. If the string length is smaller than the room, the lacking part would be filled with zero.
- `c` .. Picks the first byte of the string and packs it as a one-byte unsigned number. The packing size `n` means the number of values to be packed.

Following format character would take no value from the argument list.

- `x` .. Fills the binary with zero. The packing size `n` means the size of the room in bytes to be filled with zero.

The default byte-order for numbers of two-byte, four-byte and eight-byte depends on the system the interpreter is currently running. You can change it by the following specifiers:

- `@` .. System-dependent order.
- `=` .. System-dependent order.
- `<` .. Little endian
- `>` .. Big endian
- `!` .. Big endian

You can specify an asterisk character "*" for the number of packing size that picks that number from the argument list.

You can specify encoding name embraced with "{" and "}" in the format to change coding character set while packing a string with format character "s" from UTF-8.

`iterable#pingpong(n?:number):[sticky,sticky@top,sticky@btm] {block?}`

Creates an iterator that iterates elements in the source iterator from top to bottom, and then from bottom to top repeatedly.

The argument `n` specifies the number of elements the created iterator returns. If omitted, it would iterates elements infinitely.

Below is an example:

```
x = [`A, `B, `C, `D, `E].pingpong()
// x generates `A, `B, `C, `D, `E, `D, `C, `B, `A, `B, ..
```

The following attributes specify whether the elements on top and bottom are duplicated:

- `:sticky` .. Duplicate the top and bottom elements.
- `:sticky@top` .. Duplicate the top element.
- `:sticky@btm` .. Duplicate the bottom element.

Below is an example:

```
x = [`A, `B, `C, `D, `E].pingpong():sticky
// x generates `A, `B, `C, `D, `E, `E, `D, `C, `B, `A, `A, `B, ..
```

`iterable#print(stream?:stream:w):void`

Prints elements to the specified `stream`.

If omitted, they are printed to the standard output.

`iterable#printf(format:string, stream?:stream:w):void`

Prints items in the iterable by using the format.

`iterable#println(stream?:stream:w):void`

`iterable#rank(directive?) {block?}`

Creates an iterable of rank numbers for elements after sorting them.

In default, they are sorted in an ascending order. This means that, if two elements `x` and `y` has the relationship of `x < y`, `x` would be placed before `y`. You can change the order by specifying the argument `directive` with the following symbols:

- `` `ascend `` .. Sorts in an ascending order. This is the default.
- `` `descend `` .. Sorts in a descending order.

You can also put a function to the argument `directive` that takes two arguments `x` and `y` and is expected to return numbers below:

- `x == y` .. Zero.
- `x < y` .. A number less than zero.
- `x > y` .. A number greater than zero.

When an attribute :stable is specified, the original order shall be kept for elements that are determined as the same.

`iterable#reduce(accum) {block}`

Evaluates a block with a parameter format `|value, accum|` and leaves the result as the next `accum` value.

It returns the final `accum` value as its result.

Below is an example to calculate summation of the elements:

```
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
n = x.reduce(0) {|value, accum| value + accum}
// n is 55
```

`iterable#replace(value, replace) {block?}`

Creates an iterator that replaces the `value` in the original iterablewith the value of `replace`.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`iterable#reverse() {block?}`

Creates an iterator that iterates elements in the source iterable from tail to top.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`iterable#roundoff(threshold:number => 1e-10) {block?}`

Creates an iterator that replaces a number with zero if it is less than the specified `threshold`.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`iterable#runlength() {block?}`

Creates an iterator that counts the number of consecutive same value and generates elements in a form of `[cnt, value]` where `cnt` indicates how many `value` appears in a row.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

Below is an example:

```
x = [`A, `A, `B, `C, `C, `C, `D, `D].runlength()
// x generates [2, `A], [1, `B], [3, `C], [2, `D]
```

`iterable#since(criteria) {block?}`

Creates an iterator that picks up each element in the iterable since criteria is evaluated as true. You can specify a function object, a list or an iterator as the criteria.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.

- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`iterable#skip(n:number) {block?}`

Creates an iterator that skips `n` elements before picking up next element.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

Below is an example:

```
x = [`A, `B, `C, `D, `E, `F, `G, `H].skip(2)
// x generates `A, `D, `G
```

`iterable#skipnil() {block?}`

Creates an iterator that skips `nil` in the source iterable.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.

- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

Below is an example:

```
x = [‘A, nil, ‘C, nil, nil, ‘F, nil, ‘H].skipnil()
// x generates ‘A, ‘C, ‘F, ‘H
```

`iterable#sort(directive?, keys[]?):[stable] {block?}`

Creates an iterator of elements after sorting them.

In default, they are sorted in an ascending order. This means that, if two elements `x` and `y` has the relationship of `x < y`, `x` would be placed before `y`. You can change the order by specifying the argument `directive` with the following symbols:

- `‘ascend` .. Sorts in an ascending order. This is the default.
- `‘descend` .. Sorts in a descending order.

You can also put a function to the argument `directive` that takes two arguments `x` and `y` and is expected to return numbers below:

- `x == y` .. Zero.
- `x < y` .. A number less than zero.
- `x > y` .. A number greater than zero.

When an attribute :stable is specified, the original order shall be kept for elements that are determined as the same. If the argument `keys` is specified, it would be used as a key instead of element values.

`iterable#std():[p]`

Calculates a standard deviation of elements in the iterable.

`iterable#sum()`

Calculates a summation of elements in the iterable.

It can work on an iterable with elements of a value type that supports addition operator. Below is a list of acceptable value types:

- `number`
- `complex`
- `string`
- `rational`
- `timedelta`

`iterable#tail(n:number) {block?}`

Creates an iterator that takes the last `n` elements from the source iterable.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`iterable#until(criteria) {block?}`

Creates an iterator that picks up each element in the list until criteria is evaluated as true. You can specify a function object, a list or an iterator as the criteria.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`iterable#var():[p]`

Calculates a variance of elements in the iterable.

`iterable#while (criteria) {block?}`

Creates an iterator that picks up each element in the list while criteria is evaluated as true. You can specify a function object, a list or an iterator as the criteria.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

5.21 memory Class

An instance of the `memory` class represents a memory that is stored in `array` instances.

5.21.1 Property

A `memory` instance has the following properties:

Property	Type	R/W	Explanation
`p`	`pointer`	R	Returns a `pointer` instance that accesses the memory. This result is equivalent to that of calling the method `memory#pointer()`
`size`	`number`	R	Returns the memory size in bytes.

5.21.2 Constructor

`memory(bytes:number):map {block?}`

5.21.3 Method

`memory#array@int8():map {block?}`

Creates an `array@int8` instance that accesses the content of the target `memory` instance.

`memory#array@uint8():map {block?}`

Creates an `array@uint8` instance that accesses the content of the target `memory` instance.

`memory#array@int16():map {block?}`

Creates an `array@int16` instance that accesses the content of the target `memory` instance.

`memory#array@uint16():map {block?}`

Creates an array@uint16 instance that accesses the content of the target memory instance.

memory#array@int32():map {block?}

Creates an array@int32 instance that accesses the content of the target memory instance.

memory#array@uint32():map {block?}

Creates an array@uint32 instance that accesses the content of the target memory instance.

memory#array@int64():map {block?}

Creates an array@int64 instance that accesses the content of the target memory instance.

memory#array@uint64():map {block?}

Creates an array@uint64 instance that accesses the content of the target memory instance.

memory#array@float():map {block?}

Creates an array@float instance that accesses the content of the target memory instance.

memory#array@double():map {block?}

Creates an array@double instance that accesses the content of the target memory instance.

memory#dump(stream?:stream:w):void:[upper]

Prints a hexadecimal dump from the content of the memory to the standard output. If the argument stream is specified, the result would be output to the stream.

In default, hexadecimal digit are printed with lower-case characters. Specifying an attribute :upper would output them with upper-case characters instead.

Example:

```
>>> b'A quick brown fox jumps over the lazy dog.'.dump():upper
41 20 71 75 69 63 6B 20 62 72 6F 77 6E 20 66 6F  A quick brown fo
78 20 6A 75 6D 70 73 20 6F 76 65 72 20 74 68 65  x jumps over the
20 6C 61 7A 79 20 64 6F 67 2E                     lazy dog.
```

memory#pointer(offset?:number) {block?}

Returns a pointer instance that has an initial offset specified by the argument offset. If the argument is omitted, it would return a pointer instance that points to the top of the memory.

If block is specified, it would be evaluated with a block parameter |p:pointer|, where p is the created instance. In this case, the block's result would become the function's returned value.

5.22 nil Class

The nil class is the class of nil value that is usually used as an invalid value. In a logical operation, the nil value is recognized as false.

5.23 number Class

The number class is a type of number values. A number literal would create a number instance.

5.23.1 Method

number.roundoff(threshold:number => 1e-10)

5.24 operator Class

The `operator` class provides measures to assign operators with a user-defined procedure.

5.24.1 Property

An `operator` instance has the following properties:

Property	Type	R/W	Explanation
`symbol`	`symbol`	R	Operator symbol.

5.24.2 Constructor

operator(symbol:symbol):map {block?}

Creates an `operator` instance that is associated with the specified symbol.

If `block` is specified, it would be evaluated with a block parameter `|op:operator|`, where `op` is the created instance. In this case, the block's result would become the function's returned value.

Below is an example to create an `operator` instance that is associated with the plus symbol.

```
op = operator(`+)
```

5.24.3 Method

operator#assign(type_l:expr, type_r?:expr):map:void {block}

Associates the `operator` instance with a procedure described in `block` that takes values as a block parameter and returns its operation result.

Some `operator` instances have two forms of expression: unary and binary. This method assignes the procedure to one of them according to how it takes its arguments as below:

- `operator#assign(type:expr)` .. Assigns procedure to the unary form.
- `operator#assign(type_l:expr, type_r:expr)` .. Assignes procedure to the binary form.

They take different format of block parameters as below:

- `|value|` .. For unary form.
- `|value_l, value_r|` .. For binary form.

Below is an example to assign a procedure to a unary form of operator `-`.

```
operator(`-).assign(`string) = {|value|
    // any job
}
```

Below is an example to assign a procedure to a binary form of operator `-`.

```
operator(`-).assign(`string, `number) = {|value_l, value_r|
    // any job
}
```

`operator#entries(type?:symbol)`

Returns a list that contains type expressions that the operator can accept as its arguments.

The argument `type` takes a symbol `` `binary `` or `` `unary ``.

- If it's omitted or specified with `` `binary ``, the method would return a list of pairs of type expressions for its left element and right one.
- If it's specified with `` `unary ``, the method would return a list of type expressions for its single element.

5.25 palette Class

The `palette` instance has a set of `color` instance.

5.25.1 Constructor

`palette(type) {block?}`

Creates a `palette` instance.

If `block` is specified, it would be evaluated with a block parameter `|plt:palette|`, where `plt` is the created instance. In this case, the block's result would become the function's returned value.

This function can be called in the following two forms:

- `palette(n:number)` .. Creates an instance with the specified number of entries. All the entries are initialized with a color of black.
- `palette(type:symbol)` .. Creates an instance initialized with a pre-defined set of entries associated with the specified symbol.

In the second form, it can take one of the following symbols:

- `` `basic `` .. A palette with 16 basic colors that are: `color.black`, `color.maroon`, `color.green`, `color.olive`, `color.navy`, `color.purple`, `color.teal`, `color.gray`, `color.silver`, `color.red`, `color.lime`, `color.yellow`, `color.blue`, `color.fuchsia`, `color.aqua` and `color.white`.
- `` `win256 `` .. A palette with 256 colors defined by Windows.
- `` `websafe `` .. A palette with 216 colors that assure to be displayed correctly in any Web environments. It actually has 256 entries though the last 40 entries are initialized with black.

5.25.2 Method

`palette#each() {block?}`

Creates an iterator that iterates each element in the palette.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`palette#nearest(color:color):map:[index]`

Returns a `color` instance in the palette that is the nearest with the specified color.

If the attribute `:index` is specified, it would return an index of the nearst entry instead of its `color` instance.

`palette#shrink():reduce:[align]`

Shrinks the size of the palette to a number powered by two that is enough to contain unique entries. The ordef of existing entries will be kept intact.

`palette#updateby(image_or_palette):reduce:[align,shrink]`

Updates palette entries according to color data in an image or a palette.

The order of existing entries will be kept intact. If attribute shrink is specified, the whole size will be shrinked to a number powered by two that is enough to contain unique entries.

5.26 pointer Class

The `pointer` class provides measures to read and write content in a `binary` and `memory` instance.

5.26.1 Property

A `pointer` instance has the following properties:

Property	Type	R/W	Explanation
`offset`	`number`	R/W	The current offset.
`size`	`number`	R	Returns the size of data accessible from the current offset.
`size@all`	`number`	R	Returns the entire size of the target binary or memory. This equals to `p.offset + p.size` where 'p' is a 'pointer' instance.
`target`	`any`	R	An instance that is associated with the pointer. Currently, this can be an instance of 'binary' or 'memory'.

5.26.2 Constructor

`pointer(org:pointer):map {block?}`

Creates a `pointer` instance that is cloned from the given instance `org`. You can use this to cast a `binary` and `memory` instance to the `pointer`.

If `block` is specified, it would be evaluated with a block parameter `|ptr:pointer|`, where `ptr` is the created instance. In this case, the block's result would become the function's returned value.

5.26.3 Method

`pointer#copyfrom(src:pointer, bytes?:number):map:reduce`

Copies data from `src` to the target pointer.

If the argument `bytes` is specified, it would limit the size of data to be copied. Otherwise, all the data pointerd by `src` is to be copied.

This method returns a reference to the target instance itself.

`pointer#copyto(dst:pointer, bytes?:number):map:reduce`

Copies data from the target pointer to `dst`.

If the argument `bytes` is specified, it would limit the size of data to be copied. Otherwise, all the data pointerd by the target instance is to be copied.

This method returns a reference to the target instance itself.

`pointer#decode(codec:codec, bytes?:number) {block?}`

Decodes the content of the `pointer` as a sequence of string characters using `codec` and returns the result in `string`.

If the argument `bytes` is specified, it would limit the size of data to be decoded. Otherwise, all the data pointerd by the target instance is to be decoded.

If `block` is specified, it would be evaluated with a block parameter `|str:string|`, where `str` is the created instance. In this case, the block's result would become the function's returned value.

`pointer#dump(stream?:stream:w, bytes?:number):reduce:[upper]`

Prints a hexadecimal dump from the content of the `pointer` to the standard output. If the argument `stream` is specified, the result would be output to the stream.

If the argument `bytes` is specified, it would limit the size of data to be dumped. Otherwise, all the data pointerd by the target instance is to be dumped.

In default, hexadecimal digit are printed with lower-case characters. Specifying an attribute `:upper` would output them with upper-case characters instead.

Example:

```
>>> b'A quick brown fox jumps over the lazy dog.'.p.dump():upper
41 20 71 75 69 63 6B 20 62 72 6F 77 6E 20 66 6F  A quick brown fo
78 20 6A 75 6D 70 73 20 6F 76 65 72 20 74 68 65  x jumps over the
20 6C 61 7A 79 20 64 6F 67 2E                     lazy dog.
```

`pointer#encodeuri(bytes?:number) {block?}`

Returns a string in which non-URIC characters are converted to percent-encoded string.

For example, `b'"Hello"'.p.encodeuri()` would return `'%22Hello%22'`.

If `block` is specified, it would be evaluated with a block parameter `|str:string|`, where `str` is the created instance. In this case, the block's result would become the function's returned value.

`pointer#each@int8():[be] {block?}`

Creates an iterator that extracts numbers in size of int8 from the current pointer position.

In default, it assumes the byte seqeuces are ordered in little-endian. You can specify `:be` attribute to extract them in big-endian order.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`pointer#each@uint8():[be] {block?}`

Creates an iterator that extracts numbers in size of uint8 from the current pointer position.

In default, it assumes the byte seqeuces are ordered in little-endian. You can specify `:be` attribute to extract them in big-endian order.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.

- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`pointer#each@int16():[be] {block?}`

Creates an iterator that extracts numbers in size of int16 from the current pointer position.

In default, it assumes the byte seqeuces are ordered in little-endian. You can specify `:be` attribute to extract them in big-endian order.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`pointer#each@uint16():[be] {block?}`

Creates an iterator that extracts numbers in size of uint16 from the current pointer position.

In default, it assumes the byte seqeuces are ordered in little-endian. You can specify `:be` attribute to extract them in big-endian order.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`pointer#each@int32():[be] {block?}`

Creates an iterator that extracts numbers in size of int32 from the current pointer position.

In default, it assumes the byte seqeuces are ordered in little-endian. You can specify `:be` attribute to extract them in big-endian order.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`pointer#each@uint32():[be] {block?}`

Creates an iterator that extracts numbers in size of uint32 from the current pointer position.

In default, it assumes the byte seqeuces are ordered in little-endian. You can specify `:be` attribute to extract them in big-endian order.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`pointer#each@int64():[be] {block?}`

Creates an iterator that extracts numbers in size of int64 from the current pointer position.

In default, it assumes the byte seqeuces are ordered in little-endian. You can specify `:be` attribute to extract them in big-endian order.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`pointer#each@uint64():[be] {block?}`

Creates an iterator that extracts numbers in size of uint64 from the current pointer position.

In default, it assumes the byte seqeuces are ordered in little-endian. You can specify `:be` attribute to extract them in big-endian order.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`pointer#each@float():[be] {block?}`

Creates an iterator that extracts numbers in size of float from the current pointer position.

In default, it assumes the byte seqeuces are ordered in little-endian. You can specify `:be` attribute to extract them in big-endian order.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`pointer#each@double():[be] {block?}`

Creates an iterator that extracts numbers in size of double from the current pointer position.

In default, it assumes the byte seqeuces are ordered in little-endian. You can specify `:be` attribute to extract them in big-endian order.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`pointer#forward(distance:number):reduce`

Put the pointer offset forward by `distance`. If a negative number is specified for the argument, the offset would be put backward.

An error would occur when the pointer's offset becomes a negative value while it would be no error when the offset exceeds the target maximum range.

This method returns a reference to the target instance itself.

`pointer#get@int8():[be,nil,stay] {block?}`

Returns an extracted number in size of int8 from the current pointer position.

In default, it assumes the byte seqeuces are ordered in little-endian. You can specify `:be` attribute to extract them in big-endian order.

If `block` is specified, it would be evaluated with a block parameter `|n:number|`, where `n` is the created instance. In this case, the block's result would become the function's returned value.

`pointer#get@uint8():[be,nil,stay] {block?}`

Returns an extracted number in size of uint8 from the current pointer position.

In default, it assumes the byte seqeuces are ordered in little-endian. You can specify `:be` attribute to extract them in big-endian order.

If `block` is specified, it would be evaluated with a block parameter `|n:number|`, where `n` is the created instance. In this case, the block's result would become the function's returned value.

`pointer#get@int16():[be,nil,stay] {block?}`

Returns an extracted number in size of int16 from the current pointer position.

In default, it assumes the byte seqeuces are ordered in little-endian. You can specify `:be` attribute to extract them in big-endian order.

If `block` is specified, it would be evaluated with a block parameter `|n:number|`, where `n` is the created instance. In this case, the block's result would become the function's returned value.

`pointer#get@uint16():[be,nil,stay] {block?}`

Returns an extracted number in size of uint16 from the current pointer position.

In default, it assumes the byte seqeuces are ordered in little-endian. You can specify `:be` attribute to extract them in big-endian order.

If `block` is specified, it would be evaluated with a block parameter `|n:number|`, where `n` is the created instance. In this case, the block's result would become the function's returned value.

`pointer#get@int32():[be,nil,stay] {block?}`

Returns an extracted number in size of int32 from the current pointer position.

In default, it assumes the byte seqeuces are ordered in little-endian. You can specify `:be` attribute to extract them in big-endian order.

If `block` is specified, it would be evaluated with a block parameter `|n:number|`, where `n` is the created instance. In this case, the block's result would become the function's returned value.

`pointer#get@uint32():[be,nil,stay] {block?}`

Returns an extracted number in size of uint32 from the current pointer position.

In default, it assumes the byte seqeuces are ordered in little-endian. You can specify `:be` attribute to extract them in big-endian order.

If `block` is specified, it would be evaluated with a block parameter `|n:number|`, where `n` is the created instance. In this case, the block's result would become the function's returned value.

`pointer#get@int64():[be,nil,stay] {block?}`

Returns an extracted number in size of int64 from the current pointer position.

In default, it assumes the byte seqeuces are ordered in little-endian. You can specify `:be` attribute to extract them in big-endian order.

If `block` is specified, it would be evaluated with a block parameter `|n:number|`, where `n` is the created instance. In this case, the block's result would become the function's returned value.

`pointer#get@uint64():[be,nil,stay] {block?}`

Returns an extracted number in size of uint64 from the current pointer position.

In default, it assumes the byte seqeuces are ordered in little-endian. You can specify `:be` attribute to extract them in big-endian order.

If `block` is specified, it would be evaluated with a block parameter `|n:number|`, where `n` is the created instance. In this case, the block's result would become the function's returned value.

`pointer#get@float():[be,nil,stay] {block?}`

Returns an extracted number in size of float from the current pointer position.

In default, it assumes the byte seqeuces are ordered in little-endian. You can specify `:be` attribute to extract them in big-endian order.

If `block` is specified, it would be evaluated with a block parameter `|n:number|`, where `n` is the created instance. In this case, the block's result would become the function's returned value.

`pointer#get@double():[be,nil,stay] {block?}`

Returns an extracted number in size of double from the current pointer position.

In default, it assumes the byte seqeuces are ordered in little-endian. You can specify `:be` attribute to extract them in big-endian order.

If `block` is specified, it would be evaluated with a block parameter `|n:number|`, where `n` is the created instance. In this case, the block's result would become the function's returned value.

`pointer#head():reduce`

Moves the pointer position to the beginning.

This method returns a reference to the target instance itself.

`pointer#hex(bytes?:number):[carray,cstr,upper] {block?}`

Converts the binary data into a hexadecimal string.

In default, the result string is a sequence of joined hexadecimal values without any space. You can specify the following attribute to change the format:

- `:cstr` .. Format of C string.
- `:carray` .. Format of C array.

Alphabet characters are described in lower characters unless the attribute `:upper` is specified.

If `block` is specified, it would be evaluated with a block parameter `|str:string|`, where `str` is the created instance. In this case, the block's result would become the function's returned value.

Example:

Code	Result
`b'\x01\x23\xab\xcd'.p.hex()`	`'0123abcd'`
`b'\x01\x23\xab\xcd'.p.hex():upper`	`'0123ABCD'`
`b'\x01\x23\xab\xcd'.p.hex():cstr`	`'\\x01\\x23\\xab\\xcd'`
`b'\x01\x23\xab\xcd'.p.hex():carray`	`'0x01, 0x23, 0xab, 0xcd'`

`pointer#pack(format:string, values+):reduce:[stay]`

Packs `values` in the argument list according to specifiers in the `format` into a binary and adds it to where the pointer points. The pointer offset is automatically incremented by the added length unless `:stay` attribute is specified.

This method returns a reference to the target instance itself.

A specifier has a format of ”`nX`” where `X` is a format character that represents a packing format and `n` is a number of packing size. The number can be omitted, and it would be treated as `1` in that case.

Following format characters would take a `number` value from the argument list and pack them into a binary sequence.

- `b` .. One-byte signed number.
- `B` .. One-byte unsigned number.
- `h` .. Two-byte signed number.
- `H` .. Two-byte unsigned number.
- `i` .. Four-byte signed number.
- `I` .. Four-byte unsigned number.
- `l` .. Four-byte signed number.
- `L` .. Four-byte unsigned number.
- `q` .. Eight-byte signed number.
- `Q` .. Eight-byte unsigned number.
- `f` .. Float-typed number occupying four bytes.
- `d` .. Double-typed number occupying eight bytes.

As for them, the packing size `n` means the number of values to be packed.

Following format characters would take a `string` value from the argument list and pack them into a binary sequence.

- `s` .. Packs a sequence of UTF-8 codes in the string. The packing size `n` means the size of the room in bytes where the character codes are to be packed. Only the sequence within the allocated room would be packed. If the string length is smaller than the room, the lacking part would be filled with zero.
- `c` .. Picks the first byte of the string and packs it as a one-byte unsigned number. The packing size `n` means the number of values to be packed.

Following format character would take no value from the argument list.

- `x` .. Fills the binary with zero. The packing size `n` means the size of the room in bytes to be filled with zero.

The default byte-order for numbers of two-byte, four-byte and eight-byte depends on the system the interpreter is currently running. You can change it by the following specifiers:

- `@` .. System-dependent order.
- `=` .. System-dependent order.
- `<` .. Little endian
- `>` .. Big endian
- `!` .. Big endian

You can specify an asterisk character "*" for the number of packing size that picks that number from the argument list.

You can specify encoding name embraced with "{" and "}" in the format to change coding character set from UTF-8 while packing a string with format character "`s`".

`pointer#put@int8(n:number):map:reduce:[be,stay]`

Stores the specified number to the current pointer position in size of int8.

In default, it stores the byte sequences in the order of little-endian. You can specify `:be` sttribute to store them in big-endian order.

This method returns a reference to the target instance itself.

`pointer#put@uint8(n:number):map:reduce:[be,stay]`

Stores the specified number to the current pointer position in size of uint8.

In default, it stores the byte sequences in the order of little-endian. You can specify `:be` sttribute to store them in big-endian order.

This method returns a reference to the target instance itself.

`pointer#put@int16(n:number):map:reduce:[be,stay]`

Stores the specified number to the current pointer position in size of int16.

In default, it stores the byte sequences in the order of little-endian. You can specify `:be` sttribute to store them in big-endian order.

This method returns a reference to the target instance itself.

`pointer#put@uint16(n:number):map:reduce:[be,stay]`

Stores the specified number to the current pointer position in size of uint16.

In default, it stores the byte sequences in the order of little-endian. You can specify `:be` sttribute to store them in big-endian order.

This method returns a reference to the target instance itself.

`pointer#put@int32(n:number):map:reduce:[be,stay]`

Stores the specified number to the current pointer position in size of int32.

In default, it stores the byte sequences in the order of little-endian. You can specify `:be` sttribute to store them in big-endian order.

This method returns a reference to the target instance itself.

`pointer#put@uint32(n:number):map:reduce:[be,stay]`

Stores the specified number to the current pointer position in size of uint32.

In default, it stores the byte sequences in the order of little-endian. You can specify `:be` sttribute to store them in big-endian order.

This method returns a reference to the target instance itself.

`pointer#put@int64(n:number):map:reduce:[be,stay]`

Stores the specified number to the current pointer position in size of int64.

In default, it stores the byte sequences in the order of little-endian. You can specify `:be` sttribute to store them in big-endian order.

This method returns a reference to the target instance itself.

`pointer#put@uint64(n:number):map:reduce:[be,stay]`

Stores the specified number to the current pointer position in size of uint64.

In default, it stores the byte sequences in the order of little-endian. You can specify `:be` sttribute to store them in big-endian order.

This method returns a reference to the target instance itself.

`pointer#put@float(n:number):map:reduce:[be,stay]`

Stores the specified number to the current pointer position in size of float.

In default, it stores the byte sequences in the order of little-endian. You can specify `:be` sttribute to store them in big-endian order.

This method returns a reference to the target instance itself.

`pointer#put@double(n:number):map:reduce:[be,stay]`

Stores the specified number to the current pointer position in size of double.

In default, it stores the byte sequences in the order of little-endian. You can specify `:be` sttribute to store them in big-endian order.

This method returns a reference to the target instance itself.

`pointer#reader() {block?}`

Creates a `stream` instance with which you can read data from the memory pointerd by the pointer. If `block` is specified, it would be evaluated with a block parameter `|s:stream|`, where `s` is the created instance. In this case, the block's result would become the function's returned value.

`pointer#seek(offset:number):reduce`

Moves the pointer position to the specified `offset`.

This method returns a reference to the target instance itself.

`pointer#tail():reduce`

Moves the pointer position to the end.

This method returns a reference to the target instance itself.

`pointer#unpack(format:string, values*:number):[nil,stay] {block?}`

Extracts values from data sequence pointed by the `pointer` instance according to specifiers in the `format` and returns a list containing the values.

A specifier has a format of "`nX`" where `X` is a format character that represents a packing format and `n` is a number of packing size. The number can be omitted, and it would be treated as `1` in that case.

Following format characters would extract an integer or float value of specified size from the binary and returns a `number` value.

- `b` .. One-byte signed number.
- `B` .. One-byte unsigned number.
- `h` .. Two-byte signed number.
- `H` .. Two-byte unsigned number.
- `i` .. Four-byte signed number.

- `I` .. Four-byte unsigned number.
- `l` .. Four-byte signed number.
- `L` .. Four-byte unsigned number.
- `q` .. Eight-byte signed number.
- `Q` .. Eight-byte unsigned number.
- `f` .. Float-typed number occupying four bytes.
- `d` .. Double-typed number occupying eight bytes.

As for them, the packing size `n` means the number of values to be extracted.

Following format characters would extract a string sequence from the binary and returns a `string` value.

- `s` .. Extracts a sequence of UTF-8 codes and returns `string` instance containing it. The unpacking size `n` means the size of the room in bytes where the character codes are to be unpacked.
- `c` .. Extracts a one-byte unsigned number and returns a `string` instance containing it. The unpacking size `n` means the number of values to be extracted.

Following format character would not return any value.

- `x` .. Advances the address by one byte. If the unpacking size `n` is specifies, it would advance the address by `n` bytes.

The default byte-order for numbers of two-byte, four-byte and eight-byte depends on the system the interpreter is currently running. You can change it by the following specifiers:

- `@` .. System-dependent order.
- `=` .. System-dependent order.
- `<` .. Little endian
- `>` .. Big endian
- `!` .. Big endian

You can specify an asterisk character "*" for the number of unpacking size that picks that number from the argument list.

You can specify encoding name embraced with "{" and "}" in the format to change coding character set from UTF-8 while extracting a string with format character "`s`".

An error occurs if the binary size is smaller than the format reqeusts. If the attribute `:nil` is specified, `nil` value would be returned for such a case.

If `block` is specified, it would be evaluated with a block parameter `|list:list|`, where `list` is the created instance. In this case, the block's result would become the function's returned value.

`pointer#unpacks(format:string, values*:number):map {block?}`

Returns an iterator that extracts values from data pointed by the `pointer` instance according to specifiers in `format`.

For detailed information about specifiers, see the help of `pointer#unpack()`.

If `block` is specified, it would be evaluated with a block parameter `|iter:iterator|`, where `iter` is the created instance. In this case, the block's result would become the function's returned value.

`pointer#writer() {block?}`

Creates a `stream` instance with which you can append data to the memory pointed by the pointer. If `block` is specified, it would be evaluated with a block parameter `|s:stream|`, where `s` is the created instance. In this case, the block's result would become the function's returned value.

5.26.4 Cast Operation

A function that expects a `pointer` instance in its argument can also take a value of `binary` and `memory`.

With the above casting feature, you can call a function `f(p:pointer)` that takes a `pointer` instance in its argument as below:

- `b = b'\x01\x23\x45\x67\x89\xab', f(b)`
- `m = memory(32), f(m)`

5.27 rational Class

The `rational` class provides measures to handle rational numbers.

You can create a `rational` instance with following ways:

- Use `rational()` function.
- Append `r` suffix after a number literal.

Below are examples to realize a common fraction two-thirds:

```
rational(2, 3)
2r / 3
2 / 3r
```

5.27.1 Constructor

`rational(numer:number, denom?:number):map {block?}`

Creates a rational value from given numerator `numer` and denominator `denom`.

If the argument `denom` is omitted, one is set as its denominator.

If `block` is specified, it would be evaluated with a block parameter `|r:rational|`, where `r` is the created instance. In this case, the block's result would become the function's returned value.

5.27.2 Method

`rational.reduce()`

Reduces the rational number by dividing its numerator and denominator by their GCD.

5.28 semaphore Class

5.28.1 Constructor

`semaphore()`

5.28.2 Method

`semaphore#release()`

Releases the owership of the semaphore that is grabbed by semaphore#wait().

`semaphore#session() {block}`

Forms a critical session by grabbing the semaphore's ownership, executing the block and releasing that ownership. It internally proccesses the same job as semaphore#wait() and semaphore#release() before and after the block execution

`semaphore#wait()`

Watis for the semaphore being released by other threads, and ghen grabs that ownership.

5.29 stream Class

The `stream` class provides methods to read and write data through a stream, an abstract structure to handle a byte sequence. It also provides information of the stream such as the pathname and the creation date and time.

You can specify a proper `codec` when creating the `stream` instance, which is used to decode/encode character codes that appear in the stream. Features of `codec` would affect on functions and methods that handle text data like follows:

- Decode
  - readlines()
  - stream#readchar()
  - stream#readline()
  - stream#readlines()
  - stream#readtext()
- Encode
  - operator `<<`
  - stream#print()
  - stream#printf()
  - stream#println()

5.29.1 Property

A `stream` instance has the following properties:

Property	Type	R/W	Explanation
`stat`	`object`	R	Status of the stream.
`name`	`string`	R	Name of the stream.
`identifier`	`string`	R	Identifier of the stream.
`readable`	`boolean`	R	Indicates whether the stream is readable.
`writable`	`boolean`	R	Indicates whether the stream is writable.
`codec`	`codec`	R	‘codec‘ instance associated with the stream.

5.29.2 Operator

You can use the operator ”`<<`” to output a content of a value to a `stream`. It comes like ”`stream << obj`” where `obj` is converted to a string before output to the stream.

```
sys.stdout << 'Hello World.'
```

Since the operator returns the `stream` instance specified on the left as its result, you can chain multiple operations as below:

```
sys.stdout << 'First' << 'Second'
```

5.29.3 Cast Operation

A function that expects a `stream` instance in its argument can also take a value of `string` and `binary` as below:

- `string` .. Recognized the `string` as a path name from which `stream` instance is created.
- `binary` .. Creates a `stream` instance that reads or modifies the content of the specified `binary` data. If the `binary` data is a constant one, which might be created from a binary literal such as `b'\x00\x12\x34\x56'`, the stream is created with read-only attribute.

Using the above casting feature, you can call a function `f(stream:stream)` that takes a `stream` instance in its argument as below:

- `f(stream('foo.txt'))` .. The most explicit way.
- `f('foo.txt')` .. Implicit casting from `string` to `stream`.
- `f(b'\x00\x12\x34\x56')` .. Implicit casting from `binary` to `stream` that reads the content.

5.29.4 Constructor

`stream(pathname:string, mode?:string, codec?:codec):map {block?}`

Creates a `stream` instance from the specified `pathname`.

The argument `mode` takes one of the strings that specifies what access should be allowed with the stream. If omitted, the stream would be opened with read mode.

- `'r'` .. read

- 'w' .. write
- 'a' .. append

The argument `codec` specifies a name of the character codec that converts between the stream's character code and UTF-8, which is a code used in the iterpreter's internal process.

If `block` is specified, it would be evaluated with a block parameter `|s:stream|`, where `s` is the created instance. In this case, the block's result would become the function's returned value.

You can also call `open()` function that is just an alias of `stream()` to create a `stream` instance.

5.29.5 Utility Function

`readlines(stream?:stream:r):[chop] {block?}`

Creates an iterator that reads text from the specified stream line by line.

If attribute `:chop` is specified, it eliminates an end-of-line character that appears at the end of each line.

This function decodes character codes in the stream using `codec` instance that is specified when the `stream` instance is created.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

5.29.6 Method

`stream#addcr(flag?:boolean):reduce`

The codec's encoder in the stream has a feature to add a CR code (`0x0d`) before a LF code (`0x0a`) so that the lines are joined with CR-LF codes in the encoded result. This method enables or disables the feature.

- To enable it, call the method with the argument `flag` set to `true` or without any argument.
- To disable it, call the method with the argument `flag` set to `false`.

`stream#close():void`

Closes the stream.

`stream#compare(stream:stream:r):map`

Returns `true` if there's no difference between the binary sequences of the target stream instance and that of `stream` in the argument.

`stream.copy(src:stream:r, dst:stream:w, bytesunit:number => 65536):static:map:void:[finalize] {block?}`

Copies the content in `src` to the stream `dst`.

The copying is done by the following process:

1. Reads data from stream `src` into a buffer with the size specified by `bytesunit`.
2. If `block` is specified, it would be evaluated with a block parameter `|buff:binary|` where `buff` contains the read data. When the block's result is a `binary` instance, the content would be written to the stream `dst`. Otherwise, the read data would be written to stream `dst`.
3. If `block` is not specified, the read data would be written to stream `dst`.
4. Continues from step 1 to 3 until data from `src` runs out.

If the attribute `:finalize` is specified, some finalizing process will be applied at the end such as copying time stamp and attributes.

This has the same feature as `stream#copyfrom()` and `stream#copyto()`.

`stream#copyfrom(src:stream:r, bytesunit:number => 65536):map:reduce:[finalize] {block?}`

Copies the content in `src` to the target stream instance.

The copying is done by the following process:

1. Reads data from stream `src` into a buffer with the size specified by `bytesunit`.
2. If `block` is specified, it would be evaluated with a block parameter `|buff:binary|` where `buff` contains the read data. When the block's result is a `binary` instance, the content would be written to the stream `dst`. Otherwise, the read data would be written to stream `dst`.
3. If `block` is not specified, the read data would be written to stream `dst`.
4. Continues from step 1 to 3 until data from `src` runs out.

If the attribute `:finalize` is specified, some finalizing process will be applied at the end such as copying time stamp and attributes.

This has the same feature as `stream.copy()` and `stream#copyto()`.

`stream#copyto(dst:stream:w, bytesunit:number => 65536):map:reduce:[finalize] {block?}`

Copies the content in the target stream instance to stream `dst`.

The copying is done by the following process:

1. Reads data from stream `src` into a buffer with the size specified by `bytesunit`.
2. If `block` is specified, it would be evaluated with a block parameter `|buff:binary|` where `buff` contains the read data. When the block's result is a `binary` instance, the content would be written to the stream `dst`. Otherwise, the read data would be written to stream `dst`.

3. If `block` is not specified, the read data would be written to stream `dst`.
4. Continues from step 1 to 3 until data from `src` runs out.

If the attribute `:finalize` is specified, some finalizing process will be applied at the end such as copying time stamp and attributes.

This has the same feature as `stream.copy()` and `stream#copyfrom()`.

`stream#delcr(flag?:boolean):reduce`

The codec's decoder in the stream has a feature to delete a CR code (`0x0d`) before a LF code (`0x0a`) so that the lines are joined with LF code in the decoded result. This method enables or disables the feature.

- To enable it, call the method with the argument `flag` set to `true` or without any argument.
- To disable it, call the method with the argument `flag` set to `false`.

`stream#deserialize()`

`stream#flush():void`

Flushes cached data to the stream.

`stream#peek(bytes?:number)`

Reads specified length of data from the stream and returns a `binary` instance that contains it. This doesn't move the stream's current file position.

`stream#print(values*):map:void`

Prints out `values` to the `stream` instance after converting them to strings.

This function encodes character codes in the string using `codec` instance that is specified when the `stream` instance is created.

`stream#printf(format:string, values*):map:void`

Prints out `values` to the `stream` instance according to formatter specifiers in `format`.

Refer to the help of `printf()` function to see information about formatter specifiers.

This function encodes character codes in the string using `codec` instance that is specified when the `stream` instance is created.

`stream#println(values*):map:void`

Prints out `values` and an end-of-line character to the `stream` instanceafter converting them to strings.

This function encodes character codes in the string using `codec` instance that is specified when the `stream` instance is created.

`stream#read(bytes?:number) {block?}`

Reads specified length of data from the stream and returns a `binary` instance that contains it. If the argument `bytes` is omitted, all the data available from the stream would be read.

If `block` is specified, it would be evaluated with a block parameter `|buff:binary|`, where `buff` is the created instance. In this case, the block's result would become the function's returned value.

`stream#readchar() {block?}`

Reads one character from the stream and returns a `string` instance that contains it.

This method decodes character codes in the stream using `codec` instance that is specified when the `stream` instance is created.

If `block` is specified, it would be evaluated with a block parameter `|ch:string|`, where `ch` is the created instance. In this case, the block's result would become the function's returned value.

`stream#readline():[chop] {block?}`

Reads one line from the stream and returns a `string` instance that contains it.

If the attribute `:chop` is specified, it would remove the last new line character from the result. This method decodes character codes in the stream using `codec` instance that is specified when the `stream` instance is created.

If `block` is specified, it would be evaluated with a block parameter `|line:string|`, where `line` is the created instance. In this case, the block's result would become the function's returned value.

`stream#readlines(nlines?:number):[chop] {block?}`

Creates an iterator that reads text from the specified stream line by line.

The argument `nlines` specifies how many lines should be read from the stream. If omitted, it would read all the lines.

If attribute `:chop` is specified, it eliminates an end-of-line character that appears at the end of each line.

This method decodes character codes in the stream using `codec` instance that is specified when the `stream` instance is created.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`stream#readtext() {block?}`

Reads the whole data in the stream as a text sequence and returns a `string` instance that contains it. This method decodes character codes in the stream using `codec` instance that is specified when the `stream` instance is created.

If `block` is specified, it would be evaluated with a block parameter `|text:string|`, where `text` is the created instance. In this case, the block's result would become the function's returned value.

`stream#seek(offset:number, origin?:symbol):reduce`

Seeks the current file position to the offset specified by the argument `offset`.

The argument `origin` specifies the meaning of `offset` value as follows:

- `set‘ ... ‘offset‘ is an absolute offset from the begining of the stream.`
- `cur‘ ... ‘offset‘ is a relative offset from the current position.`

This method returns the target stream instance itself.

`stream#serialize(value):void`

`stream#setcodec(codec:codec:nil):reduce`

Sets `codec` instance to the target stream. If `nil` is specified for the argument, the current `codec` instance would be removed.

This method returns the target stream instance itself.

`stream#tell()`

Returns the current file position at which read/write operation works.

`stream#write(ptr:pointer, bytes?:number):reduce`

Writes binary data pointer by `ptr` to the stream. The argument `bytes` limits the number of data that is to be written to the stream.

5.30 string Class

The `string` class provides measures to operate on strings.

You can create a `string` instance by embracing a sequence of characters with a pair of single- or double-quotes.

```
'Hello World'

"Hello World"
```

If you need to declare a string that contains multiple lines, embrace it with a pair of sequences of three single- or double-quotes.

```
'''first line
second line
third line
'''
```

5.30.1 Suffix Management

When an string literal is suffixed by a character `$`, a handler registered by `string.translate()` function that is supposed to translate the string into other natural languages would be evaluated.

5.30.2 Method

`string#align(width:number, padding:string => ' '):map:[center,left,right] {block?}`

Align the string to the left, right or center within the specified `width` and returns the result.

The following attributes specify the alignment position:

- `:center` .. Aligns to the center. This is the default.
- `:left` .. Aligns to the left
- `:right` .. Aligns to the right

If the string width is narrower than the specified `width`, nothing would be done.

It uses a string specified by the argument `padding` to fill lacking spaces. If omitted, a white space is used for padding.

This method takes into account the character width based on the specification of East Asian Width. A kanji-character occupies two characters in width.

`string.binary() {block?}`

Converts the string into `binary` instance.

`string#capitalize() {block?}`

Returns a string that capitalizes the first character.

`string#chop(suffix*:string):[eol,icase] {block?}`

Returns a string that removes a last character.

If an attribute `:eol` is specified, only the end-of-line character shall be removed. In this case, if the end-of-line has a sequence of CR-LF, CR code shall be removed as well.

`string#decodeuri() {block?}`

Returns a string in which percent-encoded characters are decoded.

`string#each():map:[utf32,utf8] {block?}`

Creates an iterator generating strings of each character in the original one.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero.

In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`string#eachline(nlines?:number):[chop] {block?}`

Creates an iterator generating strings of each line in the original one.

In default, end-of-line characters are involved in the result. You can eliminates them by specifying `:chop` attribute.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`string#embed(dst?:stream:w):[lasteol,noindent]`

Evaluates a string that contains embedded scripts and renders the result to the specified stream.

If the stream is omitted, the function returns the rendered result as a string.

Calling this method is equivalent to calling a method `string#template()` to create a `template` instance on which a method `template#render()` is applied afterward.

`string.encode(codec:codec) {block?}`

Encodes the string with the given `codec` and return the result as a `binary`.

`string#encodeuri() {block?}`

Returns a string in which non-URIC characters are percent-encoded.

`string#endswith(suffix:string, endpos?:number):map:[icase,rest]`

Returns `true` if the string ends with suffix.

If attribute `:rest` is specified, it returns the rest part if the string ends with suffix, or `nil` otherewise. You can specify a bottom position for the matching by an argument `endpos`.

With an attribute `:icase`, character cases are ignored while matching.

`string#escapehtml():[quote] {block?}`

Converts some characters into HTML entity symbols. If attribute `:quote` is specified, a double-quotation character would be converted to an entity symbol "".

`string#find(sub:string, pos:number => 0):map:[icase,rev]`

Finds a sub string from the string and returns its position.

Number of position starts from zero. You can specify a position to start finding by an argument pos. It returns nil if finding fails.

With an attribute `:icase`, case of characters are ignored while finding.

When an attribute `:rev`, finding starts from tail of the string

`string#fold(len:number, step?:number):[neat] {block?}`

Creates an iterator that folds the source string by the specified length.

The argument `step` specifies the length of advancement for the next folding point. If omitted, it would be the same amount as `len`.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`string#foldw(width:number):[padding] {block?}`

Creates an iterator that folds the source string by the specified width.

This method takes into account the character width based on the specification of East Asian Width.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`string#format(values*):map`

Taking the string instance as a printf-styled formatter string, it converts `values` into a string depending on formatter specifiers in it.

`string#isempty()`

Returns `true` if the string is empty.

`string#left(len?:number):map`

Extracts the specified length of string from left of the source string.

If the argument is omitted, it would return whole the source string.

`string#len()`

Returns the length of the string in characters.

`string#lower()`

Converts upper-case to lower-case characters.

`string#mid(pos:number => 0, len?:number):map {block?}`

Extracts the specified length of string from the position `pos` and returns the result.

If an argument `len` is omitted, it returns a string from `pos` to the end. The number of an argument `pos` starts from zero.

Below are examples:

```
'Hello world'.mid(3, 2) // 'lo'
'Hello world'.mid(5)    // 'world'
```

`string.print(stream?:stream:w):void`

Prints out the string to the specified `stream`.

If the argument is omitted, it would print to the standard output.

`string.println(stream?:stream:w):void`

Prints out the string and a line-break to the specified `stream`.

If the argument is omitted, it would print to the standard output.

`string#reader() {block?}`

Returns a `stream` instance that reads the string content as a binary sequence.

If `block` is specified, it would be evaluated with a block parameter `|s:stream|`, where `s` is the created instance. In this case, the block's result would become the function's returned value.

`string#replace(match:string, sub:string, count?:number):map:[icase] {block?}`

Replaces sub strings that matches the string `match` with a string specified by `sub` and returns the result.

The argument `count` limits the maximum number of substitution. If omitted, there's no limit of the work.

With an attribute `:icase`, character cases are ignored while matching strings.

If `block` is specified, it would be evaluated with a block parameter `|result:string, replaced:boolean|`, where `result` is the result string and `replaced` indicates if there is any change between the result and its original string. In this case, the block's result would become the function's returned value.

`string#replaces(map[]:string, count?:number):map:[icase] {block?}`

Replaces string parts according to a list of pairs of a matching and a substituting string and returns the result.

The argument `map` is a `list` of match-substitution paris like `[match1, sub1, match2, sub2, ..]` with which a sub string that matches `match`*n* would be replaced with `sub`*n*.

The argument `count` limits the maximum number of substitution. If omitted, there's no limit of the work.

With an attribute `:icase`, character cases are ignored while matching strings.

If `block` is specified, it would be evaluated with a block parameter `|result:string, replaced:boolean|`, where `result` is the result string and `replaced` indicates if there is any change between the result and its original string. In this case, the block's result would become the function's returned value.

`string#right(len?:number):map {block?}`

Extracts the specified length of string from right of the source string.

If the argument is omitted, it would return whole the source string.

`string#split(sep?:string, count?:number):[icase] {block?}`

Creates an iterator generating sub strings extracted from the original one separated by a specified string `sep`. With an attribute `:icase`, character cases are ignored while finding the separator.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`string#startswith(prefix:string, pos:number => 0):map:[icase,rest]`

Returns `true` if the string starts with `prefix`.

If attribute `:rest` is specified, it returns the rest part if the string starts with prefix, or `nil` otherwise. You can specify a top position for the matching by an argument `pos`.

With an attribute `:icase`, character cases are ignored while matching.

`string#strip():[both,left,right] {block?}`

Returns a string that removes space characters on the left, the right or the both sides of the original string.

The following attributes would specify which side of spaces should be removed:

- `:both` .. Removes spaces on both sides. This is the default.
- `:left` .. Removes spaces on the left side.
- `:right` .. Removes spaces on the right side.

`string#template():[lasteol,noindent] {block?}`

Parses the content of the string as a text containing embedded scripts and returns a `template` instance.

`string#tosymbol() {block?}`

Convers the string into a symbol.

`string.translator():static:void {block}`

Register a procedure evaluated when a string literal appears with a suffix symbol "`$`", which is meant to translate the string into another language.

The procedure is described in `block` takes a block parameter `|str:string|` where `str` is the original string, and is expected to return a string translated from the original.

`string#unescapehtml() {block?}`

Converts escape sequences into readable characters.

`string#upper() {block?}`

Converts lower-case to upper-case characters.

`string#width()`

Returns the width of the string.

This method takes into account the character width based on the specification of East Asian Width. For example, a kanji-character of Japanese occupies two characters in width.

`string#zentohan() {block?}`

Converts zenkaku to hankaku characters.

5.31 suffixmgr Class

The `suffixmgr` class provides measures to access suffix managers that are responsible to handle suffix symbols appended to number or string literals.

Below is an example to register a suffix `X` that converts a string into upper case after being appended to a string literal:

```
suffixmgr(`string).assign(`X) {|body| body.upper()}
```

You can use that suffix like below:

```
'hello world'X
```

5.31.1 Constructor

`suffixmgr(type:symbol) {block?}`

Creates a reference to one of two suffix managers, number and string.

- The number suffix manager works with number literals.
- The string suffix manager works with string literals.

Specify the argument `type` with a symbol `` `number `` for a number suffix manager and `` `string `` for a string suffix manager.

5.31.2 Method

`suffixmgr#assign(suffix:symbol):void:[overwrite] {block}`

Assigns a procedure to a specified symbol in the suffix manager. The procedure is provided by the `block` that takes a block parameter `|value|` where `value` comes from the preceded literal.

An error occurs if the same suffix symbol has already been assigned. Specifying `:overwrite` attribute will forcibly overwrite an existing assignment.

5.32 symbol Class

5.32.1 Method

`symbol#eval(env?:environment)`

Evaluate a symbol object.

5.33 template Class

5.33.1 Cast Operation

A function that expects a `template` instance in its argument can also take a value of `stream` as below:

- `stream` .. Creates a `template` instance by parsing the content of the stream.

As a `stream` is capable of being casted from `string` and `binary`, such values can also be passed to the argument that expects `template`.

Using the above casting feature, you can call a function `f(tmpl:template)` that takes a `template` instance in its argument as below:

- `f(template(stream('foo.txt')))` .. The most explicit way.
- `f(stream('foo.txt'))` .. Implicit casting: from `stream` to `template`.

- `f(template('foo.txt'))` .. Implicit casting: from `string` to `stream`.
- `f('foo.txt')` .. Implicit casting: from `string` to `stream`, then from `stream` to `template`.

5.33.2 Constructor

`template(src?:stream:r):map:[lasteol,noindent] {block?}`

Creates a `template` instance.

If the stream `src` is specified, the instance would be initialized with the parsed result of the script-embedded text from the stream.

Following attributes would customize the parser's behavior:

- `:lasteol`
- `:noindent`

5.33.3 Method

`template#parse(str:string):void:[lasteol,noindent]`

Creates a `template` instance by parsing a script-embedded text in a string.

Following attributes would customize the parser's behavior:

- `:lasteol`
- `:noindent`

`template#read(src:stream:r):void:[lasteol,noindent]`

Creates a `template` instance by parsing a script-embedded text from a stream.

Following attributes would customize the parser's behavior:

- `:lasteol`
- `:noindent`

`template#render(dst?:stream:w)`

Renders stored content to the specified stream.

If the stream is omitted, the function returns the rendered result as a string.

5.33.4 Method Called by Template Directive

`template#block(symbol:symbol):void`

Creates a template block which content is supposed to be replaced by a derived template.

This method is called by template directive `${=block()}` during both the initialization and presentation phase of a template process.

- **Initialization:** Creates a template block from the specified block that is then registered in the current template with the specified symbol.

- **Presentation:** Evaluates a template block registered with the specified symbol.

Consider an example. Assume that a block associated with symbol `‘foo` is declared in a template file named `base.tmpl` as below:

[base.tmpl]

```
Block begins here.
${=block(‘foo)}
Content of base.
${end}
Block ends here.
```

This template renders the following result:

```
Block begins here.
Content of derived.
Block ends here.
```

Below is another template named `derived.tmpl` that devies from `base.tmpl` and overrides the block `‘foo`.

[derived.tmpl]

```
${=extends('base.tmpl')}

${=block(‘foo)}
Content of derived.
${end}
```

This template renders the following result:

```
Block begins here.
Content of derived.
Block ends here.
```

`template#call(symbol:symbol, args*)`

Calls a template macro that has been created by directive `${=define}`.

This method is called by template directive `${=call()}` during the presentation phase of a template process.

Below is an exemple to call a template macro:

```
${=call(‘show_person, 'Harry', 24)}
```

This method would return `nil` if a line-break character is rendered at last and would return a null string otherwise.

`template#define(symbol:symbol, ‘args*):void`

Creates a template macro from the specified block, which is supposed to be called by `${=call}` directive, and associates it with the specified symbol.

This method is called by template directive `${=define()}` during the initialization phase of a template process.

Below is an example to create a template macro:

```
${=define(`show_person, name:string, age:number)}
${name} is ${age} years old.
${end}
```

`template#embed(template:template)`

Renders the specified template at the current position.

This method is called by template directive `${=embed()}` during the presentation phase of a template process.

Below is an example to embed a template file named `foo.tmpl`.

```
${=embed('foo.tmpl')}
```

As the template rendered by this method runs in a different context from the current one, macros and blocks that it defines are not reflected to the current context.

This method would return `nil` if a line-break character is rendered at last and would return a null string otherwise.

`template#extends(template:template):void`

Declares the current template as a derived one from the specified template.

This method is called by template directive `${=extends()}` during the initialization phase of a template process.

The directive must appear in a template only once. An error occurs if the current template has already derived from another template.

Below is an example to declare the current template as one derived from `base.tmpl`.

```
${=extends('base.tmpl')}
```

`template#super(symbol:symbol):void`

Evaluates a template block registered with the specified symbol in a template from which the current template has derived.

This method is called by template directive `${=super()}` during the presentation phase of a template process. The directive is intended to be used within a directive `${=block()}`.

Consider an example. Assume that a block associated with symbol `` `foo `` is declared in a template named `base.tmpl` as below:

`[base.tmpl]`

```
Block begins here.
${=block(`foo)}
```

```
Content of base.
${end}
Block ends here.
```

This template renders the following result:

```
Block begins here.
Content of derived.
Block ends here.
```

Below is another template named `derived.tmpl` that devies from `base.tmpl` and overrides the block `‘foo`.

`[derived.tmpl]`

```
${=extends(’base.tmpl’)}

${=block(‘foo)}
${=super(‘foo)}
Content of derived.
${end}
```

This template renders the following result:

```
Block begins here.
Content of base.
Content of derived.
Block ends here.
```

5.34 timedelta Class

The `timedelta` instance provides a time delta information that works with `datetime` instance. You can shift time information of `datetime` by applying addition or subtraction of `timedelta` to it.

5.34.1 Property

A `timedelta` instance has the following properties:

Property	**Type**	**R/W**	**Explanation**
`days`	`number`	R/W	Offset of days.
`secs`	`number`	R/W	Offset of seconds.
`usec`	`number`	R/W	Offset of micro seconds.

5.34.2 Constructor

`timedelta(days:number => 0, secs:number => 0, usecs:number => 0):map {block?}`

Returns a timedelta instance with specified values. The instance actually holds properties of days, secs and usecs.

5.35 uri Class

The `uri` instance analyzes a URI string and returns each part in it such as the scheme and path. A generic URI has the following format:

```
scheme:[//[user:password@]host:port]][/]path[?query][#fragment]
```

5.35.1 Property

A `uri` instance has the following properties:

Property	Type	R/W	Explanation
scheme	string	R/W	Scheme part in the URI.
user	string	R/W	User part in the URI.
password	string	R/W	Password part in the URI.
host	string	R/W	Host part in the URI.
port	string	R/W	Port part in the URI.
urlpath	string	R/W	URL path part in the URI, which contains the path, query and fragment part.
misc	string	R/W	Misc part in the URI.

5.35.2 Constructor

`uri(str?:string):map {block?}`

Creates `uri` instance.

If the argument `str` is specified, it would be parsed as a URI which is stored in the instance.

If omitted, the instance would be initialized as an empty one.

5.35.3 Method

`uri#getfragment()`

Returns the fragment part contained in the URI path.

`uri#getpath()`

Returns the path part contained in the URI path.

`uri#getquery()`

Returns a `dict` instance that is made from the query part in the URI path.

`uri.parsequery(query:string):static:map`

This is a utility function to parse a query string and return a `dict` instance that contains key-value pairs for the query.

5.35.4 Cast Operation

A function that expects a `uri` instance in its argument can also take a value of `string` that is recognized as a URI string.

With the above casting feature, you can call a function `f(uri:uri)` that takes a `uri` instance in its argument as below:

- `f(uri('http://example.com'))` .. The most explicit way.
- `f('http://example.com')` .. Implicit casting: from `string` to `uri`.

5.36 vertex Class

The `vertex` class provides vertex information that consists of x, y, z and w values.

5.36.1 Property

An `vertex` instance has the following properties:

Property	Type	R/W	Explanation
`x`	`number`	R/W	
`y`	`number`	R/W	
`z`	`number`	R/W	
`w`	`number`	R/W	

5.36.2 Constructor

`vertex(x:number, y:number, z?:number):map {block?}`

Creates a `vertex` instance that has the given coordinates `x`, `y` and `z`. The argument `z` is optional and set to zero if omitted.

If `block` is specified, it would be evaluated with a block parameter `|v:vertex|`, where `v` is the created instance. In this case, the block's result would become the function's returned value.

5.36.3 Method

`vertex.cross (v1:vertex, v2:vertex):static:map {block?}`

Calculates cross product between `v1` and `v2` and returns the result as a `vertex` instance.

`vertex.dot(v1:vertex, v2:vertex):static:map {block?}`

Calculates dot product between `v1` and `v2` and returns the result as a `number` instance.

`vertex#list() {block?}`

Creates a `list` that contains coordinate values `[x, y, z]` of the target `vertex`.

If `block` is specified, it would be evaluated with a block parameter `|list:list|`, where `list` is the created instance. In this case, the block's result would become the function's returned value.

`vertex.normal(v1:vertex, v2:vertex, v3:vertex):static:map:[unit] {block?}`

Calculates a normal vector for a face that consists of three vertices given and returns it as a `vertex` instance.

In default, it returns a vector before being regulated to have a length of one. Specifying the attribute `:unit` would apply the calculation.

If `block` is specified, it would be evaluated with a block parameter `|v:vertex|`, where `v` is the created instance. In this case, the block's result would become the function's returned value.

`vertex#rotate@x(angle:number):[deg] {block?}`

Creates a `vertex` that is rotated from the target `vertex` around X-axis by the specified `angle` in radian. It would be rotated in a direction of tilting Y-axis toward Z-axis.

If the attribute `:deg` is specified, you can specify the `angle` in degree unit.

If `block` is specified, it would be evaluated with a block parameter `|v:vertex|`, where `v` is the created instance. In this case, the block's result would become the function's returned value.

`vertex#rotate@y(angle:number):[deg] {block?}`

Creates a `vertex` that is rotated from the target `vertex` around Y-axis by the specified `angle` in radian. It would be rotated in a direction of tilting Z-axis toward X-axis.

If the attribute `:deg` is specified, you can specify the `angle` in degree unit.

If `block` is specified, it would be evaluated with a block parameter `|v:vertex|`, where `v` is the created instance. In this case, the block's result would become the function's returned value.

`vertex#rotate@z(angle:number):[deg] {block?}`

Creates a `vertex` that is rotated from the target `vertex` around Z-axis by the specified `angle` in radian. It would be rotated in a direction of tilting X-axis toward Y-axis.

If the attribute `:deg` is specified, you can specify the `angle` in degree unit.

If `block` is specified, it would be evaluated with a block parameter `|v:vertex|`, where `v` is the created instance. In this case, the block's result would become the function's returned value.

`vertex#translate(tx:number, ty:number, tz?:number) {block?}`

Creates a `vertex` that is translated from the target `vertex` by the specified offsets `tx`, `ty` and `tz`.

If `block` is specified, it would be evaluated with a block parameter `|v:vertex|`, where `v` is the created instance. In this case, the block's result would become the function's returned value.

Chapter 6

argopt Module

The `argopt` module provides measure to parse option strings in an argument list given through the command line.

Below is an example:

```
import(argopt)

argopt.Parser {|p|
    p.addParam('text', 't')
    p.addFlag('test')
    p.addFlag('bold', 'b')
    try {
        [cfg, argv] = p.parse(sys.argv)
    } catch {|e|
        println(e.text)
        sys.exit(1)
    }
}
```

6.1 argopt.Parser Class

6.1.1 Constructor

`argopt.Parser() {block?}`

Create an `argopt.Parser` instance.

6.1.2 Method

`argopt.Parser#parse(argv[]:string)`

Parses an argument list which is usually the value of `sys.argv` given by `sys` module.

It returns the result in a format `[cfg, argv]` where `cfg` is a `dict` instance containing parameter values and `argv` a list of arguments that have not been parsed as options.

`argopt.Parser#addParam(longName:string, shortName?:string, help?:string, helpValue?:string, defValue?:stri`

Adds an option that comes with a value like `--foo=value` where `foo` is a long name for the option.

The argument `longName` specifies a long option name that follows after two hyphens in the command line. This name is also used as a key when you look for a value in the dictionary `cfg` returned by `argopt.Parser#parse()`.

The argument `shortName` specifies a short option name that usually consists of a single character. If it exists, you can specify the option by a character followed by one hyphen like `-f value` where `f` is the short name.

The argument `help` and `helpValue` are used in a option parameter help created by `argopt.Parse#formatHelp()`. The string for `help` specifies a help text for the option while `helpValue` is a string printed after the equal character in the option presentation. If the argument `helpValue` is not specified, a string `X` is printed instead.

The argument `defValue` specifies a default value that would be used when the option is not specified in the command line.

`argopt.Parser#addFlag(longName:string, shortName?:string, help?:string)`

Adds an option that represents a boolean state. It comes like `--foo` where `foo` is a long name for the option.

The argument `longName` specifies a long option name that follows after two hyphens in the command line. This name is also used as a key when you look for a value in the dictionary `cfg` returned by `argopt.Parser#parse()`.

The argument `shortName` specifies a short option name that usually consists of a single character. If it exists, you can specify the option by a character followed by one hyphen like `-f` where `f` is the short name.

The argument `help` is used in a option parameter help created by `argopt.Parse#formatHelp()`. The string for `help` specifies a help text for the option.

`argopt.Parser#formatHelp(longNameFlag:boolean => true, shortNameFlag:boolean => true):[linefeed]`

Creates an iterator of strings that contain help text for each option.

If the argument `longNameFlag` is `true`, the help text would contain long names.

If the argument `shortNameFlag` is `true`, the help text would contain short names.

In default, each string doesn't contain a line feed at the end. To add a line feed, specify an attribute `:linefeed`.

Below is an example of showing help:

```
argopt.Parser {|p|
    p.addParam('text', 't', 'text value', 'txt')
    p.addFlag('flag1', 'f', 'flag option #1')
    p.addFlag('flag2', 'g', 'flag option #2')
    println(p.formatHelp())
}
```

The result comes as below:

```
-t, --text=txt  text value
-f, --flag1     flag option #1
-g, --flag2     flag option #2
```

Chapter 7

base64 Module

The `base64` module provides measures to decode/encode data formatted in base64 format.

To decode a stream that is formatted in base64, use one of the following functions:

- `base64.decode()` .. Reads base64 sequence from the given stream and returns a decoded data as `binary`. This is convenient when the data size is expected to be small.
- `base64.reader()` .. Creates a stream that decodes base64 sequence from the given stream. `stream#reader@base64()` method is another form of this function. You should use this way if the data size is expected to be large.

To encode a data into base64 format, use one of the following functions:

- `base64.encode()` .. Encodes the stream from the given stream and returns a encoded data as `binary`. This is convenient when the data size is expected to be small.
- `base64.writer()` .. Creates a stream that encodes data from `write()` method into the given stream. `stream#writer@base64()` method is another form of this function. You should use this way if the data size is expected to be large.

7.1 Module Function

`base64.decode(stream:stream:r) {block?}`

Reads text stream that is formatted in base64 and returns the decoded result in binary.

If `block` is specified, it would be evaluated with a block parameter `|data:binary|`, where `data` is the created instance. In this case, the block's result would become the function's returned value.

`base64.encode(stream:stream:r, linelen:number:nil => 76) {block?}`

Encodes content of the stream into base64 format and returns the result in binary.

If `block` is specified, it would be evaluated with a block parameter `|data:binary|`, where `data` is the created instance. In this case, the block's result would become the function's returned value.

`base64.reader(stream:stream:r) {block?}`

Creates a stream instance that reads data formatted in base64 from `stream`.

If `block` is specified, it would be evaluated with a block parameter `|s:stream|`, where `s` is the created instance. In this case, the block's result would become the function's returned value.

`base64.writer(stream:stream:w, linelen:number:nil => 76) {block?}`

Creates a stream instance that encodes data to base64 format and writes it to the `stream`.

The number of characters per line is specified by an argument `linelen`. If omitted, that is 76.

If `block` is specified, it would be evaluated with a block parameter `|s:stream|`, where `s` is the created instance. In this case, the block's result would become the function's returned value.

7.2 Extension to stream Class

This module extends the `stream` class with methods described here.

`stream#reader@base64() {block?}`

Creates a stream instance that reads data formatted in base64 from the target stream instance.

If `block` is specified, it would be evaluated with a block parameter `|s:stream|`, where `s` is the created instance. In this case, the block's result would become the function's returned value.

`stream#writer@base64(linelen:number:nil => 76) {block?}`

Creates a stream instance that encodes data to base64 format and writes it to the target stream instance.

The number of characters per line is specified by an argument `linelen`. If omitted, that is 76.

If `block` is specified, it would be evaluated with a block parameter `|s:stream|`, where `s` is the created instance. In this case, the block's result would become the function's returned value.

Chapter 8

bmp Module

The `bmp` module provides measures to read/write image data in Microsoft BMP format. To utilize it, import the `bmp` module using `import` function.

Below is an example to read a BMP file:

```
import(bmp)
img = image('foo.bmp')
```

8.1 Exntension to Function's Capability

This module extends the capability of function `image()` and instance method `image#write()` so that they can read/write BMP files.

When function `image()` is provided with a stream that satisfies the following conditions, it would recognize the stream as a BMP file.

- The identifier of the stream ends with a suffix ".bmp".
- The stream data begins with a byte sequence "BM".

When instance method `image#write()` is provided with a stream that satisfies the following condition, it would write image data in BMP format.

- The identifier of the stream ends with a suffix ".bmp".

8.2 Extension to image Class

This module extends the `image` class with methods described here.

`image#read@bmp(stream:stream:r):reduce`

Reads an BMP image from a stream.

This method returns the reference to the target instance itself.

`image#write@bmp(stream:stream:w):reduce`

Writes a BMP image to a stream.

This method returns the reference to the target instance itself.

Chapter 9

bzip2 Module

The `bzip2` module provices measures to read/write BZIP2 files. To utilize it, import the `bzip2` module using `import` function.

Below is an example to read data from a BZIP2 file and write its uncompressed data to another file.

```
import(bzip2)
bzip2.reader('foo.dat.bz2').copyto('foo.dat')
```

Below is an example to read data from a file and write its compressed data to a BZIP2 file.

```
import(bzip2)
bzip2.writer('foo.dat.bz2').copyfrom('foo.dat')
```

9.1 Module Function

`bzip2.reader(stream:stream:r) {block?}`

Creates a stream instance that decompresses bzip2 data from the specified `stream` that has readable attribute.

If `block` is specified, it would be evaluated with a block parameter `|st:stream|`, where `st` is the created instance. In this case, the block's result would become the function's returned value.

`bzip2.writer(stream:stream:w, blockSize100k?:number) {block?}`

Creates a stream instance that compresses data into bzip2 format and writes it to the specified `stream` that has writable attribute.

The argument `blockSize100k` takes a number between 1 and 9 that specifies the block size to be used for compression. The actual block size is 100000 times of this value. Nine gives the best compression but takes most memory.

If `block` is specified, it would be evaluated with a block parameter `|st:stream|`, where `st` is the created instance. In this case, the block's result would become the function's returned value.

9.2 Extension to stream Class

This module extends the `stream` class with methods described here.

`stream#reader@bzip2() {block?}`

Creates a stream instance that decompresses bzip2 data from the specified `stream` that has readable attribute.

If `block` is specified, it would be evaluated with a block parameter `|st:stream|`, where `st` is the created instance. In this case, the block's result would become the function's returned value.

`stream#writer@bzip2(blockSize100k?:number) {block?}`

Creates a stream instance that compresses data into bzip2 format and writes it to the specified `stream` that has writable attribute.

The argument `blockSize100k` takes a number between 1 and 9 that specifies the block size to be used for compression. The actual block size is 100000 times of this value. Nine gives the best compression but takes most memory.

If `block` is specified, it would be evaluated with a block parameter `|st:stream|`, where `st` is the created instance. In this case, the block's result would become the function's returned value.

9.3 Thanks

This module uses libbzip2 which is distributed in the following site:

http://www.bzip.org/

Chapter 10

cairo Module

The `cairo` module provides methods to draw 2-D graphics using Cairo library. Official site of Cairo is http://cairographics.org/.

10.1 Drawing

10.1.1 cairo.context - The cairo drawing context

Functions

`cairo.context#status()`

Checks whether an error has previously occurred for this context.

`cairo.context#save():reduce {block?}`

Makes a copy of the current state of cr and saves it on an internal stack of saved states for cr. When `cairo.context#restore()` is called, cr will be restored to the saved state. Multiple calls to `cairo.context#save()` and `cairo.context#restore()` can be nested; each call to `cairo.context#restore()` restores the state from the matching paired `cairo.context#save()`.

It isn't necessary to clear all saved states before a cairo_t is freed. If the reference count of a cairo_t drops to zero in response to a call to `cairo.context#destroy()`, any saved states will be freed along with the `cairo_t`.

`cairo.context#restore():reduce`

Restores cr to the state saved by a preceding call to `cairo.context#save()` and removes that state from the stack of saved states.

`cairo.context#get_target()`

Gets the target surface for the cairo context as passed to `cairo.context` constructor.

`cairo.context#push_group():reduce`

Temporarily redirects drawing to an intermediate surface known as a group. The redirection lasts until the group is completed by a call to `cairo.context#pop_group()` or `cairo.context#pop_group_to_sourc` These calls provide the result of any drawing to the group as a pattern, (either as an explicit object, or set as the source pattern).

This group functionality can be convenient for performing intermediate compositing. One common use of a group is to render objects as opaque within the group, (so that they occlude each other), and then blend the result with translucence onto the destination.

Groups can be nested arbitrarily deep by making balanced calls to `cairo.context#push_group()`/`cairo.context#p`
Each call pushes/pops the new target group onto/from a stack.

The `cairo.context#push_group()` function calls cairo_save() so that any changes to the graphics state will not be visible outside the group, (the pop_group functions call cairo_restore()).

By default the intermediate group will have a content type of `cairo.CONTENT_COLOR_ALPHA`. Other content types can be chosen for the group by using `cairo.context#push_group_with_content()` instead.

As an example, here is how one might fill and stroke a path with translucence, but without any portion of the fill being visible under the stroke:

`cairo.context#push_group_with_content(content:number):reduce`

Temporarily redirects drawing to an intermediate surface known as a group. The redirection lasts until the group is completed by a call to `cairo.context#pop_group()` or `cairo.context#pop_group_to_sourc`
These calls provide the result of any drawing to the group as a pattern, (either as an explicit object, or set as the source pattern).

The group will have a content type of content. The ability to control this content type is the only distinction between this function and `cairo.context#push_group()` which you should see for a more detailed description of group rendering.

`cairo.context#pop_group()`

Terminates the redirection begun by a call to `cairo.context#push_group()` or `cairo.context#push_group_with_c`
and returns a new pattern containing the results of all drawing operations performed to the group.

The `cairo.context#pop_group()` function calls cairo_restore(), (balancing a call to `cairo_save()` by the `push_group` function), so that any changes to the graphics state will not be visible outside the group.

`cairo.context#pop_group_to_source():reduce`

Terminates the redirection begun by a call to `cairo.context#push_group()` or `cairo.context#push_group_with_c`
and installs the resulting pattern as the source pattern in the given cairo context.

The `cairo.context#pop_group()` function calls cairo_restore(), (balancing a call to cairo_save() by the push_group function), so that any changes to the graphics state will not be visible outside the group.

`cairo.context#get_group_target()`

Gets the current destination surface for the context. This is either the original target surface as passed to `cairo.context` constructor or the target surface for the current group as started by the most recent call to `cairo.context#push_group()` or `cairo.context#push_group_with_content()`.

`cairo.context#set_source_rgb(red:number, green:number, blue:number):reduce`

Sets the source pattern within cr to an opaque color. This opaque color will then be used for any subsequent drawing operation until a new source pattern is set.

The color components are floating point numbers in the range 0 to 1. If the values passed in are outside that range, they will be clamped.

The default source pattern is opaque black, (that is, it is equivalent to `cr.set_source_rgb(0.0, 0.0, 0.0)`).

`cairo.context#set_source_rgba(red:number, green:number, blue:number, alpha:number):reduce`

Sets the source pattern within cr to a translucent color. This color will then be used for any subsequent drawing operation until a new source pattern is set.

The color and alpha components are floating point numbers in the range 0 to 1. If the values passed in are outside that range, they will be clamped.

The default source pattern is opaque black, (that is, it is equivalent to `cr.set_source_rgba(0.0, 0.0, 0.0, 1.0)`).

`cairo.context#set_source(source:cairo.pattern):reduce`

Sets the source pattern within cr to source. This pattern will then be used for any subsequent drawing operation until a new source pattern is set.

Note: The pattern's transformation matrix will be locked to the user space in effect at the time of `cairo.context#set_source()`. This means that further modifications of the current transformation matrix will not affect the source pattern. See `cairo.pattern#set_matrix()`.

The default source pattern is a solid pattern that is opaque black, (that is, it is equivalent to `cr.set_source_rgb(0.0, 0.0, 0.0)`).

`cairo.context#set_source_surface(surface:cairo.surface, x:number, y:number):reduce`

This is a convenience function for creating a pattern from `surface` and setting it as the source in `cr` with `cairo.context#set_source()`.

The `x` and `y` parameters give the user-space coordinate at which the surface origin should appear. (The surface origin is its upper-left corner before any transformation has been applied.) The x and y parameters are negated and then set as translation values in the pattern matrix.

Other than the initial translation pattern matrix, as described above, all other pattern attributes, (such as its extend mode), are set to the default values as in `cairo.pattern.create_for_surface()`. The resulting pattern can be queried with `cairo.context#get_source()` so that these attributes can be modified if desired, (eg. to create a repeating pattern with `cairo.pattern#set_extend()`).

`cairo.context#get_source()`

Gets the current source pattern for `cr`.

`cairo.context#set_antialias(antialias:number):reduce`

Set the antialiasing mode of the rasterizer used for drawing shapes. This value is a hint, and a particular backend may or may not support a particular value. At the current time, no backend supports `cairo.ANTIALIAS_SUBPIXEL` when drawing shapes.

Note that this option does not affect text rendering, instead see `cairo.font_options#set_antialias()`.

`cairo.context#get_antialias()`

Gets the current shape antialiasing mode, as set by `cairo.context#set_antialias()`.

`cairo.context#set_dash(dashes[]:number, offset:number):reduce`

Sets the dash pattern to be used by `cairo.context#stroke()`. A dash pattern is specified by dashes, an array of positive values. Each value provides the length of alternate "on" and "off" portions of the stroke. The offset specifies an offset into the pattern at which the stroke begins.

Each "on" segment will have caps applied as if the segment were a separate sub-path. In particular, it is valid to use an "on" length of 0.0 with `cairo.LINE_CAP_ROUND` or `cairo.LINE_CAP_SQUARE` in order to distributed dots or squares along a path.

Note: The length values are in user-space units as evaluated at the time of stroking. This is not necessarily the same as the user space at the time of `cairo.context#set_dash()`.

If length of dashes is 0 dashing is disabled.

If length of dashes is 1 a symmetric pattern is assumed with alternating on and off portions of the size specified by the single value in dashes.

If any value in dashes is negative, or if all values are 0, then cr will be put into an error state with a status of `cairo.STATUS_INVALID_DASH`.

`cairo.context#get_dash()`

Gets the current dash array.

`cairo.context#set_fill_rule(fill_rule:number):reduce`

Set the current fill rule within the cairo context. The fill rule is used to determine which regions are inside or outside a complex (potentially self-intersecting) path. The current fill rule affects both `cairo.context#fill()` and `cairo.context#clip()`. See cairo_fill_rule_t for details on the semantics of each available fill rule.

The default fill rule is `cairo.FILL_RULE_WINDING`.

`cairo.context#get_fill_rule()`

Gets the current fill rule, as set by `cairo.context#set_fill_rule()`.

`cairo.context#set_line_cap(line_cap:number):reduce`

Sets the current line cap style within the cairo context. See `cairo_line_cap_t` for details about how the available line cap styles are drawn.

As with the other stroke parameters, the current line cap style is examined by `cairo.context#stroke()`, `cairo.context#stroke_extents()`, and `cairo.context#stroke_to_path()`, but does not have any effect during path construction.

The default line cap style is `cairo.LINE_CAP_BUTT`.

`cairo.context#get_line_cap()`

Gets the current line cap style, as set by `cairo.context#set_line_cap()`.

`cairo.context#set_line_join(line_join:number):reduce`

Sets the current line join style within the cairo context. See `cairo_line_join_t` for details about how the available line join styles are drawn.

As with the other stroke parameters, the current line join style is examined by `cairo.context#stroke()`, `cairo.context#stroke_extents()`, and `cairo.context#stroke_to_path()`, but does not have any effect during path construction.

The default line join style is `cairo.LINE_JOIN_MITER`.

`cairo.context#get_line_join()`

Gets the current line join style, as set by `cairo.context#set_line_join()`.

`cairo.context#set_line_width(width:number):reduce`

Sets the current line width within the cairo context. The line width value specifies the diameter of a pen that is circular in user space, (though device-space pen may be an ellipse in general due to scaling/shear/rotation of the CTM).

Note: When the description above refers to user space and CTM it refers to the user space and CTM in effect at the time of the stroking operation, not the user space and CTM in effect at the time of the call to `cairo.context#set_line_width()`. The simplest usage makes both of these spaces identical. That is, if there is no change to the CTM between a call to `cairo.context#set_line_width()` and the stroking operation, then one can just pass user-space values to `cairo.context#set_line_width()` and ignore this note.

As with the other stroke parameters, the current line width is examined by `cairo.context#stroke()`, `cairo.context#stroke_extents()`, and `cairo.context#stroke_to_path()`, but does not have any effect during path construction.

The default line width value is 2.0.

`cairo.context#get_line_width()`

This function returns the current line width value exactly as set by `cairo.context#set_line_width()`. Note that the value is unchanged even if the CTM has changed between the calls to `cairo.context#set_line_width` and `cairo.context#get_line_width()`.

`cairo.context#set_miter_limit(limit:number):reduce`

Sets the current miter limit within the cairo context.

If the current line join style is set to `cairo.LINE_JOIN_MITER` (see `cairo.context#set_line_join()`), the miter limit is used to determine whether the lines should be joined with a bevel instead of a miter. Cairo divides the length of the miter by the line width. If the result is greater than the miter limit, the style is converted to a bevel.

As with the other stroke parameters, the current line miter limit is examined by `cairo.context#stroke()`, `cairo.context#stroke_extents()`, and `cairo.context#stroke_to_path()`, but does not have any effect during path construction.

The default miter limit value is 10.0, which will convert joins with interior angles less than 11 degrees to bevels instead of miters. For reference, a miter limit of 2.0 makes the miter cutoff at 60 degrees, and a miter limit of 1.414 makes the cutoff at 90 degrees.

A miter limit for a desired angle can be computed as: miter limit = 1/sin(angle/2)

`cairo.context#get_miter_limit()`

Gets the current miter limit, as set by cairo.context#set_miter_limit().

`cairo.context#set_operator(op:number):reduce`

Sets the compositing operator to be used for all drawing operations. See `cairo_operator_t` for details on the semantics of each available compositing operator.

The default operator is `cairo.OPERATOR_OVER`.

`cairo.context#get_operator()`

Gets the current compositing operator for a cairo context.

`cairo.context#set_tolerance(tolerance:number):reduce`

Sets the tolerance used when converting paths into trapezoids. Curved segments of the path will be subdivided until the maximum deviation between the original path and the polygonal approximation is less than tolerance. The default value is 0.1. A larger value will give better performance, a smaller value, better appearance. (Reducing the value from the default value of 0.1 is unlikely to improve appearance significantly.) The accuracy of paths within Cairo is limited by the precision of its internal arithmetic, and the prescribed tolerance is restricted to the smallest representable internal value.

`cairo.context#get_tolerance()`

Gets the current tolerance value, as set by `cairo.context#set_tolerance()`.

`cairo.context#clip():reduce`

Establishes a new clip region by intersecting the current clip region with the current path as it would be filled by `cairo.context#fill()` and according to the current fill rule (see `cairo.context#set_fill_rule()`).

After `cairo.context#clip()`, the current path will be cleared from the cairo context.

The current clip region affects all drawing operations by effectively masking out any changes to the surface that are outside the current clip region.

Calling `cairo.context#clip()` can only make the clip region smaller, never larger. But the current clip is part of the graphics state, so a temporary restriction of the clip region can be achieved by calling `cairo.context#clip()` within a `cairo.context#save()/cairo.context#restore()` pair. The only other means of increasing the size of the clip region is `cairo.context#reset_clip()`.

`cairo.context#clip_preserve():reduce`

Establishes a new clip region by intersecting the current clip region with the current path as it would be filled by `cairo.context#fill()` and according to the current fill rule (see `cairo.context#set_fill_rule()`). Unlike `cairo.context#clip()`, `cairo.context#clip_preserve()` preserves the path within the cairo context.

The current clip region affects all drawing operations by effectively masking out any changes to the surface that are outside the current clip region.

Calling `cairo.context#clip_preserve()` can only make the clip region smaller, never larger. But the current clip is part of the graphics state, so a temporary restriction of the clip region can be achieved by calling `cairo.context#clip_preserve()` within a `cairo.context#save()/cairo.context#restor` pair. The only other means of increasing the size of the clip region is `cairo.context#reset_clip()`.

`cairo.context#clip_extents()`

Computes a bounding box in user coordinates covering the area inside the current clip.

`cairo.context#in_clip(x:number, y:number)`

Tests whether the given point is inside the area that would be visible through the current clip, i.e. the area that would be filled by a `cairo.context#paint()` operation.

See `cairo.context#clip()`, and `cairo.context#clip_preserve()`.

`cairo.context#reset_clip():reduce`

Reset the current clip region to its original, unrestricted state. That is, set the clip region to an infinitely large shape containing the target surface. Equivalently, if infinity is too hard to grasp, one can imagine the clip region being reset to the exact bounds of the target surface.

Note that code meant to be reusable should not call `cairo.context#reset_clip()` as it will cause results unexpected by higher-level code which calls `cairo.context#clip()`. Consider using `cairo.context#save()` and `cairo.context#restore()` around `cairo.context#clip()` as a more robust means of temporarily restricting the clip region.

`cairo.context#copy_clip_rectangle_list()`

Gets the current clip region as a list of rectangles in user coordinates.

The status in the list may be `cairo.STATUS_CLIP_NOT_REPRESENTABLE` to indicate that the clip region cannot be represented as a list of user-space rectangles. The status may have other values to indicate other errors.

`cairo.context#fill():reduce`

A drawing operator that fills the current path according to the current fill rule, (each sub-path is implicitly closed before being filled). After `cairo.context#fill()`, the current path will be cleared from the cairo context. See `cairo.context#set_fill_rule()` and `cairo.context#fill_preserve()`.

`cairo.context#fill_preserve():reduce`

A drawing operator that fills the current path according to the current fill rule, (each sub-path is implicitly closed before being filled). Unlike `cairo.context#fill()`, `cairo.context#fill_preserve()` preserves the path within the cairo context.

See `cairo.context#set_fill_rule()` and `cairo.context#fill()`.

`cairo.context#fill_extents():reduce`

Computes a bounding box in user coordinates covering the area that would be affected, (the "inked" area), by a `cairo.context#fill()` operation given the current path and fill parameters. If the current path is empty, returns an empty rectangle ((0,0), (0,0)). Surface dimensions and clipping are not taken into account.

Contrast with `cairo.context#path_extents()`, which is similar, but returns non-zero extents for some paths with no inked area, (such as a simple line segment).

Note that `cairo.context#fill_extents()` must necessarily do more work to compute the precise inked areas in light of the fill rule, so `cairo.context#path_extents()` may be more desirable for sake of performance if the non-inked path extents are desired.

See `cairo.context#fill()`, `cairo.context#set_fill_rule()` and `cairo.context#fill_preserve()`.

`cairo.context#in_fill(x:number, y:number)`

Tests whether the given point is inside the area that would be affected by a `cairo.context#fill()` operation given the current path and filling parameters. Surface dimensions and clipping are not taken into account.

See `cairo.context#fill()`, `cairo.context#set_fill_rule()` and `cairo.context#fill_preserve()`.

`cairo.context#mask(pattern:cairo.pattern):reduce`

A drawing operator that paints the current source using the alpha channel of pattern as a mask. (Opaque areas of pattern are painted with the source, transparent areas are not painted.)

`cairo.context#mask_surface(surface:cairo.surface, surface_x:number, surface_y:number):reduce`

A drawing operator that paints the current source using the alpha channel of surface as a mask. (Opaque areas of surface are painted with the source, transparent areas are not painted.)

`cairo.context#paint():reduce`

A drawing operator that paints the current source everywhere within the current clip region.

`cairo.context#paint_with_alpha(alpha:number):reduce`

A drawing operator that paints the current source everywhere within the current clip region using a mask of constant alpha value alpha. The effect is similar to cairo.context#paint(), but the drawing is faded out using the alpha value.

`cairo.context#stroke():reduce`

A drawing operator that strokes the current path according to the current line width, line join, line cap, and dash settings. After `cairo.context#stroke()`, the current path will be cleared from the cairo context. See `cairo.context#set_line_width()`, `cairo.context#set_line_join()`, `cairo.context#set_line_cap()`, `cairo.context#set_dash()`, and `cairo.context#stroke_preserve()`.

Note: Degenerate segments and sub-paths are treated specially and provide a useful result. These can result in two different situations:

1. Zero-length "on" segments set in cairo.context#set_dash(). If the cap style is `cairo.LINE_CAP_ROUND` or `cairo.LINE_CAP_SQUARE` then these segments will be drawn as circular dots or squares

respectively. In the case of `cairo.LINE_CAP_SQUARE`, the orientation of the squares is determined by the direction of the underlying path.

2. A sub-path created by `cairo.context#move_to()` followed by either a `cairo.context#close_path()` or one or more calls to `cairo.context#line_to()` to the same coordinate as the `cairo.context#move_to()`. If the cap style is `cairo.LINE_CAP_ROUND` then these sub-paths will be drawn as circular dots. Note that in the case of `cairo.LINE_CAP_SQUARE` a degenerate sub-path will not be drawn at all, (since the correct orientation is indeterminate).

In no case will a cap style of `cairo.LINE_CAP_BUTT` cause anything to be drawn in the case of either degenerate segments or sub-paths.

`cairo.context#stroke_preserve():reduce`

A drawing operator that strokes the current path according to the current line width, line join, line cap, and dash settings. Unlike `cairo.context#stroke()`, `cairo.context#stroke_preserve()` preserves the path within the cairo context.

See `cairo.context#set_line_width()`, `cairo.context#set_line_join()`, `cairo.context#set_line_cap()`, `cairo.context#set_dash()`, and `cairo.context#stroke_preserve()`.

`cairo.context#stroke_extents()`

Computes a bounding box in user coordinates covering the area that would be affected, (the "inked" area), by a cairo.context#stroke() operation given the current path and stroke parameters. If the current path is empty, returns an empty rectangle ((0,0), (0,0)). Surface dimensions and clipping are not taken into account.

Note that if the line width is set to exactly zero, then `cairo.context#stroke_extents()` will return an empty rectangle. Contrast with `cairo.context#path_extents()` which can be used to compute the non-empty bounds as the line width approaches zero.

Note that `cairo.context#stroke_extents()` must necessarily do more work to compute the precise inked areas in light of the stroke parameters, so `cairo.context#path_extents()` may be more desirable for sake of performance if non-inked path extents are desired.

See `cairo.context#stroke()`, `cairo.context#set_line_width()`, `cairo.context#set_line_join()`, `cairo.context#set_line_cap()`, `cairo.context#set_dash()`, and `cairo.context#stroke_preserve()`.

`cairo.context#in_stroke(x:number, y:number)`

Tests whether the given point is inside the area that would be affected by a `cairo.context#stroke()` operation given the current path and stroking parameters. Surface dimensions and clipping are not taken into account. See `cairo.context#stroke()`, `cairo.context#set_line_width()`, `cairo.context#set_line_join()`, `cairo.context#set_line_cap()`, `cairo.context#_set_dash()`, and `cairo.context#stroke_preserve()`.

`cairo.context#copy_page():reduce`

Emits the current page for backends that support multiple pages, but doesn't clear it, so, the contents of the current page will be retained for the next page too. Use `cairo.cairo#show_page()` if you want to get an empty page after the emission.

This is a convenience function that simply calls `cairo.context#surface_copy_page()` on cr's target.

`cairo.context#show_page():reduce`

Emits and clears the current page for backends that support multiple pages. Use `cairo.context#copy_page()` if you don't want to clear the page.

This is a convenience function that simply calls `cairo.context#surface_show_page()` on cr's target.

Types and Values

cairo.antialias

- cairo.ANTIALIAS_DEFAULT
- cairo.ANTIALIAS_NONE
- cairo.ANTIALIAS_GRAY
- cairo.ANTIALIAS_SUBPIXEL
- cairo.ANTIALIAS_FAST
- cairo.ANTIALIAS_GOOD
- cairo.ANTIALIAS_BEST

cairo.fill_fule

- cairo.FILL_RULE_WINDING
- cairo.FILL_RULE_EVEN_ODD

cairo.line_cap

- cairo.LINE_CAP_BUTT
- cairo.LINE_CAP_ROUND
- cairo.LINE_CAP_SQUARE

cairo.line_join

- cairo.LINE_JOIN_MITER
- cairo.LINE_JOIN_ROUND
- cairo.LINE_JOIN_BEVEL

cairo.operator

- cairo.OPERATOR_CLEAR
- cairo.OPERATOR_SOURCE
- cairo.OPERATOR_OVER
- cairo.OPERATOR_IN
- cairo.OPERATOR_OUT
- cairo.OPERATOR_ATOP
- cairo.OPERATOR_DEST
- cairo.OPERATOR_DEST_OVER
- cairo.OPERATOR_DEST_IN
- cairo.OPERATOR_DEST_OUT
- cairo.OPERATOR_DEST_ATOP

- `cairo.OPERATOR_XOR`
- `cairo.OPERATOR_ADD`
- `cairo.OPERATOR_SATURATE`
- `cairo.OPERATOR_MULTIPLY`
- `cairo.OPERATOR_SCREEN`
- `cairo.OPERATOR_OVERLAY`
- `cairo.OPERATOR_DARKEN`
- `cairo.OPERATOR_LIGHTEN`
- `cairo.OPERATOR_COLOR_DODGE`
- `cairo.OPERATOR_COLOR_BURN`
- `cairo.OPERATOR_HARD_LIGHT`
- `cairo.OPERATOR_SOFT_LIGHT`
- `cairo.OPERATOR_DIFFERENCE`
- `cairo.OPERATOR_EXCLUSION`
- `cairo.OPERATOR_HSL_HUE`
- `cairo.OPERATOR_HSL_SATURATION`
- `cairo.OPERATOR_HSL_COLOR`
- `cairo.OPERATOR_HSL_LUMINOSITY`

10.1.2 Paths - Creating paths and manipulating path data

Functions

`cairo.context#copy_path()`

Creates a copy of the current path and returns it to the user as a `cairo.path`. See `cairo_path_data_t` for hints on how to iterate over the returned data structure.

The result will have no data (data==nullptr and num_data==0), if either of the following conditions hold:

1. If there is insufficient memory to copy the path. In this case path-¿status will be set to `cairo.STATUS_NO_MEMORY`.
2. If cr is already in an error state. In this case `path.status` will contain the same status that would be returned by `cairo.context#status()`.

`cairo.context#copy_path_flat()`

Gets a flattened copy of the current path and returns it to the user as a cairo.path. See cairo_path_data_t for hints on how to iterate over the returned data structure.

This function is like `cairo.context#copy_path()` except that any curves in the path will be approximated with piecewise-linear approximations, (accurate to within the current tolerance value). That is, the result is guaranteed to not have any elements of type `cairo.PATH_CURVE_TO` which will instead be replaced by a series of `cairo.PATH_LINE_TO` elements.

The result will have no data (data==nullptr and num_data==0), if either of the following conditions hold:

1. If there is insufficient memory to copy the path. In this case path.status will be set to `cairo.STATUS_NO_MEMORY`.

2. If cr is already in an error state. In this case path-¿status will contain the same status that would be returned by `cairo.context#status()`.

`cairo.context#append_path(path:cairo.path):reduce`

Append the path onto the current path. The path may be either the return value from one of `cairo.context#copy_path()` or `cairo.context#copy_path_flat()` or it may be constructed manually. See `cairo.path` for details on how the path data structure should be initialized, and note that `path.status` must be initialized to `cairo.STATUS_SUCCESS`.

`cairo.context#has_current_point()`

Returns whether a current point is defined on the current path. See cairo.context#get_current_point() for details on the current point.

`cairo.context#get_current_point()`

Gets the current point of the current path, which is conceptually the final point reached by the path so far.

The current point is returned in the user-space coordinate system. If there is no defined current point or if cr is in an error status, x and y will both be set to 0.0. It is possible to check this in advance with cairo.context#has_current_point().

Most path construction functions alter the current point. See the following for details on how they affect the current point: cairo.context#new_path(), cairo.context#new_sub_path(), cairo.context#append_path(), cairo.context#close_path(), cairo.context#move_to(), cairo.context#line_to(), cairo.context#curve_to(), cairo.context#rel_move_to(), cairo.context#rel_line_to(), cairo.context#rel_curve_to(), cairo.context#arc(), cairo.context#arc_negative(), cairo.context#rectangle(), cairo.context#text_path(), cairo.context#glyph_path(), cairo.context#stroke_to_path().

Some functions use and alter the current point but do not otherwise change current path: cairo.context#show_text().

Some functions unset the current path and as a result, current point: cairo.context#fill(), cairo.context#stroke().

`cairo.context#new_path():reduce`

Clears the current path. After this call there will be no path and no current point.

`cairo.context#new_sub_path():reduce`

Begin a new sub-path. Note that the existing path is not affected. After this call there will be no current point.

In many cases, this call is not needed since new sub-paths are frequently started with cairo.context#move_to().

A call to cairo.context#new_sub_path() is particularly useful when beginning a new sub-path with one of the cairo.context#arc() calls. This makes things easier as it is no longer necessary to manually compute the arc's initial coordinates for a call to cairo.context#move_to().

`cairo.context#close_path():reduce`

Adds a line segment to the path from the current point to the beginning of the current sub-path, (the most recent point passed to cairo.context#move_to()), and closes this sub-path. After this call the current point will be at the joined endpoint of the sub-path.

The behavior of cairo.context#close_path() is distinct from simply calling cairo.context#line_to() with the equivalent coordinate in the case of stroking. When a closed sub-path is stroked, there are no caps on the ends of the sub-path. Instead, there is a line join connecting the final and initial segments of the sub-path.

If there is no current point before the call to cairo.context#close_path(), this function will have no effect.

Note: As of cairo version 1.2.4 any call to cairo.context#close_path() will place an explicit MOVE_TO element into the path immediately after the CLOSE_PATH element, (which can be seen in cairo.context#copy_path() for example). This can simplify path processing in some cases as it may not be necessary to save the "last move_to point" during processing as the MOVE_TO immediately after the CLOSE_PATH will provide that point.

`cairo.context#arc(xc:number, yc:number, radius:number, angle1?:number, angle2?:number):map:reduce:[deg]`

Adds a circular arc of the given radius to the current path. The arc is centered at (xc, yc), begins at angle1 and proceeds in the direction of increasing angles to end at angle2. If angle2 is less than angle1 it will be progressively increased by 2*M_PI until it is greater than angle1.

If there is a current point, an initial line segment will be added to the path to connect the current point to the beginning of the arc. If this initial line is undesired, it can be avoided by calling cairo.context#new_sub_path() before calling cairo.context#arc().

Angles are measured in radians. An angle of 0.0 is in the direction of the positive X axis (in user space). An angle of math.pi/2.0 radians (90 degrees) is in the direction of the positive Y axis (in user space). Angles increase in the direction from the positive X axis toward the positive Y axis. So with the default transformation matrix, angles increase in a clockwise direction.

(To convert from degrees to radians, use degrees * (math.pi / 180.).)

This function gives the arc in the direction of increasing angles; see cairo.context#arc_negative() to get the arc in the direction of decreasing angles.

The arc is circular in user space. To achieve an elliptical arc, you can scale the current transformation matrix by different amounts in the X and Y directions. For example, to draw an ellipse in the box given by x, y, width, height:

cr.save() cr.translate(x + width / 2., y + height / 2.) cr.scale(width / 2., height / 2.) cr.arc(0., 0., 1., 0., 2 * math.pi)cr.restore()

Gura: If attribute :deg is specified, angle1 and angle2 are represented in degrees instead of radians.

`cairo.context#arc_negative(xc:number, yc:number, radius:number, angle1?:number, angle2?:number):map:reduce`

Adds a circular arc of the given radius to the current path. The arc is centered at (xc, yc), begins at angle1 and proceeds in the direction of decreasing angles to end at angle2. If angle2 is greater than angle1 it will be progressively decreased by 2*math.pi until it is less than angle1.

See cairo.context#arc() for more details. This function differs only in the direction of the arc between the two angles.

Gura: If attribute :deg is specified, angle1 and angle2 are represented in degrees instead of radians.

`cairo.context#curve_to(x1:number, y1:number, x2:number, y2:number, x3:number, y3:number):map:reduce`

Adds a cubic Bezier spline to the path from the current point to position (x3, y3) in user-space coordinates, using (x1, y1) and (x2, y2) as the control points. After this call the current point will be (x3, y3).

If there is no current point before the call to cairo.context#curve_to() this function will behave as if preceded by a call to cr.move_to(x1, y1).

`cairo.context#line_to(x:number, y:number):map:reduce`

Adds a line to the path from the current point to position (x, y) in user-space coordinates. After this call the current point will be (x, y).

If there is no current point before the call to cairo.context#line_to() this function will behave as cr.move_to(x, y).

`cairo.context#move_to(x:number, y:number):map:reduce`

Begin a new sub-path. After this call the current point will be (x, y).

`cairo.context#rectangle(x:number, y:number, width:number, height:number):map:reduce`

Adds a closed sub-path rectangle of the given size to the current path at position (x, y) in user-space coordinates.

This function is logically equivalent to:

cr.move_to(x, y) cr.rel_line_to(width, 0) cr.rel_line_to(0, height) cr.rel_line_to(-width, 0) cr.close_path()

`cairo.context#text_path(text:string):map:reduce`

Adds closed paths for text to the current path. The generated path if filled, achieves an effect similar to that of cairo.context#show_text().

Text conversion and positioning is done similar to cairo.context#show_text().

Like cairo.context#show_text(), After this call the current point is moved to the origin of where the next glyph would be placed in this same progression. That is, the current point will be at the origin of the final glyph offset by its advance values. This allows for chaining multiple calls to to cairo.context#text_path() without having to set current point in between.

Note: The cairo.context#text_path() function call is part of what the cairo designers call the "toy" text API. It is convenient for short demos and simple programs, but it is not expected to be adequate for serious text-using applications. See cairo.context#glyph_path() for the "real" text path API in cairo.

`cairo.context#rel_curve_to(dx1:number, dy1:number, dx2:number, dy2:number, dx3:number, dy3:number):map:red`

Relative-coordinate version of cairo.context#curve_to(). All offsets are relative to the current point. Adds a cubic Bezier spline to the path from the current point to a point offset from the current point by (dx3, dy3), using points offset by (dx1, dy1) and (dx2, dy2) as the control points. After this call the current point will be offset by (dx3, dy3).

Given a current point of (x, y), cr.rel_curve_to(dx1, dy1, dx2, dy2, dx3, dy3) is logically equivalent to cr.curve_to(x+dx1, y+dy1, x+dx2, y+dy2, x+dx3, y+dy3).

It is an error to call this function with no current point. Doing so will cause cr to shutdown with a status of cairo.STATUS_NO_CURRENT_POINT.

`cairo.context#rel_line_to(dx:number, dy:number):map:reduce`

Relative-coordinate version of cairo.context#line_to(). Adds a line to the path from the current point to a point that is offset from the current point by (dx, dy) in user space. After this call the current point will be offset by (dx, dy).

Given a current point of (x, y), cr.rel_line_to(dx, dy) is logically equivalent to cr.line_to(x + dx, y + dy).

It is an error to call this function with no current point. Doing so will cause cr to shutdown with a status of cairo.STATUS_NO_CURRENT_POINT.

`cairo.context#rel_move_to(dx:number, dy:number):map:reduce`

Begin a new sub-path. After this call the current point will offset by (dx, dy).

Given a current point of (x, y), cr.rel_move_to(dx, dy) is logically equivalent to cr.move_to(x + dx, y + dy).

It is an error to call this function with no current point. Doing so will cause cr to shutdown with a status of cairo.STATUS_NO_CURRENT_POINT.

`cairo.context#path_extents()`

Computes a bounding box in user-space coordinates covering the points on the current path. If the current path is empty, returns an empty rectangle ((0,0), (0,0)). Stroke parameters, fill rule, surface dimensions and clipping are not taken into account.

Contrast with cairo.context#fill_extents() and cairo.context#stroke_extents() which return the extents of only the area that would be "inked" by the corresponding drawing operations.

The result of cairo.context#path_extents() is defined as equivalent to the limit of cairo.context#stroke_extents() with cairo.LINE_CAP_ROUND as the line width approaches 0.0, (but never reaching the empty-rectangle returned by cairo.context#stroke_extents() for a line width of 0.0).

Specifically, this means that zero-area sub-paths such as cairo.context#move_to();cairo.context#line_to() segments, (even degenerate cases where the coordinates to both calls are identical), will be considered as contributing to the extents. However, a lone cairo.context#move_to() will not contribute to the results of cairo.context#path_extents().

Types and Values

10.1.3 cairo.pattern - Sources for drawing

Functions

`cairo.pattern#add_color_stop_rgb(offset:number, red:number, green:number, blue:number):reduce`

Adds an opaque color stop to a gradient pattern. The offset specifies the location along the gradient's control vector. For example, a linear gradient's control vector is from (x0,y0) to (x1,y1) while a radial gradient's control vector is from any point on the start circle to the corresponding point on the end circle.

The color is specified in the same way as in cairo.context#set_source_rgb().

If two (or more) stops are specified with identical offset values, they will be sorted according to the order in which the stops are added, (stops added earlier will compare less than stops added later). This can be useful for reliably making sharp color transitions instead of the typical blend.

Note: If the pattern is not a gradient pattern, (eg. a linear or radial pattern), then the pattern will be put into an error status with a status of cairo.STATUS_PATTERN_TYPE_MISMATCH.

`cairo.pattern#add_color_stop_rgba(offset:number, red:number, green:number, blue:number, alpha:number):redu`

Adds a translucent color stop to a gradient pattern. The offset specifies the location along the gradient's control vector. For example, a linear gradient's control vector is from (x0,y0) to (x1,y1) while a radial gradient's control vector is from any point on the start circle to the corresponding point on the end circle.

The color is specified in the same way as in cairo.context#set_source_rgba().

If two (or more) stops are specified with identical offset values, they will be sorted according to the order in which the stops are added, (stops added earlier will compare less than stops added later). This can be useful for reliably making sharp color transitions instead of the typical blend.

Note: If the pattern is not a gradient pattern, (eg. a linear or radial pattern), then the pattern will be put into an error status with a status of cairo.STATUS_PATTERN_TYPE_MISMATCH.

`cairo.pattern#get_color_stop_count()`

Gets the number of color stops specified in the given gradient pattern.

`cairo.pattern#get_color_stop_rgba(index:number)`

Gets the color and offset information at the given index for a gradient pattern. Values of index are 0 to 1 less than the number returned by cairo.pattern#get_color_stop_count().

`cairo.pattern.create_rgb(red:number, green:number, blue:number):static {block?}`

Creates a new cairo.pattern corresponding to an opaque color. The color components are floating point numbers in the range 0 to 1. If the values passed in are outside that range, they will be clamped.

`cairo.pattern.create_rgba(red:number, green:number, blue:number, alpha:number):static {block?}`

Creates a new cairo,pattern corresponding to a translucent color. The color components are floating point numbers in the range 0 to 1. If the values passed in are outside that range, they will be clamped.

`cairo.pattern#get_rgba()`

Gets the solid color for a solid color pattern.

`cairo.pattern.create_for_surface(surface:cairo.surface):static {block?}`

Create a new cairo.pattern for the given surface.

`cairo.pattern#get_surface()`

Gets the surface of a surface pattern. The reference returned in surface is owned by the pattern; the caller should call cairo_surface_reference() if the surface is to be retained.

`cairo.pattern.create_linear(x0:number, y0:number, x1:number, y1:number):static {block?}`

Create a new linear gradient cairo.pattern along the line defined by (x0, y0) and (x1, y1). Before using the gradient pattern, a number of color stops should be defined using cairo.pattern#add_color_stop_rgb() or cairo.pattern#add_color_stop_rgba().

Note: The coordinates here are in pattern space. For a new pattern, pattern space is identical to user space, but the relationship between the spaces can be changed with cairo.pattern#set_matrix().

`cairo.pattern#get_linear_points()`

Gets the gradient endpoints for a linear gradient.

`cairo.pattern.create_radial(cx0:number, cy0:number, radius0:number, cx1:number, cy1:number, radius1:number`

Creates a new radial gradient cairo_pattern_t between the two circles defined by (cx0, cy0, radius0) and (cx1, cy1, radius1). Before using the gradient pattern, a number of color stops should be defined using cairo.pattern#add_color_stop_rgb() or cairo.pattern#add_color_stop_rgba().

Note: The coordinates here are in pattern space. For a new pattern, pattern space is identical to user space, but the relationship between the spaces can be changed with cairo.pattern#set_matrix().

`cairo.pattern#get_radial_circles()`

Gets the gradient endpoint circles for a radial gradient, each specified as a center coordinate and a radius.

`cairo.mesh_pattern.create():static {block?}`

`cairo.mesh_pattern#begin_patch():reduce`

`cairo.mesh_pattern#end_patch():reduce`

`cairo.mesh_pattern#move_to(x:number, y:number):reduce`

`cairo.mesh_pattern#line_to(x:number, y:number):reduce`

`cairo.mesh_pattern#curve_to(x1:number, y1:number, x2:number, y2:number, x3:number, y3:number):reduce`

`cairo.mesh_pattern#set_control_point(point_num:number, x:number, y:number):reduce`

`cairo.mesh_pattern#set_corner_color_rgb(corner_num:number, red:number, green:number, blue:number):reduce`

`cairo.mesh_pattern#set_corner_color_rgba(corner_num:number, red:number, green:number, blue:number, alpha:nu`

`cairo.pattern#status()`

Checks whether an error has previously occurred for this pattern.

`cairo.pattern#set_extend(extend:number):reduce`

Sets the mode to be used for drawing outside the area of a pattern. See cairo_extend_t for details on the semantics of each extend strategy.

The default extend mode is cairo.EXTEND_NONE for surface patterns and cairo.EXTEND_PAD for gradient patterns.

`cairo.pattern#get_extend()`

Gets the current extend mode for a pattern. See cairo_extend_t for details on the semantics of each extend strategy.

`cairo.pattern#set_filter(filter:number):reduce`

Sets the filter to be used for resizing when using this pattern. See cairo_filter_t for details on each filter.

- Note that you might want to control filtering even when you do not have an explicit cairo.pattern object, (for example when using cairo.context#set_source_surface()). In these cases, it is convenient to use cairo.context#get_source() to get access to the pattern that cairo creates implicitly. For example:

  cr.set_source_surface(image, x, y) cr.get_source().set_filter(cairo.FILTER_NEAREST)

`cairo.pattern#get_filter()`

Gets the current filter for a pattern. See cairo_filter_t for details on each filter.

`cairo.pattern#set_matrix(array:array@double):reduce`

Sets the pattern's transformation matrix to matrix. This matrix is a transformation from user space to pattern space.

When a pattern is first created it always has the identity matrix for its transformation matrix, which means that pattern space is initially identical to user space.

Important: Please note that the direction of this transformation matrix is from user space to pattern space. This means that if you imagine the flow from a pattern to user space (and on to device space), then coordinates in that flow will be transformed by the inverse of the pattern matrix.

For example, if you want to make a pattern appear twice as large as it does by default the correct code to use is:

cairo_matrix_init_scale (&matrix, 0.5, 0.5); cairo_pattern_set_matrix (pattern, &matrix);

Meanwhile, using values of 2.0 rather than 0.5 in the code above would cause the pattern to appear at half of its default size.

Also, please note the discussion of the user-space locking semantics of cairo.context#set_source().

`cairo.pattern#get_matrix()`

Stores the pattern's transformation matrix into matrix.

`cairo.pattern#get_type()`

This function returns the type a pattern. See cairo_pattern_type_t for available types.

Types and Values

`cairo.extend`

- `cairo.EXTEND_NONE`
- `cairo.EXTEND_REPEAT`
- `cairo.EXTEND_REFLECT`
- `cairo.EXTEND_PAD`

`cairo.filter`

- `cairo.FILTER_FAST`
- `cairo.FILTER_GOOD`
- `cairo.FILTER_BEST`
- `cairo.FILTER_NEAREST`
- `cairo.FILTER_BILINEAR`
- `cairo.FILTER_GAUSSIAN`

`cairo.pattern_type`

- `cairo.PATTERN_TYPE_SOLID`
- `cairo.PATTERN_TYPE_SURFACE`
- `cairo.PATTERN_TYPE_LINEAR`
- `cairo.PATTERN_TYPE_RADIAL`
- `cairo.PATTERN_TYPE_MESH`
- `cairo.PATTERN_TYPE_RASTER_SOURCE`

10.1.4 Regions - Representing a pixel-aligned area

cairo.region_overlap

- cairo.REGION_OVERLAP_IN
- cairo.REGION_OVERLAP_OUT
- cairo.REGION_OVERLAP_PART

Functions

cairo.region.create():static {block?}

cairo.region.create_rectangle(rectangle:cairo.rectangle_int):static {block?}

cairo.region.create_rectangles(rects[]:cairo.rectangle_int):static {block?}

cairo.region#copy() {block?}

cairo.region#status()

cairo.region#get_extents()

cairo.region#get_rectangle(nth:number)

cairo.region#is_empty()

cairo.region#contains_point(x:number, y:number)

cairo.region#contains_rectangle(rectangle:cairo.rectangle_int)

cairo.region#equal(region:cairo.region)

cairo.region#translate(dx:number, dy:number)

cairo.region#intersect(other:cairo.region)

cairo.region#intersect_rectangle(rectangle:cairo.rectangle_int)

cairo.region#union(other:cairo.region)

cairo.region#union_rectangle(rectangle:cairo.rectangle_int)

cairo.region#xor(other:cairo.region)

cairo.region#xor_rectangle(rectangle:cairo.rectangle_int)

Types and Values

10.1.5 Transformations - Manipulating the current transformation matrix

Functions

`cairo.context#translate(tx:number, ty:number):reduce`

Modifies the current transformation matrix (CTM) by translating the user-space origin by (tx, ty). This offset is interpreted as a user-space coordinate according to the CTM in place before the new call to cairo.context#translate(). In other words, the translation of the user-space origin takes place after any existing transformation.

`cairo.context#scale(sx:number, sy:number):reduce`

Modifies the current transformation matrix (CTM) by scaling the X and Y user-space axes by sx and sy respectively. The scaling of the axes takes place after any existing transformation of user space.

`cairo.context#rotate(angle:number):reduce:[deg]`

Modifies the current transformation matrix (CTM) by rotating the user-space axes by angle radians. The rotation of the axes takes places after any existing transformation of user space. The rotation direction for positive angles is from the positive X axis toward the positive Y axis.

Gura: If attribute :deg is specified, angle is represented in degrees instead of radians.

`cairo.context#transform(array:array@double):reduce`

Modifies the current transformation matrix (CTM) by applying matrix as an additional transformation. The new transformation of user space takes place after any existing transformation.

`cairo.context#set_matrix(array:array@double):reduce`

Modifies the current transformation matrix (CTM) by setting it equal to matrix.

`cairo.context#get_matrix()`

Stores the current transformation matrix (CTM) into matrix.

`cairo.context#identity_matrix():reduce`

Resets the current transformation matrix (CTM) by setting it equal to the identity matrix. That is, the user-space and device-space axes will be aligned and one user-space unit will transform to one device-space unit.

`cairo.context#user_to_device(x:number, y:number)`

Transform a coordinate from user space to device space by multiplying the given point by the current transformation matrix (CTM).

`cairo.context#user_to_device_distance(dx:number, dy:number)`

Transform a distance vector from user space to device space. This function is similar to cairo.context#user_to_device() except that the translation components of the CTM will be ignored when transforming (dx,dy).

`cairo.context#device_to_user(x:number, y:number)`

Transform a coordinate from device space to user space by multiplying the given point by the inverse of the current transformation matrix (CTM).

`cairo.context#device_to_user_distance(dx:number, dy:number)`

Transform a distance vector from device space to user space. This function is similar to cairo.context#device_to_user() except that the translation components of the inverse CTM will be ignored when transforming (dx,dy).

10.1.6 text - Rendering text and glyphs

Functions

`cairo.context#select_font_face(family:string, slant:number, weight:number):reduce`

Note: The cairo.context#select_font_face() function call is part of what the cairo designers call the "toy" text API. It is convenient for short demos and simple programs, but it is not expected to be adequate for serious text-using applications.

Selects a family and style of font from a simplified description as a family name, slant and weight. Cairo provides no operation to list available family names on the system (this is a "toy", remember), but the standard CSS2 generic family names, ("serif", "sans-serif", "cursive", "fantasy", "monospace"), are likely to work as expected.

If family starts with the string "cairo:", or if no native font backends are compiled in, cairo will use an internal font family. The internal font family recognizes many modifiers in the family string, most notably, it recognizes the string "monospace". That is, the family name "cairo:monospace" will use the monospace version of the internal font family.

For "real" font selection, see the font-backend-specific font_face_create functions for the font backend you are using. (For example, if you are using the freetype-based cairo-ft font backend, see cairo_ft_font_face_create_for_ft_face() or cairo_ft_font_face_create_for_pattern().) The resulting font face could then be used with cairo.scaled_font_create() and cairo.context#set_scaled_font().

Similarly, when using the "real" font support, you can call directly into the underlying font system, (such as fontconfig or freetype), for operations such as listing available fonts, etc.

It is expected that most applications will need to use a more comprehensive font handling and text layout library, (for example, pango), in conjunction with cairo.

If text is drawn without a call to cairo.context#select_font_face(), (nor cairo.context#set_font_face() nor cairo.context#set_scaled_font()), the default family is platform-specific, but is essentially "sans-serif". Default slant is cairo.FONT_SLANT_NORMAL, and default weight is cairo.FONT_WEIGHT_NORMA

This function is equivalent to a call to cairo.toy_font_face.create() followed by cairo.context#set_font_face().

`cairo.context#set_font_size(size:number):reduce`

Sets the current font matrix to a scale by a factor of size, replacing any font matrix previously set with cairo.context#set_font_size() or cairo.context#set_font_matrix(). This results in a font size of size user space units. (More precisely, this matrix will result in the font's em-square being a size by size square in user space.)

If text is drawn without a call to cairo.context#set_font_size(), (nor cairo.context#set_font_matrix() nor cairo.context#set_scaled_font()), the default font size is 10.0.

`cairo.context#set_font_matrix(array:array@double):reduce`

Sets the current font matrix to matrix. The font matrix gives a transformation from the design space of the font (in this space, the em-square is 1 unit by 1 unit) to user space. Normally, a simple scale is used (see cairo_set_font_size()), but a more complex font matrix can be used to shear the font or stretch it unequally along the two axes.

`cairo.context#get_font_matrix()`

Stores the current font matrix into matrix. See cairo.context#set_font_matrix().

`cairo.context#set_font_options(options:cairo.font_options):reduce`

Sets a set of custom font rendering options for the cairo_t. Rendering options are derived by merging these options with the options derived from underlying surface; if the value in options has a default value (like cairo.ANTIALIAS_DEFAULT), then the value from the surface is used.

`cairo.context#get_font_options()`

Retrieves font rendering options set via cairo.context#set_font_options. Note that the returned options do not include any options derived from the underlying surface; they are literally the options passed to cairo.context#set_font_options().

`cairo.context#set_font_face(font_face:cairo.font_face):reduce`

Replaces the current cairo_font_face_t object in the cairo_t with font_face. The replaced font face in the cairo_t will be destroyed if there are no other references to it.

`cairo.context#get_font_face()`

Gets the current font face for a cairo_t.

`cairo.context#set_scaled_font(scaled_font:cairo.scaled_font):reduce`

Replaces the current font face, font matrix, and font options in the cairo_t with those of the cairo_scaled_font_t. Except for some translation, the current CTM of the cairo_t should be the same as that of the cairo_scaled_font_t, which can be accessed using cairo.context#scaled_font_get_ctm().

`cairo.context#get_scaled_font()`

Gets the current scaled font for a cairo_t.

`cairo.context#show_text(text:string):reduce`

A drawing operator that generates the shape from a string of UTF-8 characters, rendered according to the current font_face, font_size (font_matrix), and font_options.

This function first computes a set of glyphs for the string of text. The first glyph is placed so that its origin is at the current point. The origin of each subsequent glyph is offset from that of the previous glyph by the advance values of the previous glyph.

After this call the current point is moved to the origin of where the next glyph would be placed in this same progression. That is, the current point will be at the origin of the final glyph offset by its advance values. This allows for easy display of a single logical string with multiple calls to cairo.context#show_text().

Note: The cairo.context#show_text() function call is part of what the cairo designers call the "toy" text API. It is convenient for short demos and simple programs, but it is not expected to be adequate for serious text-using applications. See cairo.context#show_glyphs() for the "real" text display API in cairo.

`cairo.context#show_glyphs(glyphs:cairo.glyph):reduce`

A drawing operator that generates the shape from an array of glyphs, rendered according to the current font face, font size (font matrix), and font options.

`cairo.context#font_extents()`

Gets the font extents for the currently selected font.

`cairo.context#text_extents(text:string)`

Gets the extents for a string of text. The extents describe a user-space rectangle that encloses the "inked" portion of the text, (as it would be drawn by cairo.context#show_text()). Additionally, the x_advance and y_advance values indicate the amount by which the current point would be advanced by cairo.context#show_text().

Note that whitespace characters do not directly contribute to the size of the rectangle (extents.width and extents.height). They do contribute indirectly by changing the position of non-whitespace characters. In particular, trailing whitespace characters are likely to not affect the size of the rectangle, though they will affect the x_advance and y_advance values.

`cairo.context#glyph_extents(glyphs:cairo.glyph)`

Gets the extents for an array of glyphs. The extents describe a user-space rectangle that encloses the "inked" portion of the glyphs, (as they would be drawn by cairo.context#show_glyphs()). Additionally, the x_advance and y_advance values indicate the amount by which the current point would be advanced by cairo.context#show_glyphs().

Note that whitespace glyphs do not contribute to the size of the rectangle (extents.width and extents.height).

`cairo.toy_font_face.create(family:string, slant:number, weight:number):static {block?}`

Creates a font face from a triplet of family, slant, and weight. These font faces are used in implementation of the the cairo_t "toy" font API.

If family is the zero-length string "", the platform-specific default family is assumed. The default family then can be queried using cairo.toy_font_face#get_family().

The cairo.context#select_font_face() function uses this to create font faces. See that function for limitations and other details of toy font faces.

`cairo.toy_font_face#get_family()`

Gets the familly name of a toy font.

`cairo.toy_font_face#get_slant()`

Gets the slant a toy font.

`cairo.toy_font_face#get_weight()`

Gets the weight a toy font.

Types and Values

10.1.7 Raster Sources - Supplying arbitary image data

Functions

10.2 Fonts

10.2.1 cairo.font_face - Base class for font faces

Functions

10.2.2 cairo.scaled_font - Font face at particular size and options

Functions

cairo.scaled_font.create(font_face:cairo.font_face, font_matrix:array@double, ctm:array@double, options):st

10.2.3 cairo_font_options_t - How a font should be rendered

Functions

cairo.font_options.create():static {block?}

cairo.font_options#status()

cairo.font_options#merge(other:cairo.font_options):void

cairo.font_options#hash()

cairo.font_options#equal(other:cairo.font_options)

cairo.font_options#set_antialias(antialias:number):void

cairo.font_options#get_antialias()

cairo.font_options#set_subpixel_order(subpixel_order:number):void

cairo.font_options#get_subpixel_order()

cairo.font_options#set_hint_style(hint_style:number):void

cairo.font_options#get_hint_style()

cairo.font_options#set_hint_metrics(hint_metrics:number):void

cairo.font_options#get_hint_metrics()

10.2.4 FreeType Fonts - Font support for FreeType

Functions

10.2.5 Win32 Fonts - Font support for Microsoft Windows

Functions

10.2.6 Quartz (CGFont) Fonts - Font support via CGFont on OS X

Functions

10.2.7 User Fonts - Font support with font data provided by the user

Functions

10.3 Surfaces

10.3.1 cairo.device - interface to underlying rendering system

Functions

`cairo.device#status()`

`cairo.device#finish():reduce`

`cairo.device#flush():reduce`

`cairo.device#get_type()`

`cairo.device#acquire()`

`cairo.device#release():void`

10.3.2 cairo.surface - Base class for surfaces

Functions

`cairo.surface.create_similar(other:cairo.surface, content:number, width:number, height:number):static {blo`

Create a new surface that is as compatible as possible with an existing surface. For example the new surface will have the same fallback resolution and font options as other. Generally, the new surface will also use the same backend as other, unless that is not possible for some reason. The type of the returned surface may be examined with cairo.surface#get_type().

Initially the surface contents are all 0 (transparent if contents have transparency, black otherwise.)

Use cairo.surface.create_similar_image() if you need an image surface which can be painted quickly to the target surface.

`cairo.surface.create_similar_image(other:cairo.surface, format:number, width:number, height:number):static`

Create a new image surface that is as compatible as possible for uploading to and the use in conjunction with an existing surface. However, this surface can still be used like any normal image surface.

Initially the surface contents are all 0 (transparent if contents have transparency, black otherwise.)

Use cairo.surface.create_similar() if you don't need an image surface.

`cairo.surface.create_for_rectangle(other:cairo.surface, format:number, width:number, height:number):static`

Create a new surface that is a rectangle within the target surface. All operations drawn to this surface are then clipped and translated onto the target surface. Nothing drawn via this sub-surface outside of its bounds is drawn onto the target surface, making this a useful method for passing constrained child surfaces to library routines that draw directly onto the parent surface, i.e. with no further backend allocations, double buffering or copies.

Note: The semantics of subsurfaces have not been finalized yet unless the rectangle is in full device units, is contained within the extents of the target surface, and the target or subsurface's device transforms are not changed.

`cairo.surface#status()`

Checks whether an error has previously occurred for this surface.

`cairo.surface#finish():reduce`

This function finishes the surface and drops all references to external resources. For example, for the Xlib backend it means that cairo will no longer access the drawable, which can be freed. After calling cairo.surface#finish() the only valid operations on a surface are getting and setting user, referencing and destroying, and flushing and finishing it. Further drawing to the surface will not affect the surface but will instead trigger a cairo.STATUS_SURFACE_FINISHED error.

When the last call to cairo_surface_destroy() decreases the reference count to zero, cairo will call cairo_surface_finish() if it hasn't been called already, before freeing the resources associated with the surface.

`cairo.surface#flush():reduce`

Do any pending drawing for the surface and also restore any temporary modifications cairo has made to the surface's state. This function must be called before switching from drawing on the surface with cairo to drawing on it directly with native APIs. If the surface doesn't support direct access, then this function does nothing.

`cairo.surface#get_device()`

This function returns the device for a surface. See cairo.device.

`cairo.surface#get_font_options()`

Retrieves the default font rendering options for the surface. This allows display surfaces to report the correct subpixel order for rendering on them, print surfaces to disable hinting of metrics and so forth. The result can then be used with cairo.scaled_font.create().

`cairo.surface#get_content()`

This function returns the content type of surface which indicates whether the surface contains color and/or alpha information. See cairo_content_t.

`cairo.surface#mark_dirty():reduce`

Tells cairo that drawing has been done to surface using means other than cairo, and that cairo should reread any cached areas. Note that you must call cairo.surface#flush() before doing such drawing.

`cairo.surface#mark_dirty_rectangle(x:number, y:number, width:number, height:number):reduce`

Like cairo.surface#mark_dirty(), but drawing has been done only to the specified rectangle, so that cairo can retain cached contents for other parts of the surface.

Any cached clip set on the surface will be reset by this function, to make sure that future cairo calls have the clip set that they expect.

`cairo.surface#set_device_offset(x_offset:number, y_offset:number):reduce`

Sets an offset that is added to the device coordinates determined by the CTM when drawing to surface. One use case for this function is when we want to create a cairo.surface that redirects drawing for a portion of an onscreen surface to an offscreen surface in a way that is completely invisible to the user of the cairo API. Setting a transformation via cairo.context#translate() isn't sufficient to do this, since functions like cairo.context#device_to_user() will expose the hidden offset.

Note that the offset affects drawing to the surface as well as using the surface in a source pattern.

`cairo.surface#get_device_offset()`

This function returns the previous device offset set by cairo.surface#set_device_offset().

`cairo.surface#set_fallback_resolution(x_pixels_per_inch:number, y_pixels_per_inch:number):reduce`

Set the horizontal and vertical resolution for image fallbacks.

When certain operations aren't supported natively by a backend, cairo will fallback by rendering operations to an image and then overlaying that image onto the output. For backends that are natively vector-oriented, this function can be used to set the resolution used for these image fallbacks, (larger values will result in more detailed images, but also larger file sizes).

Some examples of natively vector-oriented backends are the ps, pdf, and svg backends.

For backends that are natively raster-oriented, image fallbacks are still possible, but they are always performed at the native device resolution. So this function has no effect on those backends.

Note: The fallback resolution only takes effect at the time of completing a page (with cairo.context#show_page() or cairo.context#copy_page()) so there is currently no way to have more than one fallback resolution in effect on a single page.

The default fallback resoultion is 300 pixels per inch in both dimensions.

`cairo.surface#get_fallback_resolution()`

This function returns the previous fallback resolution set by cairo.surface#set_fallback_resolution(), or default fallback resolution if never set.

`cairo.surface#get_type()`

This function returns the type of the backend used to create a surface. See cairo_surface_type_t for available types.

`cairo.surface#copy_page():reduce`

Emits the current page for backends that support multiple pages, but doesn't clear it, so that the contents of the current page will be retained for the next page. Use cairo.surface#show_page() if you want to get an empty page after the emission.

There is a convenience function for this that takes a cairo.context, namely cairo.context#copy_page().

`cairo.surface#show_page():reduce`

Emits and clears the current page for backends that support multiple pages. Use cairo.surface#copy_page() if you don't want to clear the page.

There is a convenience function for this that takes a cairo.context, namely cairo.context#show_page().

`cairo.surface#has_show_text_glyphs()`

Returns whether the surface supports sophisticated cairo.context#show_text_glyphs() operations. That is, whether it actually uses the provided text and cluster data to a cairo.context#show_text_glyphs() call.

Note: Even if this function returns false, a cairo.context#show_text_glyphs() operation targeted at surface will still succeed. It just will act like a cairo.context#show_glyphs() operation. Users can use this function to avoid computing UTF-8 text and cluster mapping if the target surface does not use it.

`cairo.surface#set_mime_data():reduce`

`cairo.surface#get_mime_data()`

`cairo.surface#supports_mime_type()`

`cairo.surface#map_to_image(extents:cairo.rectangle_int)`

`cairo.surface#unmap_image()`

`cairo.surface#write_to_png(stream:stream:w):reduce`

10.3.3 Image Surfaces - Rendering to memory buffers

Functions

`cairo.image_surface.create(image:image):static {block?}`

`cairo.image_surface.create_from_png(stream:stream:r):static {block?}`

`cairo.image_surface#get_format()`

`cairo.image_surface#get_width()`

`cairo.image_surface#get_height()`

`cairo.image_surface#get_stride()`

10.3.4 PDF Surfaces - Rendering PDF documents

Functions

`cairo.pdf_surface.create(stream:stream:w, width_in_points:number, height_in_points:number):static {block?}`

`cairo.pdf_surface#restrict_to_version(version:number):reduce`

`cairo.pdf_surface#set_size(width_in_points:number, height_in_points:number):reduce`

10.3.5 PNG Support - Reading and writing PNG images

Functions

10.3.6 PostScript Surfaces - Rendering PostScript documents

Functions

10.3.7 Recording Surfaces - Records all drawing operations

Functions

10.3.8 Win32 Surfaces - Microsoft Windows surface support

Functions

10.3.9 SVG Surfaces - Rendering SVG documents

Functions

`cairo.svg_surface.create(stream:stream:w, width_in_points:number, height_in_points:number):static {block?}`

`cairo.svg_surface#restrict_to_version(version:number):reduce`

10.3.10 Quartz Surfaces - Rendering to Quartz surfaces

Functions

10.3.11 XCB Surfaces - X Window System rendering using the XCB library

Functions

10.3.12 XLib Surfaces - X Window System rendering using XLib

Functions

10.3.13 XLib-XRender Backend - X Window System rendering using XLib and the X Render extension

Functions

10.3.14 Script Surfaces - Rendering to replayable scripts

Functions

10.4 Utilities

Functions

10.4.1 cairo.matrix - Generic matrix operations

Functions

10.5 Thanks

This module uses Cairo library which is distributed in the following site:

http://cairographics.org/

Chapter 11

calendar Module

The `calendar` module provides a function to generate a string of calendar for the specified year.

Below is an example to print a calendar for the year 2015.

```
println(calendar.calendar(2015))
```

11.1 Module Function

`calendar.calendar(year:number, weekoffset:number => 0, ncols:number => 3)`

Prints calendars of a specified year. The argument `weekoffset` specifies from which week the calendar starts, `0` from Sunday, `1` from Monday, and so on. The argument `ncols` specifies how many months are printed in one row.

Chapter 12

cbridge Module

The `cbridge` module ...

12.1 Module Function

Chapter 13

conio Module

The `conio` module provides following measures to work on a console screen:

- Moves the cursor where texts are printed.
- Changes text colors.
- Retrieves console size.
- Waits for keyboard input.

To utilize it, import the `conio` module using `import` function.

Below is an example to print a frame around a console:

```
import(conio)

conio.clear()
[w, h] = conio.getwinsize()
conio.moveto(0, 0) {
    print('*' * w)
}
conio.moveto(0, 1 .. (h - 2)) {
    print('*', ' ' * (w - 2), '*')
}
conio.moveto(0, h - 1) {
    print('*' * w)
}
conio.waitkey():raise
```

13.1 Module Function

`conio.clear(region?:symbol):void`

Clears the screen.

In default, it clears whole the screen. Argument `region` that takes one of the symbols below would specify the region to be cleared.

- `` `line `` .. clears characters in the line where the cursor exists.
- `` `left `` .. clears characters on the left side of the cursor.

- `‘right` .. clears characters on the right side of the cursor.
- `‘top` .. clears characters on the above side of the cursor.
- `‘bottom` .. clears characters on the below side of the cursor.

`conio.getwinsize()`

Returns the screen size as a list `[width, height]`.

`conio.setcolor(fg:symbol:nil, bg?:symbol):map:void {block?}`

Sets foreground and background color of text by specifying a color symbol. Available color symbols are listed below:

- `‘black`
- `‘blue`
- `‘green`
- `‘aqua`
- `‘cyan`
- `‘red`
- `‘purple`
- `‘magenta`
- `‘yellow`
- `‘white`
- `‘gray`
- `‘bright_blue`
- `‘bright_green`
- `‘bright_aqua`
- `‘bright_cyan`
- `‘bright_red`
- `‘bright_purple`
- `‘bright_magenta`
- `‘bright_yellow`
- `‘bright_white`

If `fg` is set to nil, the foreground color remains unchanged. If `bg` is omitted or set to nil, the background color remains unchanged.

If `block` is specified, the color is changed before evaluating the block, and then gets back to what has been set when done.

`conio.moveto(x:number, y:number):map:void {block?}`

Moves cursor to the specified position. The most top-left position on the screen is represented as `0, 0`.

If `block` is specified, the cursor is moved before evaluating the block, and then gets back to where it has been when done.

`conio.waitkey():[raise]`

Waits for a keyboard input and returns a character code number associated with the key.

If `:raise` attribute is specified, hitting `Ctrl-C` issues a terminating signal that causes the program done.

Character code numbers of some of the special keys are defined as below:

- `conio.K_BACKSPACE`
- `conio.K_TAB`
- `conio.K_RETURN`
- `conio.K_ESCAPE`
- `conio.K_SPACE`
- `conio.K_UP`
- `conio.K_DOWN`
- `conio.K_RIGHT`
- `conio.K_LEFT`
- `conio.K_INSERT`
- `conio.K_HOME`
- `conio.K_END`
- `conio.K_PAGEUP`
- `conio.K_PAGEDOWN`
- `conio.K_DELETE`

Chapter 14

csv Module

The `csv` module provices measures to read/write CSV files. To utilize it, import the `csv` module using `import()` function.

Below is an example to read a CSV file that contains three fields per line:

```
import(csv)

Record = struct(name:string, age:number, email:string)
records = Record * csv.read('records.csv')
printf('name:%s, age:%d, email:%s¥n',
       records:*name, records:*age, records:*email)
```

14.1 Module Function

`csv.parse(str:string):map {block?}`

Creates an iterator that parses a text in CSV format that is contained in the specified string and returns a list of fields as its each element.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`csv.read(stream:stream:r) {block?}`

Creates an iterator that parses a text in CSV format from the specified stream and returns a list of fields as its each element.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

14.2 csv.writer Class

14.2.1 Constructor

`csv.writer(stream:stream:w, format?:string) {block?}`

Creates a `csv.writer` instance that provides methods to write CSV text to the specified stream.

The argument `format` specifies a printf-style format string that is used to convert a `number` and `complex` value to a string.

14.2.2 Method

`csv.writer#write(fields+):map:reduce`

Writes values in CSV format.

The argument `fields` takes `string`, `number` or `complex` values that are to be put out in a row.

14.3 Extension of stream Class

This module extends the `stream` class with methods described here.

`stream#read@csv() {block?}`

Creates an iterator that parses a text in CSV format from the specified stream and returns a list of fields as its each element.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.

- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`stream#writer@csv(format?:string) {block?}`

Creates a `csv.writer` instance that provides methods to write CSV text to the target stream.

The argument `format` specifies a printf-style format string that is used to convert a `number` and `complex` value to a string.

Chapter 15

curl Module

The `curl` module provices measures to access Internet resources using cURL library. To utilize it, import the `curl` module using `import` function.

15.1 Module Function

`curl.version() {block?}`

Returns a string of the libcurl version.

`curl.easy_init() {block?}`

Initializes cURL and returns a easy_handle object.

15.2 curl.easy_handle Class

`curl.easy_handle#escape(string:string):void`

`curl.easy_handle#getinfo(info:number)`

`curl.easy_handle#perform(stream?:stream:w):void`

`curl.easy_handle#recv(buflen:number)`

`curl.easy_handle#reset():void`

`curl.easy_handle#send(buffer:binary)`

`curl.easy_handle#setopt(option:number, arg):void`

`curl.easy_handle#unescape(string:string):void`

15.3 Thanks

This module uses libcurl which is distributed in the following site:

http://curl.haxx.se/libcurl/

Chapter 16

diff Module

The `diff` module provices measures to detect differences between texts. To utilize it, import the `diff` module using `import` function.

Below is an example to show differences between files `file1.txt` and `file2.txt`:

```
diff.compose(stream('file1.txt'), stream('file2.txt')).render(sys.stdout)
```

16.1 Module Function

`diff.compose(src1, src2):[icase,sync] {block?}`

Extracts differences between two sets of line sequence and returns `diff.diff@line` instance that contains the difference information.

You can specify a value of `string`, `stream`, `iterator` or `list` for the argument `src1` and `src2`. In the result, the content of `src1` is referred to as an "original" one and that of `src2` as a "new" one.

Below is an example to compare between two strings:

```
str1 = '...'
str2 = '...'
result = diff.compose(str1, str2)
```

Below is an example to compare between two files:

```
file1 = stream('file1.txt')
file2 = stream('file2.txt')
result = diff.compose(file1, file2)
```

Below is an example to compare between two iterators:

```
chars1 = '...'.each()
chars2 = '...'.each()
result = diff.compose(chars1, chars2)
```

Below is an example to compare between a file and a string:

```
file = stream('file.txt')
str = '...'
result = diff.compose(file, str)
```

If `block` is specified, it would be evaluated with a block parameter `|d:diff.diff@line|`, where `d` is the created instance. In this case, the block's result would become the function's returned value.

If attribute `:icase` is specified, it wouldn't distinguish upper and lower case of characters.

`diff.compose@char(src1:string, src2:string):[icase] {block?}`

Extracts differences between two strings and returns `diff.diff@line` instance that contains the difference information.

If `block` is specified, it would be evaluated with a block parameter `|d:diff.diff@char|`, where `d` is the created instance. In this case, the block's result would become the function's returned value.

If attribute `:icase` is specified, it wouldn't distinguish upper and lower case of characters.

16.2 diff.diff@line Class

The `diff.diff@line` instance is created by function `diff.compose()` and provides information about differences between two texts by lines.

16.2.1 Property

Property	Type	R/W	Explanation
`distance`	`number`	R	The distance between the texts. Zero means that they are identical each other.
`edits`	`iterator`	R	An iterator that returns `diff.edit@line` instances stored in the result.
`nlines@org`	`number`	R	Number of lines in the "original" text.
`nlines@new`	`number`	R	Number of lines in the "new" text.

16.2.2 Method

`diff.diff@line#eachhunk(format?:symbol, lines?:number) {block?}`

Creates an iterator that returns `diff.hunk@line` instance stored in the result.

The argument `format` takes one of the symbols that specifies the hunk format:

- `` `normal `` .. Normal format (not supported yet).
- `` `context `` .. Context format (not supported yet).
- `` `unified `` .. Unified format. This is the default.

The argument `lines` specifies a number of common lines appended before and after different lines

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`diff.diff@line#render(out?:stream:w, format?:symbol, lines?:number) {block?}`

Renders diff result to the specified stream.

If the argument `out` is omitted, this method returns a string of the rendered text. Otherwise, it returns `nil`.

The argument `format` takes one of the symbols that specifies the rendering format:

- `‘normal` .. Normal format (not supported yet).
- `‘context` .. Context format (not supported yet).
- `‘unified` .. Unified format. This is the default.

The argument `lines` specifies a number of common lines appended before and after different lines.

16.3 diff.hunk@line Class

The `diff.hunk@line` instance provides information about a hunk.

16.3.1 Property

Property	Type	R/W	Explanation
`edits`	`iterator`	R	An iterator that returns `diff.edit@line` instances stored in the hunk.
`lineno@org`	`number`	R	Top line number of the "original" text covered by the hunk.
`lineno@new`	`number`	R	Top line number of the "new" text covered by the hunk.
`nlines@org`	`number`	R	Number of lines in the "original" text covered by the hunk.
`nlines@new`	`number`	R	Number of lines in the "new" text covered by the hunk.

16.3.2 Method

diff.hunk@line#print(out?:stream):void {block?}

Prints the content of the `diff.hunk` instance to the specified stream.

16.4 diff.edit@line Class

The `diff.edit@line` provides information about an edit operation.

16.4.1 Property

Property	Type	R/W	Explanation
diff.edit@line#type	symbol	R	Edit operation: `'copy` .. Copy the line. `'add` .. Add the line. `'delete` .. Delete the line.
mark	string	R	A mark string that appears on the top of each line in Unified format.
lineno@org	number	R	Line number of the "original" text correspond to the edit.
lineno@new	number	R	Lop line number of the "new" text correspond to the edit.
source	string	R	A source text.
unified	string	R	A composed string in Unified format.

16.4.2 Method

diff.edit@line#print(out?:stream):void {block?}

Prints the content of the `diff.edit` instance to the specified stream.

16.5 diff.diff@char Class

The `diff.diff@char` instance is created by function `diff.compose@char()` and provides information about differences between two texts by characters.

16.5.1 Property

Property	Type	R/W	Explanation
distance	number	R	The distance between the texts. Zero means that they are identical each other.
edits	iterator	R	An iterator that returns `diff.edit@char` instances stored in the result.
edits@org	iterator	R	An iterator that returns `diff.edit@char` instances that are applied to the "original" string.
edits@new	iterator	R	An iterator that returns `diff.edit@char` instances that are applied to the "new" string.

16.6 diff.edit@char Class

The `diff.edit@char` provides information about an edit operation.

16.6.1 Property

Property	Type	R/W	Explanation
`diff.edit@char#type`	`symbol`	R	Edit operation: `` `copy `` .. Copy the line. `` `add `` .. Add the line. `` `delete `` .. Delete the line.
`diff.edit@char#mark`	`string`	R	A mark string that appears on the top of each line in Unified format.
`diff.edit@char#source`	`string`	R	A source text.

16.7 Thanks

This module uses dtl (Diff Template Library) which is distributed in the following site:

https://code.google.com/p/dtl-cpp/

Chapter 17

doxygen Module

The doxygen module provides measures to parse a document written in Doxygen syntax. To utilize it, import the doxygen module using import function.

```
+----------+  1.. +-----------+  1.. +------+
| document *------| structure *------| elem |
+----------+      +-----------+      +------+

+---------------+  1 +---------+
| configuration *----| aliases |
+---------------+    +---------+

+----------+     +-------------------+
| renderer |<----| specific_renderer |
+----------+     +-------------------+
```

17.1 doxygen.document Class

17.1.1 Constructor

`doxygen.document(stream?:stream, aliases?:doxygen.aliases, extracted?:boolean) {block?}`

Reads a Doxygen document from `stream` and creates an instance of `doxygen.document` class.

The argument `aliases` is an instance that is available as a member of `doxygen.configuration` instance and contains information about command aliases, or custom commands in the other word.

In default, the parser expects the Doxygen document is written within C-style comments and extracts the document body from them before parsing. If the argument `extracted` is set to `true`, it exepcts the document already have been extracted from the comments.

If `block` is specified, it would be evaluated with a block parameter `|doc:doxygen.document|`, where `doc` is the created instance. In this case, the block's result would become the function's returned value.

17.1.2 Method

`doxygen.document#structures() {block?}`

Creates an iterator that returns instances of `doxygen.structure` contained in the `doxygen.document`.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

17.2 doxygen.structure Class

17.2.1 Property

Property	Type	R/W	Explanation
`aftermember`	`boolean`	R	

17.2.2 Method

`doxygen.structure#elems():map {block?}`

Creates an iterator that returns `doxygen.elem` instances of all the elements contained in the structure.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`doxygen.structure#substructures() {block?}`

Creates an iterator that returns `doxygen.structure` instances of sub structures contained in the structure.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

17.3 doxygen.elem Class

17.3.1 Method

`doxygen.elem#print(indent?:number, out?:stream):map:void`

Prints out the content of the element to `out` with an indentation level specified by `indent` that starts from zero. If `out` is omitted, the result would be put out to standard output.

`doxygen.elem#render(renderer:doxygen.renderer):void`

Renders the element content using `doxygen.renderer`.

17.4 doxygen.configuration Class

17.4.1 Property

Property	Type	R/W	Explanation
aliases	doxygen.aliases	R	

17.4.2 Constructor

`doxygen.configuration(stream?:stream) {block?}`

Reads a configuration file, which is usually dubbed "Doxyfile", from `stream` and creates a `doxygen.configuration` instance.

If `block` is specified, it would be evaluated with a block parameter `|cfg:doxygen.configuration|`, where `cfg` is the created instance. In this case, the block's result would become the function's returned value.

17.4.3 Method

`doxygen.configuration#get(tagname:string):map:[raise]`

Returns a value associated with the tag specified by the argument `tagname`.

If the specified tag is not found, the method would return `nil` while it would cause an error in the case the attribute `:raise` is specified.

`doxygen.configuration#print(out?:stream):map:void`

Prints out the content of the configuration to `out`. If omitted, the result would be put out to standard output.

17.5 doxygen.aliases Class

17.5.1 Method

`doxygen.aliases#print(out?:stream):map:void`

Prints out definitions of aliases to the stream `out`. If the argument is omitted, the result would be put out to the standard output.

17.6 doxygen.renderer Class

17.6.1 Constructor

`doxygen.renderer(out:stream, cfg:doxygen.configuration) {block?}`

Creates a `doxygen.renderer` instance.

If `block` is specified, it would be evaluated with a block parameter `|renderer:doxygen.renderer|`, where `renderer` is the created instance. In this case, the block's result would become the function's returned value.

Chapter 18

example Module

The `example` module is just an example that is supposed to be referenced as a skeleton when you want to create a new module.

Chapter 19

freetype Module

The `freetype` module provices measures to access vectorized font data using freetype library. To utilize it, import the `freetype` module using `import` function.

19.1 Module Function

`freetype.sysfontpath(name:string):map`

19.2 freetype.BBox Class

19.3 freetype.BDF_Property Class

19.4 freetype.Bitmap Class

19.4.1 Method

`freetype.Bitmap#Embolden(strength:number):reduce`

19.5 freetype.CharMap Class

19.5.1 Method

`freetype.CharMap#Get_Index()`

19.6 freetype.FTC_CMapCache Class

19.7 freetype.FTC_ImageCache Class

19.8 freetype.FTC_ImageType Class

19.9 freetype.FTC_Manager Class

19.10 freetype.FTC_Node Class

19.11 freetype.FTC_SBit Class

19.12 freetype.FTC_SBitCache Class

19.13 freetype.FTC_Scaler Class

19.14 freetype.Face Class

19.14.1 Constructor

`freetype.Face(stream:stream, face_index:number => 0):map {block?}`

19.14.2 Method

`freetype.Face#CheckTrueTypePatents()`

`freetype.Face#Get_Advance(glyph_index:number, load_flags:number)`

`freetype.Face#Get_Advances(glyph_index_start:number, count:number, load_flags:number)`

`freetype.Face#Get_Glyph_Name(glyph_index:number)`

`freetype.Face#Get_Postscript_Name()`

`freetype.Face#Get_Kerning(left_glyph:number, right_glyph:number, kern_mode:number)`

`freetype.Face#Load_Char(char_code:number, load_flags:number):reduce`

`freetype.Face#Load_Glyph(glyph_index:number, load_flags:number):reduce`

`freetype.Face#Set_Charmap(charmap:freetype.CharMap):reduce`

`freetype.Face#Set_Pixel_Sizes(pixel_width:number, pixel_height:number):reduce`

19.15 freetype.Glyph Class

19.15.1 Method

freetype.Glyph#Copy()

freetype.Glyph#Stroke(stroker:freetype.Stroker):reduce

freetype.Glyph#StrokeBorder(stroker:freetype.Stroker, inside:boolean):reduce

19.16 freetype.GlyphSlot Class

19.16.1 Method

freetype.GlyphSlot#Get_Glyph()

freetype.GlyphSlot#Render(render_mode:number):reduce

19.17 freetype.Matrix Class

19.17.1 Constructor

freetype.Matrix(array:array@double):map {block?}

19.17.2 Method

freetype.Matrix#Multiply(matrix:freetype.Matrix):reduce

freetype.Matrix#Invert():reduce

19.18 freetype.Outline Class

19.18.1 Method

freetype.Outline#Translate(xOffset:freetype.Matrix, yOffset:freetype.Matrix):reduce

freetype.Outline#Transform(matrix:freetype.Matrix):reduce

freetype.Outline#Embolden(strength:number):reduce

freetype.Outline#Reverse():reduce

19.19 freetype.Raster Class

19.20 freetype.Span Class

19.21 freetype.Stroker Class

19.21.1 Constructor

`freetype.Stroker():map {block?}`

19.21.2 Method

`freetype.Stroker#BeginSubPath(to:freetype.Vector, open:boolean):reduce`

19.22 freetype.Vector Class

19.22.1 Constructor

`freetype.Vector(x:number, y:number):map {block?}`

19.22.2 Method

`freetype.Vector#Length()`

`freetype.Vector#Transform(matrix:freetype.Matrix):reduce`

19.23 freetype.font Class

19.24 Constructor

`freetype.font(face:freetype.Face):map {block?}`

19.24.1 Method

`freetype.font#cleardeco():reduce`

`freetype.font#drawtext(image:image, x:number, y:number, str:string):map:reduce {block?}`

Draws a text on the image.

`freetype.font#calcsize(str:string):map`

`freetype.font#calcbbox(x:number, y:number, str:string):map`

19.25 Extension to image Class

This module extends the `image` class with methods described here.

`image#drawtext(font:freetype.font, x:number, y:number, str:string):map:reduce {block?}`

Draws a text on the image.

19.26 Thanks

This module uses FreeType library which is distributed in the following site:

http://www.freetype.org/

Chapter 20

fs Module

The `fs` module provides measures to access and modify information in file systems. This is a built-in module, so you can use it without being imported.

20.1 Module Function

`fs.chdir(pathname:string) {block?}`

Changes the current working directory to `pathname`.

The block would be evaluated if specified, and the working directory would be changed only during that evaluation period.

`fs.chmod(mode, pathname:string):map:void:[follow_link]`

Changes the access mode of a file specified by `pathname`.

There are two formats to specify the mode: one is by a number, and another in a string.

When specified in a number, following bits are associated with access permissions:

- `b8 b7 b6` .. Read, write and executable permissions for owners
- `b5 b4 b3` .. Read, write and executable permissions for groups
- `b2 b1 b0` .. Read, write and executable permissions for others

When set to one, each permission is validated.

When specified in a string, it accepts a permission directive in a format of following regular expression

```
[ugoa]+([-+=][rwx]+)+
```

It starts with characters that represent target which permissions are modified as described below:

- `u` .. owners
- `g` .. groups
- `o` .. others

- `a` .. all users

Then, follows an operation:

- `-` .. remove
- `+` .. append
- `=` .. set

At last, permission attributes are specified as below:

- `r` .. read permission
- `w` .. write permission
- `x` .. executable permission

If the modification target is a link file, each platform would have different result:

- Linux .. Modifies permissions of the link file itself. Specifying `:follow_link` attribute would modify permsisions of the target file instead.
- MacOS .. Modifies permissions of the target file. Attribute `:follow_link` has no effect.
- Windows .. Modifies permissions of the link file. Attribute `:follow_link` has no effect.

`fs.copy(src:string, dst:string):map:void:[overwrite]`

Copies a file.

An argument `src` needs to specify a path name of a file that is to be copied while `dst` can specify a path name of either a file or a directory. If `dst` is a directory, the file would be copied into that. Otherwise, it would create a copy of `src` that has a name specified by `dst`.

If a destination file already exists, an error occurs. Specifying an attribute `:overwrite` would overwrite an existing one.

`fs.cpdir(src:string, dst:string):map:void:[tree]`

Copies a directory.

Arguments `src` and `dst` specify source directory and destination directory respectively. In default, sub directories are not copied.Specifying `:tree` attribute would copy all the sub directories in the source.

`fs.getcwd()`

Returns the current working directory.

`fs.mkdir(pathname:string):map:void:[tree]`

Creates a directory.

If `pathname` consists of multiple sub directories and some of them still doesn't exist, an error occurs. Specifying `:tree` attribute would create such directories.

`fs.remove(pathname:string):map:void`

Removes a file from the file system.

`fs.rename(src:string, dst:string):map:void`

Renames a file or directory.

`fs.rmdir(pathname:string):map:void:[tree]`

Removes a directory.

If the directory contains sub directories, an error occurs. Specifying `:tree` attribute would delete such a directory.

20.2 fs.stat Class

An instance of `fs.stat` class contains information about a file or directory on the file system, which includes its full path name, size, creation time and file attributes. A `stream` instance has a property named `stat` that is a `fs.stat` instance when it comes from a file or directory in a file system. You can also get the instance using `fs.stat()` function.

20.2.1 Constructor

`fs.stat(pathname:string) {block?}`

20.2.2 Property

A `fs.stat` instance has the following properties:

Property	Type	R/W	Explanation
pathname	string	R	
dirname	string	R	
filename	string	R	
size	number	R	
uid	number	R	
gid	number	R	
atime	datetime	R	
mtime	datetime	R	
ctime	datetime	R	
isdir	boolean	R	
ischr	boolean	R	
isblk	boolean	R	
isreg	boolean	R	
isfifo	boolean	R	
islnk	boolean	R	
issock	boolean	R	

Chapter 21

gif Module

The `gif` module provides measures to read/write image data in GIF format. To utilize it, import the `gif` module using `import` function.

Below is an example to read a GIF file:

```
import(gif)
img = image('foo.gif')
```

Below is an example to create a GIF file that contains multiple images:

```
import(gif)
g = gif.content()
g.addimage(['cell1.png', 'cell2.png', 'cell3.png'], 10) g.write('anim.gif')
```

21.1 Exntension to Function's Capability

This module extends the capability of function `image()` and instance method `image#write()` so that they can read/write GIF files.

When function `image()` is provided with a stream that satisfies the following conditions, it would recognize the stream as a GIF file.

- The identifier of the stream ends with a suffix ".gif".
- The stream data begins with a byte sequence "GIF87a" or "GIF89a".

When instance method `image#write()` is provided with a stream that satisfies the following condition, it would write image data in GIF format.

- The identifier of the stream ends with a suffix ".gif".

21.2 gif.content Class

The `gif.content` class provides properties to explain GIF information and methods to manipulate contents of GIF file. Below is a class diagram of `gif.content`:

```
+-------------+                         +-----------------------------+
| gif.content |images                   |            image            |
|-------------*-------------------------+-----------------------------|
|             |                     1.. +-----------------------------+
|             |
|             |                         +-----------------------------+
|             |Header                   |         gif.Header          |
|             *-------------------------+-----------------------------|
|             |                         +-----------------------------+
|             |
|             |                         +-----------------------------+
|             |LogicalScreenDescriptor  | gif.LogicalScreenDescriptor |
|             *-------------------------+-----------------------------|
|             |                         +-----------------------------+
|             |
|             |                         +-----------------------------+
|             |CommentExtension         |    gif.CommentExtension     |
|             *-------------------------+-----------------------------|
|             |                         +-----------------------------+
|             |
|             |                         +-----------------------------+
|             |PlainTextExntension      |   gif.PlainTextExtension    |
|             *-------------------------+-----------------------------|
|             |                         +-----------------------------+
|             |
|             |                         +-----------------------------+
|             |ApplicationExntension    | gif.ApplicationExtension    |
|             *-------------------------+-----------------------------|
|             |                         +-----------------------------+
+-------------+
```

- A property of `gif.content` has one or more images. Multiple images are mainly used for animation.
- The property named `Header` is an instance of `gif.Header` class.
- The property named `LogicalScreenDescriptor` is an instance of `gif.LogicalScreenDescriptor` class.
- The property named `CommentExtension` is an instance of `gif.CommentExtension` class.
- The property named `PlainTextExtension` is an instance of `gif.PlainTextExtension` class.
- The property named `ApplicationExtension` is an instance of `gif.ApplicationExtension` class.

21.2.1 Constructor

`gif.content(stream?:stream:r, format:symbol => 'rgba) {block?}`

Reads a GIF data from a stream and returns an object that contains GIF related information and images of a specified format. format is is `rgb`,rgba or `noimage`. `If`noimage is specified, only the information data is read

21.2.2 Property

A `gif.content` instance has the following properties:

Property	Type	R/W	Explanation
images	image[]	R	
Header	gif.Header	R	
LogicalScreenDescriptor	gif.LogicalScreenDescriptor	R	
CommentExtension	gif.CommentExtension	R	
PlainTextExtension	gif.PlainTextExtension	R	
ApplicationExtension	gif.ApplicationExtension	R	

21.2.3 Method

`gif.content#addimage(image:image, delayTime:number => 10, leftPos:number => 0, topPos:number => 0, disposa`

Adds an image to GIF information.

You can add multiple images that can be played as a motion picture.

The argument `delayTime` specifies the delay time in 10 milli seconds between images.

The arguments `leftPost` and `topPos` specifies the rendered offset in the screen.

The argument `disposalMethod` takes one of following symbols that specifies how the image will be treated after being rendered.

- ‘`none` ..
- ‘`keep` ..
- ‘`background`..
- ‘`previous` ..

This method returns the reference to the target instance itself.

`gif.content#write(stream:stream:w):reduce`

Writes a GIF image to a stream.

This method returns the reference to the target instance itself.

21.3 gif.Header Class

A `gif.Header` instance provides information of Header structure in GIF format.

21.3.1 Property

A `gif.Header` instance has the following properties:

Property	Type	R/W	Explanation
Signature	binary	R	
Version	binary	R	

21.4 gif.LogicalScreenDescriptor Class

A `gif.LogicalScreenDescriptor` instance provides information of Logical Screen Descriptor structure in GIF format.

21.4.1 Property

A `gif.LogicalScreenDescriptor` instance has the following properties:

Property	Type	R/W	Explanation
`LogicalScreenWidth`	`number`	R	
`LogicalScreenHeight`	`number`	R	
`GlobalColorTableFlag`	`boolean`	R	
`ColorResolution`	`number`	R	
`SortFlag`	`boolean`	R	
`SizeOfGlobalColorTable`	`number`	R	
`BackgroundColorIndex`	`number`	R	
`BackgroundColor`	`color`	R	
`PixelAspectRatio`	`number`	R	

21.5 gif.CommentExtension Class

A `gif.CommentExtnsion` instance provides information of Comment Extension structure in GIF format.

21.5.1 Property

A `gif.CommentExtension` instance has the following properties:

Property	Type	R/W	Explanation
`CommentData`	`binary`	R	

21.6 gif.PlainTextExtension Class

A `gif.PlainTextExtnsion` instance provides information of Plain Text Extension structure in GIF format.

21.6.1 Property

A `gif.PlainTextExtension` instance has the following properties:

Property	Type	R/W	Explanation
`TextGridLeftPosition`	`number`	R	
`TextGridTopPosition`	`number`	R	
`TextGridWidth`	`number`	R	
`TextGridHeight`	`number`	R	
`CharacterCellWidth`	`number`	R	
`CharacterCellHeight`	`number`	R	
`TextForegroundColorIndex`	`number`	R	
`TextBackgroundColorIndex`	`number`	R	
`PlainTextData`	`binary`	R	

21.7 gif.ApplicationExtension Class

A `gif.ApplicationExtnsion` instance provides information of Application Extension structure in GIF format.

21.7.1 Property

A `gif.ApplicationExtension` instance has the following properties:

Property	Type	R/W	Explanation
`ApplicationIdentifier`	`binary`	R	
`AuthenticationCode`	`binary`	R	
`ApplicationData`	`binary`	R	

21.8 gif.GraphicControl Class

A `gif.GraphicControl` instance provides information of Graphi Control Extension structure in GIF format.

21.8.1 Property

A `gif.GraphicControl` instance has the following properties:

Property	Type	R/W	Explanation
`DisposalMethod`	`symbol`	R	
`UserInputFlag`	`boolean`	R	
`TransparentColorFlag`	`boolean`	R	
`DelayTime`	`number`	R	
`TransparentColorIndex`	`number`	R	

21.9 gif.ImageDescriptor Class

A `gif.ImageDescriptor` instance provides information of Image Descriptor structure in GIF format.

21.9.1 Property

A `gif.ImageDescriptor` instance has the following properties:

Property	Type	R/W	Explanation
`ImageLeftPosition`	`number`	R	
`ImageTopPosition`	`number`	R	
`ImageWidth`	`number`	R	
`ImageHeight`	`number`	R	
`LocalColorTableFlag`	`boolean`	R	
`InterlaceFlag`	`boolean`	R	
`SortFlag`	`boolean`	R	
`SizeOfLocalColorTable`	`number`	R	

21.10 gif.imgprop Class

Below is a class diagram of `gif.imgprop`:

```
+-------------+       +-------------+                  +---------------------+
|    image    |gif    | gif.imgprop |GraphicControl    | gif.GraphicControl  |
|-------------*--------+-------------*------------------+---------------------|
+-------------+       |             |                  +---------------------+
                      |             |
                      |             |                  +---------------------+
                      |             |ImageDescriptor   | gif.ImageDescriptor |
                      |             *------------------+---------------------|
                      |             |                  +---------------------+
                      +-------------+
```

- An `image` instance that the gif module creates from GIF file holds a `gif.imgprop` instance as its property that is named `gif`.
- The property named `GraphicControl` is an instance of `gif.GraphiControl` class.
- The property named `ImageDescriptor` is an instance of `gif.ImageDescriptor` class.

21.10.1 Property

A `gif.imgprop` instance has the following properties:

Property	Type	R/W	Explanation
`GraphicControl`	`gif.GraphicControl`	R	
`ImageDescriptor`	`gif.ImageDescriptor`	R	

21.11 Extension to image Class

This module extends the `stream` class with methods described here.

`image#read@gif(stream:stream:r):reduce`

Reads a GIF image from a stream.

This method returns the reference to the target instance itself.

`image#write@gif(stream:stream:w):reduce`

Writes a GIF image to a stream.

This method returns the reference to the target instance itself.

Property	Type	R/W	Explanation
`gif`	`gif.imgprop`	R	

Chapter 22

glu Module

The glu module provides functions of GLU library.

22.1 Module Function

glu.gluBeginCurve(nurb:glu.Nurbs):void {block?}

glu.gluBeginPolygon(tess:glu.Tesselator):void {block?}

glu.gluBeginSurface(nurb:glu.Nurbs):void {block?}

glu.gluBeginTrim(nurb:glu.Nurbs):void {block?}

glu.gluBuild1DMipmaps(target:number, internalFormat:number, width:number, format:number, type:number, data

glu.gluBuild1DMipmapsFromImage(target:number, internalFormat:number, image:image)

glu.gluBuild2DMipmaps(target:number, internalFormat:number, width:number, height:number, format:number, ty

glu.gluBuild2DMipmapsFromImage(target:number, internalFormat:number, image:image)

glu.gluCylinder(quad:glu.Quadric, base:number, top:number, height:number, slices:number, stacks:number):vo

glu.gluDeleteNurbsRenderer(nurb:glu.Nurbs):void

glu.gluDeleteQuadric(quad:glu.Quadric):void

glu.gluDeleteTess(tess:glu.Tesselator):void

glu.gluDisk(quad:glu.Quadric, inner:number, outer:number, slices:number, loops:number):void

glu.gluEndCurve(nurb:glu.Nurbs):void

glu.gluEndPolygon(tess:glu.Tesselator):void

glu.gluEndSurface(nurb:glu.Nurbs):void

glu.gluEndTrim(nurb:glu.Nurbs):void

glu.gluErrorString(error:number)

glu.gluGetNurbsProperty(nurb:glu.Nurbs, property:number, data:array@float:nomap):void

glu.gluGetString(name:number)

glu.gluGetTessProperty(tess:glu.Tesselator, which:number, data:array@double:nomap):void

glu.gluLoadSamplingMatrices(nurb:glu.Nurbs, model:array@float:nomap, perspective:array@float:nomap, view:a

glu.gluLookAt(eyeX:number, eyeY:number, eyeZ:number, centerX:number, centerY:number, centerZ:number, upX:n

glu.gluNewNurbsRenderer()

glu.gluNewQuadric()

glu.gluNewTess()

glu.gluNextContour(tess:glu.Tesselator, type:number):void

glu.gluNurbsCallback(nurbs:glu.Nurbs, which:number, func:function)

glu.gluNurbsCallbackData(nurb:glu.Nurbs, userData):void

glu.gluNurbsCallbackDataEXT(nurb:glu.Nurbs, userData):void

glu.gluNurbsCurve(nurb:glu.Nurbs, knots:array@float:nomap, stride:number, control:array@float:nomap, order

glu.gluNurbsProperty(nurb:glu.Nurbs, property:number, value:number):void

glu.gluNurbsSurface(nurb:glu.Nurbs, sKnots:array@float:nomap, tKnots:array@float:nomap, sStride:number, tS

glu.gluOrtho2D(left:number, right:number, bottom:number, top:number):void

glu.gluPartialDisk(quad:glu.Quadric, inner:number, outer:number, slices:number, loops:number, start:number

glu.gluPerspective(fovy:number, aspect:number, zNear:number, zFar:number):void

glu.gluPickMatrix(x:number, y:number, delX:number, delY:number, viewport:array@int32:nomap):void

glu.gluProject(objX:number, objY:number, objZ:number, model:array@double:nomap, proj:array@double:nomap, v

glu.gluPwlCurve(nurb:glu.Nurbs, data:array@float:nomap, stride:number, type:number):void

glu.gluQuadricCallback(quad:glu.Quadric, which:number, func:function:nil):void

glu.gluQuadricDrawStyle(quad:glu.Quadric, draw:number):void

glu.gluQuadricNormals(quad:glu.Quadric, normal:number):void

glu.gluQuadricOrientation(quad:glu.Quadric, orientation:number):void glu.gluQuadricTexture(quad:glu.Quadric, texture:boolean):void

glu.gluScaleImage(imageIn:image, wOut:number, hOut:number)

glu.gluSphere(quad:glu.Quadric, radius:number, slices:number, stacks:number):void

glu.gluTessBeginContour(tess:glu.Tesselator):void {block?}

glu.gluTessBeginPolygon(tess:glu.Tesselator, polygon_data):void {block?}

glu.gluTessCallback(tess:glu.Tesselator, which:number, func:function):void

glu.gluTessEndContour(tess:glu.Tesselator):void

glu.gluTessEndPolygon(tess:glu.Tesselator):void

glu.gluTessNormal(tess:glu.Tesselator, valueX:number, valueY:number, valueZ:number):void

glu.gluTessProperty(tess:glu.Tesselator, which:number, data:number):void

glu.gluTessVertex(tess:glu.Tesselator, location:array@double:nomap, vertex_data):void

glu.gluUnProject(winX:number, winY:number, winZ:number, model:array@double:nomap, proj:array@double:nomap,

Chapter 23

glut Module

The `glut` module provides functions of GLUT library.

23.1 Module Function

`glut.glutInit(argv[]:string) {block?}`

`glutInit` is used to initialize the GLUT library.

`glut.glutInitDisplayMode(mode:number):map:void`

`glutInitDisplayMode` sets the *initial display mode*.

`glut.glutInitDisplayString(string:string):map:void`

`glut.glutInitWindowPosition(x:number, y:number):map:void`

`glutInitWindowPosition` sets the initial window position.

`glut.glutInitWindowSize(width:number, height:number):map:void`

`glutInitWindowSize` sets the initial window size.

`glut.glutMainLoop():void`

`glutMainLoop` enters the GLUT event processing loop.

`glut.glutCreateWindow(title:string):map {block?}`

`glutCreateWindow` creates a top-level window.

`glut.glutCreateSubWindow(win:number, x:number, y:number, width:number, height:number):map {block?}`

`glutCreateSubWindow` creates a subwindow.

`glut.glutDestroyWindow(win:number):map:void`

`glutDestroyWindow` destroys the specified window.

`glut.glutPostRedisplay():void`

`glutPostRedisplay marks the *current window* as needing to be redisplayed.`

`glut.glutPostWindowRedisplay(win:number):map:void`

`glut.glutSwapBuffers():void`

`glutSwapBuffers` swaps the buffers of the *current window* if double buffered.

`glut.glutGetWindow() {block?}`

`glutGetWindow` returns the identifier of the *current window*.

`glut.glutSetWindow(win:number):map:void`

`glutSetWindow` sets the *current window*.

`glut.glutSetWindowTitle(title:string):map:void`

`glutSetWindowTitle` changes the window title of the current top-level window.

`glut.glutSetIconTitle(title:string):map:void`

`glutSetIconTitle` changes the icon title of the current top-level window.

`glut.glutPositionWindow(x:number, y:number):map:void`

`glutPositionWindow` requests a change to the position of the *current window*.

`glut.glutReshapeWindow(width:number, height:number):map:void`

`glutReshapeWindow` requests a change to the size of the *current window*.

`glut.glutPopWindow():void`

`glut.glutPushWindow():void`

`glut.glutIconifyWindow():void`

`glut.glutShowWindow():void`

`glut.glutHideWindow():void`

`glut.glutFullScreen():void`

`glut.glutSetCursor(cursor:number):map:void`

`glut.glutWarpPointer(x:number, y:number):map:void`

`glut.glutEstablishOverlay():void`

`glut.glutRemoveOverlay():void`

`glut.glutUseLayer(layer:number):map:void`

`glut.glutPostOverlayRedisplay():void`

`glut.glutPostWindowOverlayRedisplay(win:number):map:void`

`glut.glutShowOverlay():void`

glut.glutHideOverlay():void

glut.glutCreateMenu(func:function) {block?}

glut.glutDestroyMenu(menu:number):map:void

glut.glutGetMenu() {block?}

glut.glutSetMenu(menu:number):map:void

glut.glutAddMenuEntry(label:string, value:number):map:void

glut.glutAddSubMenu(label:string, submenu:number):map:void

glut.glutChangeToMenuEntry(item:number, label:string, value:number):map:void

glut.glutChangeToSubMenu(item:number, label:string, submenu:number):map:void

glut.glutRemoveMenuItem(item:number):map:void

glut.glutAttachMenu(button:number):map:void

glut.glutDetachMenu(button:number):map:void

glut.glutDisplayFunc(func:function:nil):void

glut.glutReshapeFunc(func:function:nil):void

glut.glutKeyboardFunc(func:function:nil):void

glut.glutMouseFunc(func:function:nil):void

glut.glutMotionFunc(func:function:nil):void

glut.glutPassiveMotionFunc(func:function:nil):void

glut.glutEntryFunc(func:function:nil):void

glut.glutVisibilityFunc(func:function:nil):void

glut.glutIdleFunc(func:function:nil):void

glut.glutTimerFunc(millis:number, func:function:nil, value:number):void

glut.glutMenuStateFunc(func:function:nil):void

glut.glutSpecialFunc(func:function:nil):void

glut.glutSpaceballMotionFunc(func:function:nil):void

glut.glutSpaceballRotateFunc(func:function:nil):void

glut.glutSpaceballButtonFunc(func:function:nil):void glut.glutButtonBoxFunc(func:function:nil):void

glut.glutDialsFunc(func:function:nil):void

glut.glutTabletMotionFunc(func:function:nil):void

glut.glutTabletButtonFunc(func:function:nil):void

glut.glutMenuStatusFunc(func:function:nil):void

glut.glutOverlayDisplayFunc(func:function:nil):void

glut.glutWindowStatusFunc(func:function:nil):void

glut.glutKeyboardUpFunc(func:function:nil):void

glut.glutSpecialUpFunc(func:function:nil):void

glut.glutJoystickFunc(func:function:nil, pollInterval:number):void

glut.glutSetColor(ndx:number, red:number, green:number, blue:number):void

glut.glutGetColor(ndx:number, component:number):map {block?}

glut.glutCopyColormap(win:number):map:void

glut.glutGet(type:number):map {block?}

glut.glutDeviceGet(type:number):map {block?}

glut.glutExtensionSupported(name:string):map {block?}

glut.glutGetModifiers() {block?}

glut.glutLayerGet(type:number):map {block?}

glut.glutGetProcAddress(procName:string):map:void {block?}

glut.glutBitmapCharacter(font:glut.Font, character:number):map:void

glut.glutBitmapWidth(font:glut.Font, character:number):map {block?}

glut.glutStrokeCharacter(font:glut.Font, character:number):map:void

glut.glutStrokeWidth(font:glut.Font, character:number):map {block?}

glut.glutBitmapLength(font:glut.Font, string:string):map {block?}

glut.glutStrokeLength(font:glut.Font, string:string):map {block?}

glut.glutWireSphere(radius:number, slices:number, stacks:number):map:void

glut.glutSolidSphere(radius:number, slices:number, stacks:number):map:void glut.glutWireCone(base:number, height:number, slices:number, stacks:number):map:void

glut.glutSolidCone(base:number, height:number, slices:number, stacks:number):map:void

glut.glutWireCube(size:number):map:void

glut.glutSolidCube(size:number):map:void

glut.glutWireTorus(innerRadius:number, outerRadius:number, sides:number, rings:number):map:void

glut.glutSolidTorus(innerRadius:number, outerRadius:number, sides:number, rings:number):map:void

glut.glutWireDodecahedron():void

glut.glutSolidDodecahedron():void

glut.glutWireTeapot(size:number):map:void

glut.glutSolidTeapot(size:number):map:void

glut.glutWireOctahedron():void

glut.glutSolidOctahedron():void

glut.glutWireTetrahedron():void

glut.glutSolidTetrahedron():void

glut.glutWireIcosahedron():void

glut.glutSolidIcosahedron():void

glut.glutVideoResizeGet(param:number):map {block?}

glut.glutSetupVideoResizing():void

glut.glutStopVideoResizing():void

glut.glutVideoResize(x:number, y:number, width:number, height:number):map:void

glut.glutVideoPan(x:number, y:number, width:number, height:number):map:void

glut.glutReportErrors():void

glut.glutIgnoreKeyRepeat(ignore:number):map:void

glut.glutSetKeyRepeat(repeatMode:number):map:void

glut.glutForceJoystickFunc():void

glut.glutGameModeString(string:string):map:void

glut.glutEnterGameMode() {block?}

`glut.glutLeaveGameMode():void`

`glut.glutGameModeGet(mode:number):map {block?}`

23.2 Thanks

This module uses freeglut which official site is:

http://freeglut.sourceforge.net/

Chapter 24

gmp Module

The `gmp` module provides measures to calculate numbers with multiple precision using GMP library. To utilize it, import the `gmp` module using `import` function.

It expands features of operators like addition and multiplier so that they can calculate such numbers.

24.1 Operator

Following tables show values types of operands and returned value for each operator:

+x	gmp.mpz	gmp.mpq	gmp.mpf

-x	gmp.mpz	gmp.mpq	gmp.mpf

x	gmp.mpz	gmp.mpq	gmp.mpf

x + y	gmp.mpz	gmp.mpq	gmp.mpf	number	rational
gmp.mpz	gmp.mpz	gmp.mpq	gmp.mpf	gmp.mpf	gmp.mpq
gmp.mpq	gmp.mpz	gmp.mpq	gmp.mpf	gmp.mpf	gmp.mpq
gmp.mpf	gmp.mpz	gmp.mpq	gmp.mpf	gmp.mpf	gmp.mpq
number	gmp.mpz	gmp.mpq	gmp.mpf	number	rational
rational	gmp.mpz	gmp.mpq	gmp.mpf	rational	rational

x - y	gmp.mpz	gmp.mpq	gmp.mpf	number	rational
gmp.mpz					
gmp.mpq					
gmp.mpf					
number					
rational					

x * y	gmp.mpz	gmp.mpq	gmp.mpf	number	rational
gmp.mpz					
gmp.mpq					
gmp.mpf					
number					
rational					

x / y	gmp.mpz	gmp.mpq	gmp.mpf	number	rational
gmp.mpz					
gmp.mpq					
gmp.mpf					
number					
rational					

x % y	gmp.mpz	gmp.mpq	gmp.mpf	number	rational
gmp.mpz					
gmp.mpq					
gmp.mpf					
number					
rational					

`x == y; x != y; x > y; x < y; x >= y; x <= y; x <=> y`

comparator	gmp.mpz	gmp.mpq	gmp.mpf	number	rational
gmp.mpz					
gmp.mpq					
gmp.mpf					
number					
rational					

x & y	gmp.mpz	gmp.mpq	gmp.mpf	number	rational
gmp.mpz					
gmp.mpq					
gmp.mpf					
number					
rational					

x | y	gmp.mpz	gmp.mpq	gmp.mpf	number	rational
gmp.mpz					
gmp.mpq					
gmp.mpf					
number					
rational					

x ∧ y	gmp.mpz	gmp.mpq	gmp.mpf	number	rational
gmp.mpz					
gmp.mpq					
gmp.mpf					
number					
rational					

x y	gmp.mpz	gmp.mpq	gmp.mpf	number	rational
gmp.mpz					
gmp.mpq					
gmp.mpf					
number					
rational					

x y	gmp.mpz	gmp.mpq	gmp.mpf	number	rational
gmp.mpz					
gmp.mpq					
gmp.mpf					
number					
rational					

`x..; x .. y`

24.2 Module Function

`gmp.gcd(num1:gmp.mpz, num2:gmp.mpz):map`

Calculates the greatest common divisor, GCD, between `num1` and `num2` and returns the result as `gmp.mpz`.

`gmp.lcm(num1:gmp.mpz, num2:gmp.mpz):map`

Calculates the least common multiple, LCM, between `num1` and `num2` and returns the result as `gmp.mpz`.

`gmp.sqrt(num):map`

Calculates the square root of `num`.

The type of the argument `num` must be `gmp.mpz`, `gmp.mpq`, `gmp.mpf` or `number`.

24.3 gmp.mpf Class

24.3.1 Constructor

`gmp.mpf(value?, prec?:number):map {block?}`

Creates a `gmp.mpf` instance.

If the argument `value` is specified, it would be casted to `gmp.mpf`. Acceptable types for `value` are: `number`, `string`, `gmp.mpf`, `gmp.mpz` and `gmp.mpq`.

You can specify the precision of the number by the argument `prec`. If it's omitted, a default precision would be applied.

24.3.2 Method

`gmp.mpf.get_default_prec():static`

Gets the default precision for `gmp.mpf`.

`gmp.mpf.set_default_prec(prec:number):static:void`

Sets the default precision for `gmp.mpf`.

24.4 gmp.mpq Class

24.4.1 Constructor

`gmp.mpq(numer?, denom?:number):map {block?}`

Creates a `gmp.mpq` instance.

You can call this function with one of the following form.

- `gmp.mpq(numer:number)`
- `gmp.mpq(numer:number, denom:number)`
- `gmp.mpq(str:string)`
- `gmp.mpq(num:gmp.mpq)`

24.4.2 Method

`gmp.mpq#cast@mpf() {block?}`

Casts the value to `gmp.mpf`.

If `block` is specified, it would be evaluated with a block parameter `|num:gmp.mpf|`, where `num` is the created instance. In this case, the block's result would become the function's returned value.

24.5 gmp.mpz Class

24.5.1 Constructor

`gmp.mpz(value?):map {block?}`

Creates a `gmp.mpz` instance.

If the argument `value` is specified, it would be casted to `gmp.mpz`. Acceptable types for `value` are: `number`, `string`, `gmp.mpf` and `gmp.mpz`.

24.6 Extention to string Class

This module extends the `string` class with methods described here.

`string#cast@mpf(prec?:number):map`

Casts the string to `gmp.mpf`.

You can specify the precision of the number by the argument `prec`. If it's omitted, a default precision would be applied.

If `block` is specified, it would be evaluated with a block parameter `|num:gmp.mpf|`, where `num` is the created instance. In this case, the block's result would become the function's returned value.

`string#cast@mpq():map {block?}`

Casts the string to `gmp.mpq`.

If `block` is specified, it would be evaluated with a block parameter `|num:gmp.mpq|`, where `num` is the created instance. In this case, the block's result would become the function's returned value.

`string#cast@mpz(base?:number):map`

Casts the string to `gmp.mpz`.

You can specify the basement of the number format by the argument `base`. If it's omitted, the basement would be decided by the prefix described in the string such as "`0`" and "`0x`".

If `block` is specified, it would be evaluated with a block parameter `|num:gmp.mpz|`, where `num` is the created instance. In this case, the block's result would become the function's returned value.

24.7 Thanks

This module uses GMP and its forked project MPIR which are distributed in the following sites:

- https://gmplib.org
- http://www.mpir.org/

Chapter 25

gurcbuild Module

The `gurcbuild` module is prepared to help create a composite Gura file, which contains script and other data files.

The example below would create a composite Gura file named `hello.gurc` that contains three files:

```
import(gurcbuild)

gurcbuild.build(['hello.gura', 'startimg.jpg', 'README.txt'])
```

25.1 Module Function

`gurcbuild.build(pathNames[]:string, dirName?:string)`

Creates a composite Gura file from files specified by `pathNames`, which includes script and other data files. The first entry of `pathNames` must be a script file that is to be executed as a main script.

The result file would be created in the directory specified by `dirName`. If the argument is omitted, the file would be created in the current working directory.

Chapter 26

gzip Module

The `gzip` module provides measures to read/write GZIP files. To utilize it, import the `gzip` module using `import` function.

Below is an example to read data from a GZIP file and write its uncompressed data to another file.

```
import(gzip)
gzip.reader('foo.dat.gz').copyto('foo.dat')
```

Below is an example to read data from a file and write its compressed data to a GZIP file.

```
import(gzip)
gzip.writer('foo.dat.gz').copyfrom('foo.dat')
```

26.1 Module Function

`gzip.reader(stream:stream:r) {block?}`

`gzip.writer(stream:stream:w, level?:number) {block?}`

26.2 Extension to stream Class

This module extends the `stream` class with methods described here.

`stream#reader@gzip() {block?}`

`stream#writer@gzip(level?:number) {block?}`

26.3 Thanks

This module uses zlib which official site is:

http://zlib.net/

Chapter 27

hash Module

The `hash` module provides measures to calculate hash values of a data sequence in a stream. To utilize it, import the `hash` module using `import` function.

Below is an example to calculate MD5, SHA-1 and CRC32 hash values of a file named `foo.txt`.

```
import(hash)

fileName = 'foo.txt'
println('MD5: ', hash.md5(fileName).hexdigest)
println('SHA-1: ', hash.sha1(fileName).hexdigest)
println('CRC32: ', hash.crc32(fileName).hexdigest)
```

27.1 hash.accumulator Class

The `hash.accumulator` class provides measures to calculate hashed numbers including MD5, SHA-1 and CRC32.

As the class inhefits from `stream`, you can call methods of `stream` class with `hash.accumulator` instances.

27.1.1 Property

Property	Type	R/W	Explanation
digest	binary	R	Returns the hashed result as `binary`.
hexdigest	string	R	Returns the hashed result as `string` in hexadecimal format.
number	number	R	Returns the hashed result as `number`. This field is valid only for CRC32 and returns 'nil' for other hashes.

27.1.2 Constructor

`hash.md5(stream?:stream:r) {block?}`

Creates an `hash.accumulator` instance that calculates MD5 hashed value from the content of `stream`.

`hash.sha1(stream?:stream:r) {block?}`

Creates an `hash.accumulator` instance that calculates SHA1 hashed value from the content of `stream`.

`hash.crc32(stream?:stream:r) {block?}`

Creates an `hash.accumulator` instance that calculates CRC32 hashed value from the content of `stream`.

27.1.3 Method

`hash.accumulator#init():reduce`

Initializes the state of the accumulator.

`hash.accumulator#update(stream:stream:r):reduce`

Updates the accumulator with the content of `stream`.

Chapter 28

http Module

The `http` module provides measures to connect the Internet through HTTP protocol.

28.1 Module Function

Chapter 29

jpeg Module

The `jpeg` module provides measures to read/write image data in JPEG format. To utilize it, import the `jpeg` module using `import` function.

Below is an example to read a JPEG file:

```
import(jpeg)
img = image('foo.jpeg')
```

29.1 Exntension to Function's Capability

This module extends the capability of function `image()` and instance method `image#write()` so that they can read/write JPEG files.

When function `image()` is provided with a stream that satisfies the following conditions, it would recognize the stream as a JPEG file.

- The identifier of the stream ends with a suffix "`.jpeg`", "`.jpg`" or "`.jpe`".
- The stream data begins with a byte sequence "`\xff\xd8`" that means SOI (start of Image) marker in JPEG specification.

When instance method `image#write()` is provided with a stream that satisfies the following condition, it would write image data in JPEG format.

- The identifier of the stream ends with a suffix "`.jpeg`", "`.jpg`" or "`.jpe`".

29.2 jpeg.exif Class

The `jpeg.exif` class provides EXIF information in a JPEG stream.

A `jpeg.exif` instance contains `jpeg.ifd` instances as properties named `jpeg.exif#ifd0` and `jpeg.exif#ifd1` that include a list of `jpeg.tag` instances.

```
+-----------+             +----------+        +----------+
| jpeg.exif |ifd0, ifd1   | jpeg.ifd |1..     | jpeg.tag |
|-----------*-------------+----------*--------+----------|
|           |             |          |        |          |
+-----------+             +----------+        +----------+
```

29.2.1 Property

A `jpeg.exif` instance has the following properties:

Property	Type	R/W	Explanation
`endian`	`symbol`	R	The endian type: `` `big `` for big-endian and `` `little `` for little-endian.
`ifd0`	`jpeg.ifd`	R	IFD0 instance.
`ifd1`	`jpeg.ifd`	R	IFD1 instance.
`thumbnail`	`image`	R	Thumbnail image as `image` value.
`thumbnail@jpeg`	`binary`	R	Thumbnail image as JPEG binary data.

29.2.2 Constructor

`jpeg.exif(stream?:stream:r):map:[raise] {block?}`

Reads EXIF data from `stream` and creates a `jpeg.exif` instance.

If no EXIF information exists in the stream, this function returns `nil`. If the attribute `:raise` is specified, an error occurs for that case.

If `block` is specified, it would be evaluated with a block parameter `|exif:jpeg.exif|`, where `exif` is the created instance. In this case, the block's result would become the function's returned value.

29.2.3 Method

`jpeg.exif#each() {block?}`

Creates an iterator that returns `jpeg.tag` values as elements that are stored in the property `jpeg.exif#ifd0`.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

29.3 jpeg.ifd Class

29.3.1 Property

A `jpeg.ifd` instance has the following properties:

Property	Type	R/W	Explanation
`name`	`string`	R	
`symbol`	`symbol`	R	

29.3.2 Method

`jpeg.ifd#each() {block?}`

Creates an iterator that returns `jpeg.tag` values as elements that are stored in the target `jpeg.ifd` instance.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

29.4 jpeg.tag Class

29.4.1 Property

A `jpeg.tag` instance has the following properties:

Property	Type	R/W	Explanation
`id`	`number`	R	Tag ID.
`name`	`string`	R	Tag name.
`symbol`	`symbol`	R	Tag name as `symbol`.
`type`	`number`	R	Tag type.
`typename`	`string`	R	Tag type name.
`value`	`any`	R	Tag value. When the attribute `:cooked` is specified, numbers in some tags are translated to human-readable symbols.
`ifd`	`jpeg.ifd`	R	IFD instance. Valid only for tags `Exif`, `GPSInfo` and `Interoperability`.

29.5 Extension to image Class

This module extends the `image` class with methods described here.

`image#read@jpeg(stream:stream:r, size?:number):reduce:[fast,rough]`

Reads a JPEG image data from the specified `stream`.

When the argument `size` is specified, the image would be shrinked so that it is boxed within the size.

The attribute `:fast` indicates a fast but less-qualified decompression process.

The attriubte `:rough` is only valid when `size` is specified and makes the shrinked image with nearest neighbor method. Othereise, shrinking shall be done with bilinear method.

`image#write@jpeg(stream:stream:w, quality:number => 75):reduce`

Writes a JPEG image data to the specified `stream`.

The argument `quality` takes a number between 0 and 100 with which a a higher number results in a higher quality of result but a less compression performance. The default value for it is 75.

29.6 Thanks

This module uses JPEG library which is distributed in the following site:

http://www.ijg.org/

Chapter 30

lexer Module

The `lexer` module provices functions that parse souces to generate tokens. This is a built-in module, so you can use it without being imported.

30.1 Module Function

Chapter 31

markdown Module

The `markdown` module provides measures to parse a text formatted in markdown syntax. To utilize it, import the `markdown` module using `import` function.

Below is an example to read a document written in Markdown format and then render its HTML text into a file.

```
import(markdown)
markdown.document('foo.md').render@html('foo.html')
```

`markdown` module consists of the following two module files:

- `markdown.gurd` .. a binary module file that provides parser procedures.
- `markdown.gura` .. a script module file that renders parsed result in desired formats.

31.1 Operator

`markdown.document << function`

31.2 markdown.document Class

The `markdown.document` class provides measures to parse a document written in Markdown format.

You can parse documents written in both string and stream using the following methods:

- `markdown.document#parse()` .. Parses document written in a string.
- `markdown.document#read()` .. Parses document from a stream.

You can get the parsed result by inspecting a property `markdown.document#root` and its children that are `markdown.item` instances.

31.2.1 Property

Property	Type	R/W	Explanation
refs	iterator	R	An iterator that returns referee items as markdown.item.
root	markdown.item	R	The root item of the parsed Markdown document.

31.2.2 Constructor

`markdown.document(stream?:stream:r) {block?}`

Returns an instance of `markdown.document`. If `stream` is specified, the content of the instance shall be initialized with the result of parsing the stream.

31.2.3 Method

`markdown.document#parse(str:string):void`

Parses a Markdown text in a string.

`markdown.document#read(stream:stream:r):void`

Parses a Markdown text from a stream.

`markdown.document#render@console(colorFlag:boolean => true)`

Renders the content of markdown document to the console.

In default, it uses colors to highlight items. Specify the argument `colorFlag` with `false` to disable the coloring process.

`markdown.document#render@html(out?:stream:w, easyFormatFlag:boolean => true, captionIndex:boolean => false`

`markdown.document#render@toc() {block}`

31.3 markdown.item Class

The `markdown.item` class provides information about items that composes a Markdown document.

Below is a table of item type:

Item Type	Explanation
`root`	container
`h1`	container
`h2`	container
`h3`	container
`h4`	container
`h5`	container
`h6`	container
`p`	container
`blockquote`	container
`em`	container
`strong`	container
`codeblock`	container
`ol`	container
`ul`	container
`li`	container
`line`	container
`a`	container
`img`	text
`text`	text
`code`	text
`entity`	text
`tag`	container/text
`hr`	no-content
`br`	no-content
`referee`	no-content

31.3.1 Property

Property	Type	R/W	Explanation
`type`	`string`	R	
`text`	`string`	R	
`children`	`iterator`	R	
`url`	`string`	R	
`title`	`string`	R	
`attrs`	`string`	R	
`align`	`symbol`	R	`none`, `left`, `center`, `right`

31.3.2 Method

`markdown.item#print(indent?:number):void`

Prints structured content of the item. Argument `indent` specifies an indentation level and is set to zero when omitted.

Chapter 32

math Module

The `math` module provices functions for mathematical calculation. This is a built-in module, so you can use it without being imported.

32.1 Module Function

`math.abs(num):map`

Returns an absolute value.

`math.acos(num):map:[deg]`

Returns an inverse cosine value.

In default, the result is returned in radian. Specifying an attribute `:deg` would return that in degree.

`math.arg(num):map:[deg]`

Returns an argument, an angle from the real-axis in the complex plane, of a complex number.

In default, the angle value is returned in radian. Specifying an attribute `:deg` would return that in degree.

`math.asin(num):map:[deg]`

Returns an inverse sine value.

In default, the result is returned in radian. Specifying an attribute `:deg` would return that in degree.

`math.atan(num):map:[deg]`

Returns an inverse tangent value.

`math.atan2(num1, num2):map:[deg]`

Returns an inverse tangent value of a fraction of num1 and num2.

In default, the result is returned in radian. Specifying an attribute `:deg` would return that in degree.

`math.bezier(nums[]+:number)`

Returns a list that consists of functions that generate coordinates of bezier curves with specified control points. One or more lists of control points can be specified. This means that if you give it two lists of numbers as arguments, it returns two functions of bezier curve.

`math.ceil(num):map`

Returns a nearest integer number above or equal to the specified value.

`math.conj(num):map`

Returns a conjugate of a complex number.

`math.cos(num):map:[deg]`

Returns a cosine value.

In default, the given argument is treated as a radian number. Specifying an attribute `:deg` would treat that as a degree number.

`math.cosh(num):map`

Returns a hyperbolic cosine value.

`math.covariance(a, b)`

Returns a covariance between the `a` and `b`.

`math.cross (a, b)`

Calculates a cross product between `a` and `b`.

`math.delta(num):map`

Evaluates a delta function with a given argument `num` that returns `1` when `num == 0` and `0` otherwise.

`math.diff(expr:expr, var:symbol):map {block?}`

Calculates a mathematical differential expression of the given `expr` by a variable `var`.

If `block` is specified, it would be evaluated with a block parameter `|rtn:expr|`, where `rtn` is the created instance. In this case, the block's result would become the function's returned value.

Example: `math.diff(`(math.sin(x 2)), `x)`∗∗

`math.dot(a, b)`

Calculates a dot product between `a` and `b`.

`math.exp(num):map`

Returns an exponential value.

`math.fft(seq[])`

`math.floor(num):map`

Returns a nearest integer number below or equal to the specified value.

`math.gcd(a:number, b+:number):map`

Returns a greatest common divisor among two or more numbers.

`math.hypot(x, y):map`

Returns a hyperbolic tangent value.

`math.imag(num):map`

Returns an imaginary part of a complex number.

`math.integral()`

`math.lcm(a:number, b+:number):map`

Returns a least common multiple among two or more numbers.

`math.least_square(x:iterator, y:iterator, dim:number => 1, var:symbol => 'x)`

Takes two iterators `x` and `y` that return coordinate of points and returns a function that fits them using least square metho. You can specify the fitting curve's dimension by an argument `dim`, which default value is one. The variable symbol used in the function is `x`, which can be changed by specifying an argument `var`.

`math.log(num):map`

Returns a natural logarithm value.

`math.log10(num):map`

Returns a decadic logarithm value.

`math.norm(num):map`

Returns a norm value of a complex number.

`math.optimize(expr:expr):map {block?}`

Returns an optimized expression of the given argument `expr`, which needs to be made up of mathematical elements.

If `block` is specified, it would be evaluated with a block parameter `|rtn:expr|`, where `rtn` is the created instance. In this case, the block's result would become the function's returned value.

`math.ramp(num):map`

Evaluates a ramp function with a given argument `num` that returns `num` when `num >= 0` and `0` otherwise.

`math.real(num):map`

Returns a real part of a complex number.

`math.sin(num):map:[deg]`

Returns a sine value.

In default, the given argument is treated as a radian number. Specifying an attribute `:deg` would treat that as a degree number.

`math.sinh(num):map`

Returns a hyperbolic sine value.

`math.sqrt(num):map`

Returns a square root value.

`math.tan(num):map:[deg]`

Returns a tangent value.

In default, the given argument is treated as a radian number. Specifying an attribute `:deg` would treat that as a degree number.

`math.tanh(num):map`

Returns a hyperbolic tangent value.

`math.unitstep(num):map`

Evaluates a unit step function with a given argument `num` that returns `1` when `num >= 0` and `0` otherwise.

Chapter 33

midi Module

The `midi` module provides measures to read/write MIDI files. To utilize it, import the `midi` module using `import` function.

33.1 Module Function

33.2 midi.event Class

33.3 midi.track Class

`midi.track#seek(offset:number, origin?:symbol):reduce`

Moves the insertion point in the track at which the next event is inserted. If `origin` is omitted or set to `` `set ``, the insertion point will be set to absolute offset from the beginning. If `origin` is set to `` `cur ``, the insertion point will be moved by offset from the current position.

`midi.track#tell()`

Returns the current insertion point in the track.

`midi.track#erase(n?:number):reduce`

Deletes an event at the current insertion point in the track. The argument `n` specifies the number of events to be deleted. If `n` is omitted, one event will be deleted.

`midi.track#mml(str:string, max_velocity?:number):map:reduce`

Parses MML in the string `str` and inserts resulted MIDI events at the current insertion point in the track.

The argument `max_velocity` specifies the maximum number of velocity in the MML. If omitted, it will be set to 127.

`midi.track#note_off(channel:number, note:number, velocity:number, deltaTime?:number):map:reduce`

`midi.track#note_on(channel:number, note:number, velocity:number, deltaTime?:number):map:reduce`

`midi.track#poly_pressure(channel:number, note:number, value:number, deltaTime?:number):map:reduce`

`midi.track#control_change(channel:number, controller, value:number, deltaTime?:number):map:reduce` midi.track#program_change(channel:number, program, deltaTime?:number):map:reduce

midi.track#channel_pressure(channel:number, pressure:number, deltaTime?:number):map:reduce

midi.track#pitch_bend(channel:number, value:number, deltaTime?:number):map:reduce

midi.track#sequence_number(number:number, deltaTime?:number):map:reduce

midi.track#text_event(text:string, deltaTime?:number):map:reduce

midi.track#copyright_notice(text:string, deltaTime?:number):map:reduce

midi.track#sequence_or_track_name(text:string, deltaTime?:number):map:reduce

midi.track#instrument_name(text:string, deltaTime?:number):map:reduce

midi.track#lyric_text(text:string, deltaTime?:number):map:reduce

midi.track#marker_text(text:string, deltaTime?:number):map:reduce

midi.track#cue_point(text:string, deltaTime?:number):map:reduce

midi.track#midi_channel_prefix_assignment(channel:number, deltaTime?:number):map:reduce

midi.track#end_of_track(deltaTime?:number):map:reduce

midi.track#tempo_setting(mpqn:number, deltaTime?:number):map:reduce

midi.track#smpte_offset(hour:number, minute:number, second:number, frame:number, subFrame:number, deltaTim

midi.track#time_signature(numerator:number, denominator:number, metronome:number, cnt32nd:number, deltaTim

midi.track#key_signature(key:number, scale:number, deltaTime?:number):map:reduce

midi.track#sequencer_specific_event(binary:binary, deltaTime?:number):map:reduce

33.4 midi.sequence Class

midi.sequence(stream?:stream) {block?}

It creates an instance that contains SMF information.

midi.sequence#read(stream:stream:r):map:reduce

midi.sequence#write(stream:stream:w):map:reduce

midi.sequence#play(port:midi.port, speed?:number, repeat:number:nil => 1):[background,player]

midi.sequence#track(index:number):map {block?}

midi.sequence#mml(str:string, max_velocity?:number):reduce `midi.sequence#readmml(stream:stream, max_velocity?:number):reduce`

33.5 midi.port Class

`midi.port#send(msg+:number):map:reduce`

`midi.port#play(sequence:midi.sequence, speed?:number, repeat:number:nil => 1):map:[background,player]`

`midi.port#mml(str:string, max_velocity?:number):[background,player]`

`midi.port#readmml(stream:stream, max_velocity?:number):[background,player]`

`midi.port#note_off(channel:number, note:number, velocity:number):map:reduce`

`midi.port#note_on(channel:number, note:number, velocity:number):map:reduce`

`midi.port#poly_pressure(channel:number, note:number, value:number):map:reduce`

`midi.port#control_change(channel:number, controller:number, value:number):map:reduce`

`midi.port#program_change(channel:number, program:number):map:reduce`

`midi.port#channel_pressure(channel:number, pressure:number):map:reduce`

`midi.port#pitch_bend(channel:number, value:number):map:reduce`

33.6 midi.controller Class

33.7 midi.program Class

33.8 midi.soundfont Class

`midi.soundfont(stream:stream) {block?}`

It creates an instance to access data in SoundFont file.

`midi.soundfont#synthesizer(preset:number, bank:number, key:number, velocity:number):map {block?}`

`midi.soundfont#print():void`

33.9 midi.synthesizer Class

Chapter 34

modbuild Module

The `modbuild` module ...

34.1 Module Function

Chapter 35

model.obj Module

The `model.obj` module provides measures to read/write files in OBJ format for 3D models.

Chapter 36

model.stl Module

The `model.stl` module provides measures to read/write files in STL format for 3D models.

Below is an example to read a STL file and to print information of faces it contains.

```
solid = model.stl.solid('example.stl')
println(solid.name || solid.header)
solid.faces.each {|face|
    printf('normal:  %g, %g, %g\n', face.normal.x, face.normal.y, face.normal.z)
    printf('vertex1: %g, %g, %g\n', face.vertex1.x, face.vertex1.y, face.vertex1.z)
    printf('vertex2: %g, %g, %g\n', face.vertex2.x, face.vertex2.y, face.vertex2.z)
    printf('vertex3: %g, %g, %g\n', face.vertex3.x, face.vertex3.y, face.vertex3.z)
}
```

36.1 model.stl.face Class

An instance of `model.stl.face` class provides properties of face that consists of one normal vector and three vertices.

36.1.1 Property

Property	Type	R/W	Explanation
`normal`	`vertex`	R	Normal vector.
`vertex1`	`vertex`	R	1st vertex.
`vertex2`	`vertex`	R	2nd vertex.
`vertex3`	`vertex`	R	3rd vertex.

36.2 model.stl.solid Class

An instance of `model.stl.solid` class represents a top-level data in STL format.

36.2.1 Property

Property	Type	R/W	Explanation
`header`	`string`	R	This is only valid for binary format and is set to 'nil' for ASCII.
`name`	`string`	R	This is only valid for ASCII format and is set to 'nil' for binary.
`faces`	`iterator`	R	An iterator that returns instances of `model.stl.face`.

36.2.2 Constructor

`stl.solid(stream:stream) {block?}`

Parses a file in STL format from `stream` and creates an instance of `model.stl.solid` that contains an iterator of `model.stl.face` representing faces in the STL. It can read both binary and ASCII format of STL.

If `block` is specified, it would be evaluated with a block parameter `|solid:model.stl.solid|`, where `solid` is the created instance. In this case, the block's result would become the function's returned value.

Chapter 37

msico Module

The `msico` module provides measures to read/write image data in Microsoft Icon file format. To utilize it, import the `msico` module using `import` function.

Below is an example to read an ICO file:

```
import(msico)
img = image('foo.ico')
```

This module has been implemented referring to the specification: http://msdn.microsoft.com/en-us/library/ms997538.aspx.

37.1 Exntension to Function's Capability

This module extends the capability of function `image()` and instance method `image#write()` so that they can read/write ICO files.

When function `image()` is provided with a stream that satisfies the following conditions, it would recognize the stream as a ICO file.

- The identifier of the stream ends with a suffix ".ico".

When instance method `image#write()` is provided with a stream that satisfies the following condition, it would write image data in ICO format.

- The identifier of the stream ends with a suffix ".ico".

37.2 msico.content Class

37.2.1 Constructor

`msico.content(stream?:stream:r, format:symbol => ‘rgba) {block?}`

37.2.2 Method

`msico.content#write(stream:stream:w):reduce`

Writes an ICO image to a stream.

`msico.content#addimage(image:image):map:reduce`

37.3 Extension to image Class

This module extends the `image` class with methods described here.

`image#read@msico(stream:stream:r, idx:number => 0):reduce`

Reads an ICO image from a stream.

Chapter 38

opengl Module

The `opengl` module provides functions of OpenGL library.

38.1 Module Function

`opengl.glAccum(op:number, value:number):map:void`

operate on the accumulation buffer

`opengl.glAlphaFunc(func:number, ref:number):map:void`

specify the alpha test function

`opengl.glAreTexturesResident(textures:array@uint32:nomap):map {block?}`

determine if textures are loaded in texture memory

`opengl.glArrayElement(i:number):map:void`

render a vertex using the specified vertex array element

`opengl.glBegin(mode:number):map:void {block?}`

delimit the vertices of a primitive or a group of like primitives

`opengl.glBindTexture(target:number, texture:number):map:void`

`opengl.glBitmap(width:number, height:number, xorig:number, yorig:number, xmove:number, ymove:number, bitma`

`opengl.glBlendFunc(sfactor:number, dfactor:number):map:void`

`opengl.glCallList(list:number):map:void`

`opengl.glCallLists(type:number, lists[]:number):map:void`

`opengl.glClear(mask:number):map:void`

`opengl.glClearAccum(red:number, green:number, blue:number, alpha:number):map:void`

`opengl.glClearColor(red:number, green:number, blue:number, alpha:number):map:void` opengl.glClearDepth(depth:number):map:void

opengl.glClearIndex(c:number):map:void

opengl.glClearStencil(s:number):map:void

opengl.glClipPlane(plane:number, equation:array@double:nomap):map:void {block?}

opengl.glColor3b(red:number, green:number, blue:number):map:void

opengl.glColor3bv(v:array@int8:nomap):map:void

opengl.glColor3d(red:number, green:number, blue:number):map:void

opengl.glColor3dv(v:array@double:nomap):map:void

opengl.glColor3f(red:number, green:number, blue:number):map:void

opengl.glColor3fv(v:array@float:nomap):map:void

opengl.glColor3i(red:number, green:number, blue:number):map:void

opengl.glColor3iv(v:array@int32:nomap):map:void

opengl.glColor3s(red:number, green:number, blue:number):map:void

opengl.glColor3sv(v:array@int16:nomap):map:void

opengl.glColor3ub(red:number, green:number, blue:number):map:void

opengl.glColor3ubv(v:array@uint8:nomap):map:void

opengl.glColor3ui(red:number, green:number, blue:number):map:void

opengl.glColor3uiv(v:array@uint32:nomap):map:void

opengl.glColor3us(red:number, green:number, blue:number):map:void

opengl.glColor3usv(v:array@uint16:nomap):map:void

opengl.glColor4b(red:number, green:number, blue:number, alpha:number):map:void

opengl.glColor4bv(v:array@int8:nomap):map:void

opengl.glColor4d(red:number, green:number, blue:number, alpha:number):map:void

opengl.glColor4dv(v:array@double:nomap):map:void

opengl.glColor4f(red:number, green:number, blue:number, alpha:number):map:void

opengl.glColor4fv(v:array@float:nomap):map:void

opengl.glColor4i(red:number, green:number, blue:number, alpha:number):map:void opengl.glColor4iv(v:array@int32:nomap):map:void

opengl.glColor4s(red:number, green:number, blue:number, alpha:number):map:void

opengl.glColor4sv(v:array@int16:nomap):map:void

opengl.glColor4ub(red:number, green:number, blue:number, alpha:number):map:void

opengl.glColor4ubv(v:array@uint8:nomap):map:void

opengl.glColor4ui(red:number, green:number, blue:number, alpha:number):map:void

opengl.glColor4uiv(v:array@uint32:nomap):map:void

opengl.glColor4us(red:number, green:number, blue:number, alpha:number):map:void

opengl.glColor4usv(v:array@uint16:nomap):map:void

opengl.glColorMask(red:boolean, green:boolean, blue:boolean, alpha:boolean):map:void

opengl.glColorMaterial(face:number, mode:number):map:void

opengl.glCopyPixels(x:number, y:number, width:number, height:number, type:number):map:void

opengl.glCopyTexImage1D(target:number, level:number, internalformat:number, x:number, y:number, width:numb

opengl.glCopyTexImage2D(target:number, level:number, internalformat:number, x:number, y:number, width:numb

opengl.glCopyTexSubImage1D(target:number, level:number, xoffset:number, x:number, y:number, width:number)

opengl.glCopyTexSubImage2D(target:number, level:number, xoffset:number, yoffset:number, x:number, y:number

opengl.glCullFace(mode:number):map:void

opengl.glDeleteLists(list:number, range:number):map:void

opengl.glDeleteTextures(textures:array@uint32:nomap):map:void

opengl.glDepthFunc(func:number):map:void

opengl.glDepthMask(flag:boolean):map:void

opengl.glDepthRange(zNear:number, zFar:number):map:void

opengl.glDisable(cap:number):map:void

opengl.glDisableClientState(array:number):map:void

opengl.glDrawArrays(mode:number, first:number, count:number):map:void

opengl.glDrawBuffer(mode:number):map:void

opengl.glDrawPixels(width:number, height:number, format:number, type:number, pixels:array:nomap):map:void opengl.glDrawPixelsFromImage(image:image):map:void

opengl.glEdgeFlag(flag:boolean):map:void

opengl.glEdgeFlagv(flag[]:boolean):map:void

opengl.glEnable(cap:number):map:void

opengl.glEnableClientState(array:number):map:void

opengl.glEnd():void

opengl.glEndList():void

opengl.glEvalCoord1d(u:number):map:void

opengl.glEvalCoord1dv(u:array@double:nomap):map:void

opengl.glEvalCoord1f(u:number):map:void

opengl.glEvalCoord1fv(u:array@float:nomap):map:void

opengl.glEvalCoord2d(u:number, v:number):map:void

opengl.glEvalCoord2dv(u:array@double:nomap):map:void

opengl.glEvalCoord2f(u:number, v:number):map:void

opengl.glEvalCoord2fv(u:array@float:nomap):map:void

opengl.glEvalMesh1(mode:number, i1:number, i2:number):map:void

opengl.glEvalMesh2(mode:number, i1:number, i2:number, j1:number, j2:number):map:void

opengl.glEvalPoint1(i:number):map:void

opengl.glEvalPoint2(i:number, j:number):map:void

opengl.glFeedbackBuffer(type:number, buffer:array@float:nil:nomap):void

opengl.glFinish():void

opengl.glFlush():void

opengl.glFogf(pname:number, param:number):map:void

opengl.glFogfv(pname:number, params:array@float:nomap):map:void

opengl.glFogi(pname:number, param:number):map:void

opengl.glFogiv(pname:number, params:array@int32:nomap):map:void

opengl.glFrontFace(mode:number):map:void opengl.glFrustum(left:number, right:number, bottom:number, top:number, zNear:number, zFar:number):map:void

opengl.glGenLists(range:number):map {block?}

opengl.glGenTextures(n:number):map {block?}

opengl.glGetBooleanv(pname:number):map {block?}

opengl.glGetClipPlane(plane:number):map

opengl.glGetDoublev(pname:number):map {block?}

opengl.glGetError() {block?}

opengl.glGetFloatv(pname:number):map {block?}

opengl.glGetIntegerv(pname:number):map {block?}

opengl.glGetLightfv(light:number, pname:number):map {block?}

opengl.glGetLightiv(light:number, pname:number):map {block?}

opengl.glGetMapdv(target:number, query:number, v:array@double:nomap):map:void

opengl.glGetMapfv(target:number, query:number, v:array@float:nomap):map:void

opengl.glGetMapiv(target:number, query:number, v:array@int32:nomap):map:void

opengl.glGetMaterialfv(face:number, pname:number):map {block?}

opengl.glGetMaterialiv(face:number, pname:number):map {block?}

opengl.glGetPixelMapfv(map:number, values:array@float:nomap):map:void

opengl.glGetPixelMapuiv(map:number, values:array@uint32:nomap):map:void

opengl.glGetPixelMapusv(map:number, values:array@uint16:nomap):map:void

opengl.glGetPolygonStipple():map

opengl.glGetString(name:number):map {block?}

opengl.glGetTexEnvfv(target:number, pname:number):map {block?}

opengl.glGetTexEnviv(target:number, pname:number):map {block?}

opengl.glGetTexGendv(coord:number, pname:number):map {block?}

opengl.glGetTexGenfv(coord:number, pname:number):map {block?}

opengl.glGetTexGeniv(coord:number, pname:number):map {block?}

opengl.glGetTexLevelParameterfv(target:number, level:number, pname:number):map {block?}

opengl.glGetTexLevelParameteriv(target:number, level:number, pname:number):map {block?}

opengl.glGetTexParameterfv(target:number, pname:number):map {block?}

opengl.glGetTexParameteriv(target:number, pname:number):map {block?}

opengl.glHint(target:number, mode:number):map:void

opengl.glIndexMask(mask:number):map:void

opengl.glIndexd(c:number):map:void

opengl.glIndexdv(c:array@double:nomap):map:void

opengl.glIndexf(c:number):map:void

opengl.glIndexfv(c:array@float:nomap):map:void

opengl.glIndexi(c:number):map:void

opengl.glIndexiv(c:array@int32:nomap):map:void

opengl.glIndexs(c:number):map:void

opengl.glIndexsv(c:array@int16:nomap):map:void

opengl.glIndexub(c:number):map:void

opengl.glIndexubv(c:array@uint8:nomap):map:void

opengl.glInitNames():void

opengl.glIsEnabled(cap:number):map {block?}

opengl.glIsList(list:number):map {block?}

opengl.glIsTexture(texture:number):map {block?}

opengl.glLightModelf(pname:number, param:number):map:void

opengl.glLightModelfv(pname:number, params:array@float:nomap):map:void

opengl.glLightModeli(pname:number, param:number):map:void

opengl.glLightModeliv(pname:number, params:array@int32:nomap):map:void

opengl.glLightf(light:number, pname:number, param:number):map:void

opengl.glLightfv(light:number, pname:number, params:array@float:nomap):map:void

opengl.glLighti(light:number, pname:number, param:number):map:void

opengl.glLightiv(light:number, pname:number, params:array@int32:nomap):map:void opengl.glLineStipple(factor:number, pattern:number):map:void

opengl.glLineWidth(width:number):map:void

opengl.glListBase(base:number):map:void

opengl.glLoadIdentity():void

opengl.glLoadMatrixd(m):void

opengl.glLoadMatrixf(m):void

opengl.glLoadName(name:number):map:void

opengl.glLogicOp(opcode:number):map:void

opengl.glMap1d(target:number, u1:number, u2:number, stride:number, order:number, points:array@double:nomap

opengl.glMap1f(target:number, u1:number, u2:number, stride:number, order:number, points:array@float:nomap)

opengl.glMap2d(target:number, u1:number, u2:number, ustride:number, uorder:number, v1:number, v2:number, v

opengl.glMap2f(target:number, u1:number, u2:number, ustride:number, uorder:number, v1:number, v2:number, v

opengl.glMapGrid1d(un:number, u1:number, u2:number):map:void

opengl.glMapGrid1f(un:number, u1:number, u2:number):map:void

opengl.glMapGrid2d(un:number, u1:number, u2:number, vn:number, v1:number, v2:number):map:void

opengl.glMapGrid2f(un:number, u1:number, u2:number, vn:number, v1:number, v2:number):map:void

opengl.glMaterialf(face:number, pname:number, param:number):map:void

opengl.glMaterialfv(face:number, pname:number, params:array@float:nomap):map:void

opengl.glMateriali(face:number, pname:number, param:number):map:void

opengl.glMaterialiv(face:number, pname:number, params:array@int32:nomap):map:void

opengl.glMatrixMode(mode:number):map:void

opengl.glMultMatrixd(m):void

opengl.glMultMatrixf(m):void

opengl.glNewList(list:number, mode:number):map:void {block?}

opengl.glNormal3b(nx:number, ny:number, nz:number):map:void

opengl.glNormal3bv(v:array@int8:nomap):map:void

opengl.glNormal3d(nx:number, ny:number, nz:number):map:void opengl.glNormal3dv(v:array@double:nomap):map:void

opengl.glNormal3f(nx:number, ny:number, nz:number):map:void

opengl.glNormal3fv(v:array@float:nomap):map:void

opengl.glNormal3i(nx:number, ny:number, nz:number):map:void

opengl.glNormal3iv(v:array@int32:nomap):map:void

opengl.glNormal3s(nx:number, ny:number, nz:number):map:void

opengl.glNormal3sv(v:array@int16:nomap):map:void

opengl.glOrtho(left:number, right:number, bottom:number, top:number, zNear:number, zFar:number):map:void

opengl.glPassThrough(token:number):map:void

opengl.glPixelMapfv(map:number, mapsize:number, values:array@float:nomap):map:void

opengl.glPixelMapuiv(map:number, mapsize:number, values:array@uint32:nomap):map:void

opengl.glPixelMapusv(map:number, mapsize:number, values:array@uint16:nomap):map:void

opengl.glPixelStoref(pname:number, param:number):map:void

opengl.glPixelStorei(pname:number, param:number):map:void

opengl.glPixelTransferf(pname:number, param:number):map:void

opengl.glPixelTransferi(pname:number, param:number):map:void

opengl.glPixelZoom(xfactor:number, yfactor:number):map:void

opengl.glPointSize(size:number):map:void

opengl.glPolygonMode(face:number, mode:number):map:void

opengl.glPolygonOffset(factor:number, units:number):map:void

opengl.glPolygonStipple(mask:array@uint8:nomap):map:void

opengl.glPopAttrib():void

opengl.glPopClientAttrib():void

opengl.glPopMatrix():void

opengl.glPopName():void

opengl.glPrioritizeTextures(textures:array@uint32:nomap, priorities:array@float:nomap):map:void

opengl.glPushAttrib(mask:number):map:void {block?}

opengl.glPushClientAttrib(mask:number):map:void {block?}

opengl.glPushMatrix():void {block?}

opengl.glPushName(name:number):map:void {block?}

opengl.glRasterPos2d(x:number, y:number):map:void

opengl.glRasterPos2dv(v:array@double:nomap):map:void

opengl.glRasterPos2f(x:number, y:number):map:void

opengl.glRasterPos2fv(v:array@float:nomap):map:void

opengl.glRasterPos2i(x:number, y:number):map:void

opengl.glRasterPos2iv(v:array@int32:nomap):map:void

opengl.glRasterPos2s(x:number, y:number):map:void

opengl.glRasterPos2sv(v:array@int16:nomap):map:void

opengl.glRasterPos3d(x:number, y:number, z:number):map:void

opengl.glRasterPos3dv(v:array@double:nomap):map:void

opengl.glRasterPos3f(x:number, y:number, z:number):map:void

opengl.glRasterPos3fv(v:array@float:nomap):map:void

opengl.glRasterPos3i(x:number, y:number, z:number):map:void

opengl.glRasterPos3iv(v:array@int32:nomap):map:void

opengl.glRasterPos3s(x:number, y:number, z:number):map:void

opengl.glRasterPos3sv(v:array@int16:nomap):map:void

opengl.glRasterPos4d(x:number, y:number, z:number, w:number):map:void

opengl.glRasterPos4dv(v:array@double:nomap):map:void

opengl.glRasterPos4f(x:number, y:number, z:number, w:number):map:void

opengl.glRasterPos4fv(v:array@float:nomap):map:void

opengl.glRasterPos4i(x:number, y:number, z:number, w:number):map:void

opengl.glRasterPos4iv(v:array@int32:nomap):map:void

opengl.glRasterPos4s(x:number, y:number, z:number, w:number):map:void

opengl.glRasterPos4sv(v:array@int16:nomap):map:void opengl.glReadBuffer(mode:number):map:void

opengl.glReadPixels(x:number, y:number, width:number, height:number, format:symbol):map {block?}

opengl.glRectd(x1:number, y1:number, x2:number, y2:number):map:void

opengl.glRectdv(v1:array@double:nomap, v2:array@double:nomap):map:void

opengl.glRectf(x1:number, y1:number, x2:number, y2:number):map:void

opengl.glRectfv(v1:array@float:nomap, v2:array@float:nomap):map:void

opengl.glRecti(x1:number, y1:number, x2:number, y2:number):map:void

opengl.glRectiv(v1:array@int32:nomap, v2:array@int32:nomap):map:void

opengl.glRects(x1:number, y1:number, x2:number, y2:number):map:void

opengl.glRectsv(v1:array@int16:nomap, v2:array@int16:nomap):map:void

opengl.glRenderMode(mode:number):map {block?}

opengl.glRotated(angle:number, x:number, y:number, z:number):map:void

opengl.glRotatef(angle:number, x:number, y:number, z:number):map:void

opengl.glScaled(x:number, y:number, z:number):map:void

opengl.glScalef(x:number, y:number, z:number):map:void

opengl.glScissor(x:number, y:number, width:number, height:number):map:void

opengl.glSelectBuffer(buffer:array@uint32:nil:nomap):void

opengl.glShadeModel(mode:number):map:void

opengl.glStencilFunc(func:number, ref:number, mask:number):map:void

opengl.glStencilMask(mask:number):map:void

opengl.glStencilOp(fail:number, zfail:number, zpass:number):map:void

opengl.glTexCoord1d(s:number):map:void

opengl.glTexCoord1dv(v:array@double:nomap):map:void

opengl.glTexCoord1f(s:number):map:void

opengl.glTexCoord1fv(v:array@float:nomap):map:void

opengl.glTexCoord1i(s:number):map:void

opengl.glTexCoord1iv(v:array@int32:nomap):map:void opengl.glTexCoord1s(s:number):map:void

opengl.glTexCoord1sv(v:array@int16:nomap):map:void

opengl.glTexCoord2d(s:number, t:number):map:void

opengl.glTexCoord2dv(v:array@double:nomap):map:void

opengl.glTexCoord2f(s:number, t:number):map:void

opengl.glTexCoord2fv(v:array@float:nomap):map:void

opengl.glTexCoord2i(s:number, t:number):map:void

opengl.glTexCoord2iv(v:array@int32:nomap):map:void

opengl.glTexCoord2s(s:number, t:number):map:void

opengl.glTexCoord2sv(v:array@int16:nomap):map:void

opengl.glTexCoord3d(s:number, t:number, r:number):map:void

opengl.glTexCoord3dv(v:array@double:nomap):map:void

opengl.glTexCoord3f(s:number, t:number, r:number):map:void

opengl.glTexCoord3fv(v:array@float:nomap):map:void

opengl.glTexCoord3i(s:number, t:number, r:number):map:void

opengl.glTexCoord3iv(v:array@int32:nomap):map:void

opengl.glTexCoord3s(s:number, t:number, r:number):map:void

opengl.glTexCoord3sv(v:array@int16:nomap):map:void

opengl.glTexCoord4d(s:number, t:number, r:number, q:number):map:void

opengl.glTexCoord4dv(v:array@double:nomap):map:void

opengl.glTexCoord4f(s:number, t:number, r:number, q:number):map:void

opengl.glTexCoord4fv(v:array@float:nomap):map:void

opengl.glTexCoord4i(s:number, t:number, r:number, q:number):map:void

opengl.glTexCoord4iv(v:array@int32:nomap):map:void

opengl.glTexCoord4s(s:number, t:number, r:number, q:number):map:void

opengl.glTexCoord4sv(v:array@int16:nomap):map:void

opengl.glTexEnvf(target:number, pname:number, param:number):map:void opengl.glTexEnvfv(target:number, pname:number, params:array@float:nomap):map:void

opengl.glTexEnvi(target:number, pname:number, param:number):map:void

opengl.glTexEnviv(target:number, pname:number, params:array@int32:nomap):map:void

opengl.glTexGend(coord:number, pname:number, param:number):map:void

opengl.glTexGendv(coord:number, pname:number, params:array@double:nomap):map:void

opengl.glTexGenf(coord:number, pname:number, param:number):map:void

opengl.glTexGenfv(coord:number, pname:number, params:array@float:nomap):map:void

opengl.glTexGeni(coord:number, pname:number, param:number):map:void

opengl.glTexGeniv(coord:number, pname:number, params:array@int32:nomap):map:void

opengl.glTexImage1D(target:number, level:number, internalformat:number, width:number, border:number, forma

opengl.glTexImage1DFromImage(target:number, level:number, internalformat:number, border:number, image:imag

opengl.glTexImage2D(target:number, level:number, internalformat:number, width:number, height:number, borde

opengl.glTexImage2DFromImage(target:number, level:number, internalformat:number, border:number, image:imag

opengl.glTexParameterf(target:number, pname:number, param:number):map:void

opengl.glTexParameterfv(target:number, pname:number, params:array@float:nomap):map:void

opengl.glTexParameteri(target:number, pname:number, param:number):map:void

opengl.glTexParameteriv(target:number, pname:number, params:array@int32:nomap):map:void

opengl.glTexSubImage1D(target:number, level:number, xoffset:number, width:number, format:number, type:numb

opengl.glTexSubImage1DFromImage(target:number, level:number, xoffset:number, image:image):map:void

opengl.glTexSubImage2D(target:number, level:number, xoffset:number, yoffset:number, width:number, height:n

opengl.glTexSubImage2DFromImage(target:number, level:number, xoffset:number, yoffset:number, image:image)

opengl.glTranslated(x:number, y:number, z:number):map:void

opengl.glTranslatef(x:number, y:number, z:number):map:void

opengl.glVertex2d(x:number, y:number):map:void

opengl.glVertex2dv(v:array@double:nomap):map:void

opengl.glVertex2f(x:number, y:number):map:void

opengl.glVertex2fv(v:array@float:nomap):map:void opengl.glVertex2i(x:number, y:number):map:void

opengl.glVertex2iv(v:array@int32:nomap):map:void

opengl.glVertex2s(x:number, y:number):map:void

opengl.glVertex2sv(v:array@int16:nomap):map:void

opengl.glVertex3d(x:number, y:number, z:number):map:void

opengl.glVertex3dv(v:array@double:nomap):map:void

opengl.glVertex3f(x:number, y:number, z:number):map:void

opengl.glVertex3fv(v:array@float:nomap):map:void

opengl.glVertex3i(x:number, y:number, z:number):map:void

opengl.glVertex3iv(v:array@int32:nomap):map:void

opengl.glVertex3s(x:number, y:number, z:number):map:void

opengl.glVertex3sv(v:array@int16:nomap):map:void

opengl.glVertex4d(x:number, y:number, z:number, w:number):map:void

opengl.glVertex4dv(v:array@double:nomap):map:void

opengl.glVertex4f(x:number, y:number, z:number, w:number):map:void

opengl.glVertex4fv(v:array@float:nomap):map:void

opengl.glVertex4i(x:number, y:number, z:number, w:number):map:void

opengl.glVertex4iv(v:array@int32:nomap):map:void

opengl.glVertex4s(x:number, y:number, z:number, w:number):map:void

opengl.glVertex4sv(v:array@int16:nomap):map:void

opengl.glViewport(x:number, y:number, width:number, height:number):map:void

opengl.glGetAttachedShaders(program:number, maxCount:number, count[]:number, shaders:array@uint32:nomap):m

opengl.glGetShaderInfoLog(shader:number):map {block?}

opengl.glGetProgramInfoLog(program:number):map {block?}

opengl.glGetUniformLocation(program:number, name:string):map {block?}

opengl.glGetActiveUniform(program:number, index:number):map {block?}

opengl.glGetUniformfv(program:number, location:number, params:array@float:nomap):map:void opengl.glGetUniformiv(program:number, location:number, params:array@int32:nomap):map:void

opengl.glGetShaderSource(shader:number):map:void

opengl.glBindAttribLocation(program:number, index:number, name:string):map:void

opengl.glGetActiveAttrib(program:number, index:number):map

opengl.glGetAttribLocation(program:number, name:string):map {block?}

opengl.glUniformMatrix2x3fv(location:number, count:number, transpose:boolean, value:array@float:nomap):map

opengl.glUniformMatrix3x2fv(location:number, count:number, transpose:boolean, value:array@float:nomap):map

opengl.glUniformMatrix2x4fv(location:number, count:number, transpose:boolean, value:array@float:nomap):map

opengl.glUniformMatrix4x2fv(location:number, count:number, transpose:boolean, value:array@float:nomap):map

opengl.glUniformMatrix3x4fv(location:number, count:number, transpose:boolean, value:array@float:nomap):map

opengl.glUniformMatrix4x3fv(location:number, count:number, transpose:boolean, value:array@float:nomap):map

Chapter 39

os Module

The `os` module provides functions that are specific to each OS environment. This is a built-in module, so you can use it without being imported.

39.1 Module Function

`os.clock() {block?}`

Returns the time duration in second since the system has started.

If `block` is specified, it would calculate how much time has been spent during evaluating the block.

`os.exec(pathname:string, args*:string):map:[fork]`

Executes the specified executable file.

`os.fromnative(buff:binary):map`

Converts binary data that includes OS's native string into Gura's regulated string.

`os.getenv(name:string, default?:string):map`

Returns the value of an environment variable.

`os.putenv(name:string, value:string):void`

Set the value of an environment variable.

`os.redirect(stdin:stream:nil:r, stdout:stream:nil:w, stderr?:stream:w) {block?}`

Modifies variables `os.stdin`, `os.stdout` and `os.stderr` with values of arguments. When `block` is specified, the modification only has effect within the block.

`os.sleep(secs:number)`

Sleeps for a time specified in seconds.

`os.symlink(src:string, tgt:string):map:void`

Creates a symbol link.

`os.tonative(str:string):map`

Converts Gura's regulated string into binary data that includes OS's native string.

`os.unsetenv(name:string):void`

Unset an environment variable.

Chapter 40

path Module

The `path` module provides functions related to path operations. This is a built-in module, so you can use it without being imported.

Below is an example to list path names that exist in the current directory.

```
println(path.dir('.'))
```

Below is an example to list path names that exist in the current directory and its child directories.

```
println(path.walk('.'))
```

Below is an example to list path names that matches a wild card pattern "`*.txt`".

```
println(path.glob('*.txt'))
```

40.1 Module Function

`path.absname(name:string):map:[uri]`

Returns an absolute path name of the given name.

`path.basename(pathname:string):map`

Removes a suffix part of a path name.

`path.bottom(pathname:string):map`

Returns the last part of a path name.

`path.cutbottom(pathname:string):map`

Returns a path name after eliminating its bottom part.

`path.dir(directory?:directory, pattern*:string):flat:map:[dir,file,stat] {block?}`

Creates an iterator that lists item names in the specified directory. If pathname is omitted, the current directory shall be listed. In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`path.dirname(pathname:string):map`

Splits a pathname by a directory separator and returns a directory name part.

`path.exists(pathname:string):map`

Returns true if the specified file exists in a file system.

`path.extname(pathname:string):map`

Extracts a suffix part of a path name.

`path.filename(pathname:string):map`

Splits a pathname by a directory separator and returns a file name part.

`path.glob(pattern:string):flat:map:[dir,file,stat] {block?}`

Creates an iterator for item names that match with a pattern supporting UNIX shell-style wild cards. In default, case of characters is distinguished. In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`path.join(paths+:string):map:[uri]`

Returns a path name that joins given strings with directory separators.

`path.match(pattern:string, name:string):map`

Returns true if a name matches with a pattern that supports UNIX shell-style wild cards. In default, case of characters is distinguished.

`path.regulate(name:string):map:[uri]`

Returns a regulated path name of the given name.

`path.split(pathname:string):map:[bottom]`

Splits a pathname by a directory separator and returns a list containing a directory name as the first element and a base name as the second one. This has the same result as calling path.dirname() and path.filename().

`path.splitext(pathname:string):map`

Splits a pathname by a dot character indicating a beginning of an extension and returns a list containing a path name without an extention and an extention part.

`path.stat(directory:directory):map`

Returns a stat object associated with the specified item.

`path.walk(directory?:directory, maxdepth?:number, pattern*:string):flat:map:[dir,file,stat] {block?}`

Creates an iterator that recursively lists item names under the specified directory. If pathname is omitted, search starts at the current directory In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

Chapter 41

png Module

The `png` module provides measures to read/write image data in PNG format. To utilize it, import the `png` module using `import` function.

Below is an example to read a PNG file:

```
import(png)
img = image('foo.png')
```

41.1 Exntension to Function's Capability

This module extends the capability of function `image()` and instance method `image#write()` so that they can read/write PNG files.

When function `image()` is provided with a stream that satisfies the following conditions, it would recognize the stream as a PNG file.

- The identifier of the stream ends with a suffix "`.png`".
- The stream data begins with a byte sequence "`\x89\x50\x4e\x47\x0d\x0a\x1a\x0a`".

When instance method `image#write()` is provided with a stream that satisfies the following condition, it would write image data in PNG format.

- The identifier of the stream ends with a suffix "`.png`".

41.2 Module Function

41.3 Extension to image Class

This module extends the `image` class with methods described here.

`image#read@png(stream:stream:r):reduce`

Reads a PNG image from a stream.

`image#write@png(stream:stream:w):reduce`

Writes a PNG image to a stream.

41.4 Thanks

This module uses libpng library which is distributed in the following site:
http://www.libpng.org/pub/png/libpng.html

Chapter 42

ppm Module

The `ppm` module provides measures to read/write image data in PPM format. To utilize it, import the `ppm` module using `import` function.

Below is an example to read a PPM file:

```
import(ppm)
img = image('foo.ppm')
```

42.1 Exntension to Function's Capability

This module extends the capability of function `image()` and instance method `image#write()` so that they can read/write PPM files.

When function `image()` is provided with a stream that satisfies the following conditions, it would recognize the stream as a PPM file.

- The identifier of the stream ends with a suffix "`.ppm`" or "`.pbm`".
- The stream data begins with a byte sequence "`P2`", "`P3`" or "`P6`".

When instance method `image#write()` is provided with a stream that satisfies the following condition, it would write image data in PPM format.

- The identifier of the stream ends with a suffix "`.ppm`" or "`.pbm`".

42.2 Extension to image Class

This module extends the `image` class with methods described here.

`image#read@ppm(stream:stream:r):reduce`

Reads a PPM/PGM image from a stream.

`image#write@ppm(stream:stream:w):reduce:[gray]`

Writes a PPM/PGM image to a stream.

Chapter 43

re Module

The `re` module provides measures to operate strings with a regular expression. To utilize it, import the `re` module using `import` function.

This module provides three different forms of function that has the same feature as below:

- Module function
- Method of `re.pattern` class
- Method of `string` class

For example, a feature to match a string with a regular expression can be described as below:

Using a module function:

```
m = re.match('gur[ai]', str)
```

Using a method of `re.pattern` class:

```
m = re.pattern('gur[ai]').match(str)
```

Using a method of `string` class:

```
m = str.match('gur[ai]')
```

The table below shows the features related to regular-expression and functions that provides them.

Feature	**Module Function**	**Method of re.pattern**	**Method of string**
Match	`re.match()`	`re.pattern#match()`	`string#match()`
Subtraction	`re.sub()`	`re.pattern#sub()`	`string#sub()`
Split	`re.split()`	`re.pattern#split()`	`string#splitsub()`
Scan	`re.scan()`	`re.pattern#scan()`	`string#scan()`

43.1 Regular Expression

You can describe a matching pattern using a syntax based on POSIX Extended Regular Expression.

The syntax uses a back slash character to avoid some characters such as "`(`" and "`)`" from being recognized as a meta character. Since a back slash is used as an escaping character in Gura string as well, you have to write two back slashes to represent a single back slash in a regular expression. For example, an expression "`sin\(x\)`" that matches a string "`sin(x)`" is described as below:

```
m = str.match('sin\\(x\\)')
```

Using a raw string appended with a prefix "`r`", in which a back slash is parsed as a regular character, could avoid such complications.

```
m = str.match(r'sin\(x\)')
```

43.2 re.match Class

An instance of `re.match` class is used as a result value of `re.match()`, `re.pattern#match()` and `string#match()` to provide matching information.

43.2.1 Property

Property	Type	R/W	Explanation
`source`	`string`	R	String that has been matched.
`string`	`string`	R	String of the matched part.
`begin`	`number`	R	Beginning position of the matched part.
`end`	`number`	R	Ending position of the matched part.

43.2.2 Index Access

A `re.match` instance can be indexed with a `number` or `string` value.

The value of `number` indicates the group index number that starts from zero. The group indexed by zero is special and represents the whole region of the match. The groups indexed by numbers greater than zero correspond to matching patterns of grouping.

Below is an example:

```
str = '12:34:56'\n"
m = str.match(r'(\d\d):(\d\d):(\d\d)')\n"
m[0]  // returns the whole region of matching: 12:34:56\n"
m[1]  // returns the 1st group: 12\n"
m[2]  // returns the 2nd group: 34\n"
m[3]  // returns the 3rd group: 56\n"
```

The value of `string` is used to point out a named capturing group that is described as "`(?<name>group)`" in a regular expression.

Below is an example:

```
str = '12:34:56'\n"
m = str.match(r'(?<hour>\d\d):(?<min>\d\d):(?<sec>\d\d)')\n"
m['hour']  // returns the group named 'hour': 12\n"
m['min']   // returns the group named 'min': 34\n"
m['sec']   // returns the group named 'sec': 56\n");
```

43.2.3 Method

re.match#group(index):map

Returns a `re.group` instance that is positioned by the specified index.

The argument `index` is a value of `number` or `string`.

The value of `number` indicates the group index number that starts from zero. The group indexed by zero is special and represents the whole region of the match. The groups indexed by numbers greater than zero correspond to matching patterns of grouping. Below is an example:

```
str = '12:34:56'
m = str.match(r'(\d\d):(\d\d):(\d\d)')
m.group(0).string // returns the whole region of matching: 12:34:56
m.group(1).string // returns the 1st group: 12
m.group(2).string // returns the 2nd group: 34
m.group(3).string // returns the 3rd group: 56
```

The value of `string` is used to point out a named capturing group that is described in a regular expression as "`(?<name>group)`".

Below is an example:

```
str = '12:34:56'
m = str.match(r'(?<hour>\d\d):(?<min>\d\d):(?<sec>\d\d)')
m.group('hour').string // returns the group named 'hour': 12
m.group('min').string  // returns the group named 'min': 34
m.group('sec').string  // returns the group named 'sec': 56
```

re.match#groups() {block?}

Creates an iterator that returns `re.group` instances.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

43.3 re.group Class

The `re.group` instance provides information of capturing groups that are stored in `re.match` instance.

43.3.1 Property

Property	Type	R/W	Explanation
`string`	`string`	R	String of the group.
`begin`	`number`	R	Beginning position of the group.
`end`	`number`	R	Ending position of the group.

43.4 re.pattern Class

The `re.pattern` class is used to describe a pattern of regular expression.

43.4.1 Cast Operation

A function that expects a `re.pattern` instance in its argument can also take a value of `string` below:

- `string` .. Recognized as a regular expression from which `re.pattern` instance is created.

Using the above casting feature, you can call a function `f(pattern:re.pattern)` that expects a `re.pattern` instance in its argument as below:

- `f(re.pattern('gur[ai]'))` .. The most explicit way.
- `f('gur[ai]')` .. Implicit casting: from `string` to `re.pattern`.

43.4.2 Constructor

In many cases, `re.pattern` instance may be implicitly created by cast operation when a `string` is passed to a function's argument that expects `re.pattern` type. If you want to customize the pattern's behaviour, such as indicating it to ignore alphabet cases, you can explicitly create the instance with the constructor described below.

`re.pattern(pattern:string):map:[icase,multiline] {block?}`

Creates a `re.pattern` instance from the given pattern string.

Following attributes would customize some traits of the pattern:

- `:icase` .. Ignores character cases.

- `:multiline` .. Matches "`.`" with a line break.

If `block` is specified, it would be evaluated with a block parameter `|pat:re.pattern|`, where `pat` is the created instance. In this case, the block's result would become the function's returned value.

43.4.3 Method

`re.pattern#match(str:string, pos:number => 0, endpos?:number):map {block?}`

Applies a pattern matching to the given string and returns a `re.match` instance if the matching successes. If not, it would return `nil`.

The argument `pos` specifies the starting position for matching process. If omitted, it starts from the beginning of the string.

The argument `endpos` specifies the ending position for matching process. If omitted, it would be processed until the end of the string.

If `block` is specified, it would be evaluated with a block parameter `|m:re.match|`, where `m` is the created instance. In this case, the block's result would become the function's returned value.

`re.pattern#sub(replace, str:string, count?:number):map {block?}`

Substitutes strings that matches `pattern` with the specified replacer.

The argument `replace` takes a `string` or `function`.

If a `string` is specified, it would be used as a substituting string, in which you can use macros `\0`, `\1`, `\2` .. to refer to matched groups.

If a `function` is specified, it would be called with an argument `m:re.match` and is expected to return a string for subsitution.

The argument `count` specifies the maximum number of substitutions. If omitted, no limit would be applied.

If `block` is specified, it would be evaluated with a block parameter `|str:string|`, where `str` is the created instance. In this case, the block's result would become the function's returned value.

`re.pattern#split(str:string, count?:number):map {block?}`

Creates an iterator that splits the source string with the specified pattern.

The argument `count` specifies the maximum number for splitting. If omitted, no limit would be applied.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`re.pattern#scan(str:string, pos:number => 0, endpos?:number):map {block?}`

Creates an iterator that returns strings that match the specified pattern.

The argument `pos` specifies the starting position for matching process. If omitted, it starts from the beginning of the string.

The argument `endpos` specifies the ending position for matching process. If omitted, it would be processed until the end of the string.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

43.5 Extension to string Class

This module extends the `string` class with methods described here.

`string#match(pattern:re.pattern, pos:number => 0, endpos?:number):map {block?}`

Applies a pattern matching to the given string and returns a `re.match` instance if the matching successes. If not, it would return `nil`.

The argument `pos` specifies the starting position for matching process. If omitted, it starts from the beginning of the string.

The argument `endpos` specifies the ending position for matching process. If omitted, it would be processed until the end of the string.

If `block` is specified, it would be evaluated with a block parameter `|m:re.match|`, where `m` is the created instance. In this case, the block's result would become the function's returned value.

`string#sub(pattern:re.pattern, replace, count?:number):map {block?}`

Substitutes strings that matches `pattern` with the specified replacer.

The argument `replace` takes a `string` or `function`.

If a `string` is specified, it would be used as a substituting string, in which you can use macros `\0`, `\1`, `\2` .. to refer to matched groups.

If a `function` is specified, it would be called with an argument `m:re.match` and is expected to return a string for subsitution.

The argument `count` specifies the maximum number of substitutions. If omitted, no limit would be applied.

If `block` is specified, it would be evaluated with a block parameter `|str:string|`, where `str` is the created instance. In this case, the block's result would become the function's returned value.

`string#splitreg(pattern:re.pattern, count?:number):map {block?}`

Creates an iterator that splits the source string with the specified pattern.

The argument `count` specifies the maximum number for splitting. If omitted, no limit would be applied.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

`string#scan(pattern:re.pattern, pos:number => 0, endpos?:number):map {block?}`

Creates an iterator that returns strings that match the specified pattern.

The argument `pos` specifies the starting position for matching process. If omitted, it starts from the beginning of the string.

The argument `endpos` specifies the ending position for matching process. If omitted, it would be processed until the end of the string.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

43.6 Extension to iterable Classes

This module extends the iterable classes, `list` and `iterator`, with methods described here.

`iterable#grep(pattern:re.pattern):map {block?}`

43.7 Module Function

`re.match(pattern:re.pattern, str:string, pos:number => 0, endpos?:number):map {block?}`

Applies a pattern matching to the given string and returns a `re.match` instance if the matching successes. If not, it would return `nil`.

The argument `pos` specifies the starting position for matching process. If omitted, it starts from the beginning of the string.

The argument `endpos` specifies the ending position for matching process. If omitted, it would be processed until the end of the string.

If `block` is specified, it would be evaluated with a block parameter `|m:re.match|`, where `m` is the created instance. In this case, the block's result would become the function's returned value.

`re.sub(pattern:re.pattern, replace, str:string, count?:number):map {block?}`

Substitutes strings that matches `pattern` with the specified replacer.

The argument `replace` takes a `string` or `function`.

If a `string` is specified, it would be used as a substituting string, in which you can use macros `\0`, `\1`, `\2` .. to refer to matched groups.

If a `function` is specified, it would be called with an argument `m:re.match` and is expected to return a string for subsitution.

The argument `count` specifies the maximum number of substitutions. If omitted, no limit would be applied.

If `block` is specified, it would be evaluated with a block parameter `|str:string|`, where `str` is the created instance. In this case, the block's result would become the function's returned value.

`re.split(pattern:re.pattern, str:string, count?:number):map {block?}`

Creates an iterator that splits the source string with the specified pattern.

The argument `count` specifies the maximum number for splitting. If omitted, no limit would be applied.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.

- :xiter .. An iterator that eliminates nil from its elements.
- :list .. A list.
- :xlist .. A list that eliminates nil from its elements.
- :set .. A list that eliminates duplicated values from its elements.
- :xset .. A list that eliminates duplicated values and nil from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters |value, idx:number| where value is the iterated value and idx the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

re.scan(pattern:re.pattern, str:string, pos:number => 0, endpos?:number):map {block?}

Creates an iterator that returns strings that match the specified pattern.

The argument pos specifies the starting position for matching process. If omitted, it starts from the beginning of the string.

The argument endpos specifies the ending position for matching process. If omitted, it would be processed until the end of the string.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- :iter .. An iterator. This is the default behavior.
- :xiter .. An iterator that eliminates nil from its elements.
- :list .. A list.
- :xlist .. A list that eliminates nil from its elements.
- :set .. A list that eliminates duplicated values from its elements.
- :xset .. A list that eliminates duplicated values and nil from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters |value, idx:number| where value is the iterated value and idx the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

43.8 Thanks

This module uses Oniguruma library which is distributed in the following site:

http://www.geocities.jp/kosako3/oniguruma/index.html

Chapter 44

show Module

The `show` module provides a method to display the content of `image` instance.

44.1 Extension to image Class

This module extends the `image` class with a method described here.

`image#show(width => 640, height => 480)`

Displays the content of `image` instance in a window.

Chapter 45

sdl2 Module

The sdl2 module provices functions of SDL2 library.

45.1 Module Function

sdl2.Init(flags:number):void

Use this function to initialize the SDL library. This must be called before using any other SDL function.

The Event Handling, File I/O, and Threading subsystems are initialized by default. You must specifically initialize other subsystems if you use them in your application.

flags may be any of the following OR'd together:

- sdl2.INIT_TIMER .. timer subsystem
- sdl2.INIT_AUDIO .. audio subsystem
- sdl2.INIT_VIDEO .. video subsystem
- sdl2.INIT_JOYSTICK .. joystick subsystem
- sdl2.INIT_HAPTIC .. haptic (force feedback) subsystem
- sdl2.INIT_GAMECONTROLLER .. controller subsystem
- sdl2.INIT_EVENTS .. events subsystem
- sdl2.INIT_EVERYTHING .. all of the above subsystems
- sdl2.INIT_NOPARACHUTE .. compatibility; this flag is ignored

If you want to initialize subsystems separately you would call SDL_Init(0) followed by SDL_InitSubSystem() with the desired subsystem flag.

sdl2.InitSubSystem(flags:number):void

Use this function to initialize specific SDL subsystems.

After SDL has been initialized with SDL_Init() you may initialize uninitialized subsystems with SDL_InitSubSystem().

These are the flags which may be passed to SDL_InitSubSystem() and may be OR'd together to initialize multiple subsystems simultaneously.

- `sdl2.INIT_TIMER` .. timer subsystem
- `sdl2.INIT_AUDIO` .. audio subsystem
- `sdl2.INIT_VIDEO` .. video subsystem
- `sdl2.INIT_JOYSTICK` .. joystick subsystem
- `sdl2.INIT_HAPTIC` .. haptic (force feedback) subsystem
- `sdl2.INIT_GAMECONTROLLER` .. controller subsystem
- `sdl2.INIT_EVENTS` .. events subsystem
- `sdl2.INIT_EVERYTHING` .. all of the above subsystems
- `sdl2.INIT_NOPARACHUTE` .. compatibility; this flag is ignored

If you want to initialize subsystems separately you would call `SDL_Init(0)` followed by `SDL_InitSubSystem()` with the desired subsystem flag.

`sdl2.Quit():void`

Use this function to clean up all initialized subsystems. You should call it upon all exit conditions.

You should call this function even if you have already shutdown each initialized subsystem with `SDL_QuitSubSystem()`.

If you start a subsystem using a call to that subsystem's init function (for example `SDL_VideoInit()`) instead of `SDL_Init()` or `SDL_InitSubSystem()`, then you must use that subsystem's quit function (`SDL_VideoQuit()`) to shut it down before calling `SDL_Quit()`.

You can use this function with `atexit()` to ensure that it is run when your application is shutdown, but it is not wise to do this from a library or other dynamically loaded code.

`sdl2.QuitSubSystem(flags:number):void`

Use this function to shut down specific SDL subsystems.

These are the flags which may be passed to `SDL_QuitSubSystem()` and may be OR'd together to quit multiple subsystems simultaneously.

- `sdl2.INIT_TIMER` .. timer subsystem
- `sdl2.INIT_AUDIO` .. audio subsystem
- `sdl2.INIT_VIDEO` .. video subsystem
- `sdl2.INIT_JOYSTICK` .. joystick subsystem
- `sdl2.INIT_HAPTIC` .. haptic (force feedback) subsystem
- `sdl2.INIT_GAMECONTROLLER` .. controller subsystem
- `sdl2.INIT_EVENTS` .. events subsystem
- `sdl2.INIT_EVERYTHING` .. all of the above subsystems
- `sdl2.INIT_NOPARACHUTE` .. compatibility; this flag is ignored

If you want to initialize subsystems separately you would call `SDL_Init(0)` followed by `SDL_InitSubSystem()` with the desired subsystem flag.

`sdl2.SetMainReady():void`

Use this function to circumvent failure of `SDL_Init()` when not using `SDL_main()` as an entry point.

This function is defined in SDL_main.h, along with the preprocessor rule to redefine `main()` as `SDL_main()`. Thus to ensure that your `main()` function will not be changed it is necessary to define `SDL_MAIN_HANDLED` before including SDL.h.

`sdl2.WasInit(flags:number) {block?}`

Use this function to return a mask of the specified subsystems which have previously been initialized.

These are the flags which may be passed to `SDL_WasInit()` and may be OR'd together to query multiple subsystems simultaneously.

- `sdl2.INIT_TIMER` .. timer subsystem
- `sdl2.INIT_AUDIO` .. audio subsystem
- `sdl2.INIT_VIDEO` .. video subsystem
- `sdl2.INIT_JOYSTICK` .. joystick subsystem
- `sdl2.INIT_HAPTIC` .. haptic (force feedback) subsystem
- `sdl2.INIT_GAMECONTROLLER` .. controller subsystem
- `sdl2.INIT_EVENTS` .. events subsystem
- `sdl2.INIT_EVERYTHING` .. all of the above subsystems
- `sdl2.INIT_NOPARACHUTE` .. compatibility; this flag is ignored

If you want to initialize subsystems separately you would call `SDL_Init(0)` followed by `SDL_InitSubSystem()` with the desired subsystem flag.

`sdl2.AddHintCallback():void`

`sdl2.ClearHints():void`

`sdl2.DelhintCallback():void`

`sdl2.GetHint():void`

`sdl2.SetHint():void`

`sdl2.SetHintWithPriority():void`

`sdl2.ClearError():void`

Use this function to clear any previous error message.

`sdl2.GetError() {block?}`

Use this function to retrieve a message about the last error that occurred.

Returns a message with information about the specific error that occurred, or an empty string if there hasn't been an error since the last call to `SDL_ClearError()`. Without calling `SDL_ClearError()`, the message is only applicable when an SDL function has signaled an error. You must check the return values of SDL function calls to determine when to appropriately call `SDL_GetError()`.

This string is statically allocated and must not be freed by the application.

It is possible for multiple errors to occur before calling `SDL_GetError()`. Only the last error is returned.

`sdl2.SetError():void`

`sdl2.Log():void`

`sdl2.LogCritical():void`

`sdl2.LogDebug():void`

`sdl2.LogError():void`

`sdl2.LogGetOutputFunction():void`

`sdl2.LogGetPriority():void`

`sdl2.LogInfo():void`

`sdl2.LogMessage():void`

`sdl2.LogMessageV():void`

`sdl2.LogResetPriorities():void`

`sdl2.LogSetAllPriority():void`

`sdl2.LogSetOutputFunction():void`

`sdl2.LogSetPriority():void`

`sdl2.LogVerbose():void`

`sdl2.LogWarn():void`

`sdl2.GetAssertionHandler():void`

`sdl2.GetAssertionReport():void`

`sdl2.GetDefaultAssertionHandler():void`

`sdl2.ResetAssertionReport():void`

`sdl2.SetAssertionHandler():void`

`sdl2.TriggerBreakpoint():void`

`sdl2.assert():void`

`sdl2.assert_paranoid():void`

sdl2.assert_release():void

sdl2.GetRevision() {block?}

sdl2.GetRevisionNumber() {block?}

sdl2.GetVersion() {block?}

sdl2.VERSION() {block?}

sdl2.VERSION_ATLEAST(X:number, Y:number, Z:number) {block?}

sdl2.CreateWindow(title:string, x:number, y:number, w:number, h:number, flags:number) {block?}

sdl2.CreateWindowAndRenderer(width:number, height:number, window_flags:number) {block?}

sdl2.CreateWindowFrom():void

sdl2.DestroyWindow(window:sdl2.Window):void

sdl2.DisableScreenSaver():void

sdl2.EnableScreenSaver():void

sdl2.GL_CreateContext(window:sdl2.Window) {block?}

sdl2.GL_DeleteContext(context:sdl2.GLContext):void

sdl2.GL_ExtensionSupported(extension:string) {block?}

sdl2.GL_GetAttribute(attr:number) {block?}

sdl2.GL_GetCurrentContext() {block?}

sdl2.GL_GetCurrentWindow() {block?}

sdl2.GL_GetDrawableSize(window:sdl2.Window) {block?}

sdl2.GL_GetProcAddress():void

sdl2.GL_GetSwapInterval() {block?}

sdl2.GL_LoadLibrary(path:string):void

sdl2.GL_MakeCurrent(window:sdl2.Window, context:sdl2.GLContext):void

sdl2.GL_ResetAttributes():void

sdl2.GL_SetAttribute(attr:number, value:number):void

sdl2.GL_SetSwapInterval(interval:number):void

sdl2.GL_SwapWindow(window:sdl2.Window):void sdl2.GL_UnloadLibrary():void

sdl2.GetClosestDisplayMode(displayIndex:number, mode:sdl2.DisplayMode) {block?}

sdl2.GetCurrentDisplayMode(displayIndex:number) {block?}

sdl2.GetCurrentVideoDriver() {block?}

sdl2.GetDesktopDisplayMode(displayIndex:number) {block?}

sdl2.GetDisplayBounds(displayIndex:number) {block?}

sdl2.GetDisplayMode(displayIndex:number, modeIndex:number) {block?}

sdl2.GetDisplayName(dipslayIndex:number) {block?}

sdl2.GetNumDisplayModes(displayIndex:number) {block?}

sdl2.GetNumVideoDisplays() {block?}

sdl2.GetNumVideoDrivers() {block?}

sdl2.GetVideoDriver(index:number) {block?}

sdl2.GetWindowBrightness(window:sdl2.Window) {block?}

sdl2.GetWindowData(window:sdl2.Window, name:string):void

sdl2.GetWindowDisplayIndex(window:sdl2.Window) {block?}

sdl2.GetWindowDisplayMode(window:sdl2.Window, mode:sdl2.DisplayMode):void

sdl2.GetWindowFlags(window:sdl2.Window) {block?}

sdl2.GetWindowFromID(id:number) {block?}

sdl2.GetWindowGammaRamp(window:sdl2.Window) {block?}

sdl2.GetWindowGrab(window:sdl2.Window) {block?}

sdl2.GetWindowID(window:sdl2.Window) {block?}

sdl2.GetWindowMaximumSize(window:sdl2.Window) {block?}

sdl2.GetWindowMinimumSize(window:sdl2.Window) {block?}

sdl2.GetWindowPixelFormat(window:sdl2.Window) {block?}

sdl2.GetWindowPosition(window:sdl2.Window) {block?}

sdl2.GetWindowSize(window:sdl2.Window) {block?}

sdl2.GetWindowSurface(window:sdl2.Window) {block?}

sdl2.GetWindowTitle(window:sdl2.Window) {block?}

sdl2.GetWindowWMInfo(window:sdl2.Window):void

sdl2.HideWindow(window:sdl2.Window):void

sdl2.IsScreenSaverEnabled() {block?}

sdl2.MaximizeWindow(window:sdl2.Window):void

sdl2.MinimizeWindow(window:sdl2.Window):void

sdl2.RaiseWindow(window:sdl2.Window):void

sdl2.RestoreWindow(window:sdl2.Window):void

sdl2.SetWindowBordered(window:sdl2.Window, bordered:boolean):void

sdl2.SetWindowBrightness(window:sdl2.Window, brightness:number):void

sdl2.SetWindowData(window:sdl2.Window, name:string):void

sdl2.SetWindowDisplayMode(window:sdl2.Window, mode:sdl2.DisplayMode):void

sdl2.SetWindowFullscreen(window:sdl2.Window, flags:number):void

sdl2.SetWindowGammaRamp(window:sdl2.Window, red[]:number, green[]:number, blue[]:number):void

sdl2.SetWindowGrab(window:sdl2.Window, grabbed:boolean):void

sdl2.SetWindowHitTest(window:sdl2.Window):void

sdl2.SetWindowIcon(window:sdl2.Window, icon:sdl2.Surface):void

sdl2.SetWindowMaximumSize(window:sdl2.Window, max_w:number, max_h:number):void

sdl2.SetWindowMinimumSize(window:sdl2.Window, min_w:number, min_h:number):void

sdl2.SetWindowPosition(window:sdl2.Window, x:number, y:number):void

sdl2.SetWindowSize(window:sdl2.Window, w:number, h:number):void

sdl2.SetWindowTitle(window:sdl2.Window, title:string):void

sdl2.ShowMessageBox():void

sdl2.ShowSimpleMessageBox(flags:number, title:string, message:string, window:sdl2.Window):void

sdl2.ShowWindow(window:sdl2.Window):void

sdl2.UpdateWindowSurface(window:sdl2.Window):void

sdl2.UpdateWindowSurfaceRects(window:sdl2.Window, rects[]:sdl2.Rect):void sdl2.VideoInit(driver_name:string):void

sdl2.VideoQuit():void

sdl2.CreateRenderer(window:sdl2.Window, index:number, flags:number) {block?}

sdl2.CreateSoftwareRenderer(surface:sdl2.Surface) {block?}

sdl2.CreateTexture(renderer:sdl2.Renderer, format:number, access:number, w:number, h:number) {block?}

sdl2.CreateTextureFromSurface(renderer:sdl2.Renderer, surface:sdl2.Surface) {block?}

sdl2.DestroyRenderer(renderer:sdl2.Renderer):void

sdl2.DestroyTexture(texture:sdl2.Texture):void

sdl2.GL_BindTexture(texture:sdl2.Texture) {block?}

sdl2.GL_UnbindTexture(texture:sdl2.Texture):void

sdl2.GetNumRenderDrivers() {block?}

sdl2.GetRenderDrawBlendMode(renderer:sdl2.Renderer) {block?}

sdl2.GetRenderDrawColor(renderer:sdl2.Renderer) {block?}

sdl2.GetRenderDriverInfo(index:number) {block?}

sdl2.GetRenderTarget(renderer:sdl2.Renderer) {block?}

sdl2.GetRenderer(window:sdl2.Window) {block?}

sdl2.GetRendererInfo(renderer:sdl2.Renderer) {block?}

sdl2.GetRenderOutputSize(renderer:sdl2.Renderer) {block?}

sdl2.GetTextureAlphaMod(texture:sdl2.Texture) {block?}

sdl2.GetTextureBlendMode(texture:sdl2.Texture) {block?}

sdl2.GetTextureColorMod(texture:sdl2.Texture) {block?}

sdl2.LockTexture(texture:sdl2.Texture, rect:sdl2.Rect):void

sdl2.QueryTexture(texture:sdl2.Texture) {block?}

sdl2.RenderClear(renderer:sdl2.Renderer):void

sdl2.RenderCopy(renderer:sdl2.Renderer, texture:sdl2.Texture, srcrect:sdl2.Rect:nil, dstrect:sdl2.Rect:nil

sdl2.RenderCopyEx(renderer:sdl2.Renderer, texture:sdl2.Texture, srcrect:sdl2.Rect:nil, dstrect:sdl2.Rect:

sdl2.RenderDrawLine(renderer:sdl2.Renderer, x1:number, y1:number, x2:number, y2:number):void sdl2.RenderDrawLines(renderer:sdl2.Renderer, points[]:sdl2.Point):void

sdl2.RenderDrawPoint(renderer:sdl2.Renderer, x:number, y:number):void

sdl2.RenderDrawPoints(renderer:sdl2.Renderer, points[]:sdl2.Point):void

sdl2.RenderDrawRect(renderer:sdl2.Renderer, rect:sdl2.Rect:nil):void

sdl2.RenderDrawRects(renderer:sdl2.Renderer, rects[]:sdl2.Rect):void

sdl2.RenderFillRect(renderer:sdl2.Renderer, rect:sdl2.Rect:nil):void

sdl2.RenderFillRects(renderer:sdl2.Renderer, rects[]:sdl2.Rect):void

sdl2.RenderGetClipRect(renderer:sdl2.Renderer) {block?}

sdl2.RenderGetLogicalSize(renderer:sdl2.Renderer) {block?}

sdl2.RenderGetScale(renderer:sdl2.Renderer) {block?}

sdl2.RenderGetViewport(renderer:sdl2.Renderer) {block?}

sdl2.RenderIsClipEnabled(renderer:sdl2.Renderer)

sdl2.RenderPresent(renderer:sdl2.Renderer):void

sdl2.RenderReadPixels(renderer:sdl2.Renderer, rect:sdl2.Rect:nil, format:symbol) {block?}

sdl2.RenderSetClipRect(renderer:sdl2.Renderer, rect:sdl2.Rect:nil):void

sdl2.RenderSetLogicalSize(renderer:sdl2.Renderer, w:number, h:number):void

sdl2.RenderSetScale(renderer:sdl2.Renderer, scaleX:number, scaleY:number):void

sdl2.RenderSetViewport(renderer:sdl2.Renderer, rect:sdl2.Rect:nil):void

sdl2.RenderTargetSupported(renderer:sdl2.Renderer) {block?}

sdl2.SetRenderDrawBlendMode(renderer:sdl2.Renderer, blendMode:number):void

sdl2.SetRenderDrawColor(renderer:sdl2.Renderer, r:number, g:number, b:number, a:number):void

sdl2.SetRenderTarget(renderer:sdl2.Renderer, texture:sdl2.Texture:nil):void

sdl2.SetTextureAlphaMod(texture:sdl2.Texture, alpha:number):void

sdl2.SetTextureBlendMode(texture:sdl2.Texture, blendMode:number):void

sdl2.SetTextureColorMod(texture:sdl2.Texture, r:number, g:number, b:number):void

sdl2.UnlockTexture(texture:sdl2.Texture):void

sdl2.UpdateTexture(texture:sdl2.Texture, rect:sdl2.Rect:nil, pitch:number):void sdl2.UpdateYUVTexture():void

sdl2.AllocFormat(pixel_format:number) {block?}

sdl2.AllocPalette(ncolors:number) {block?}

sdl2.CalculateGammaRamp(gamma:number) {block?}

sdl2.FreeFormat(format:sdl2.PixelFormat):void

sdl2.FreePalette(palette:sdl2.Palette):void

sdl2.GetPixelFormatName(format:number) {block?}

sdl2.GetRGB(pixel:number, format:sdl2.PixelFormat) {block?}

sdl2.GetRGBA(pixel:number, format:sdl2.PixelFormat) {block?}

sdl2.MapRGB(format:sdl2.PixelFormat, r:number, g:number, b:number) {block?}

sdl2.MapRGBA(format:sdl2.PixelFormat, r:number, g:number, b:number, a:number) {block?}

sdl2.MasksToPixelFormatEnum(bpp:number, Rmask:number, Gmask:number, Bmask:number, Amask:number) {block?}

sdl2.PixelFormatEnumToMasks(format:number) {block?}

sdl2.SetPaletteColors(palette:sdl2.Palette, colors[]:sdl2.Color, firstcolor:number, ncolors:number):void

sdl2.SetPixelFormatPalette(format:sdl2.PixelFormat, palette:sdl2.Palette):void

sdl2.EnclosePoints(points[]:sdl2.Point, clip:sdl2.Rect) {block?}

sdl2.HasIntersection(A:sdl2.Rect, B:sdl2.Rect) {block?}

sdl2.IntersectRect(A:sdl2.Rect, B:sdl2.Rect) {block?}

sdl2.IntersectRectAndLine(rect:sdl2.Rect, X1:number, Y1:number, X2:number, Y2:number):void

sdl2.PointInRect(p:sdl2.Point, r:sdl2.Rect):void

sdl2.RectEmpty(r:sdl2.Rect) {block?}

sdl2.RectEquals(a:sdl2.Rect, b:sdl2.Rect) {block?}

sdl2.UnionRect(A:sdl2.Rect, B:sdl2.Rect) {block?}

sdl2.BlitScaled(src:sdl2.Surface, srcrect:sdl2.Rect:nil, dst:sdl2.Surface, dstrect:sdl2.Rect:nil):void

sdl2.BlitSurface(src:sdl2.Surface, srcrect:sdl2.Rect:nil, dst:sdl2.Surface, dstrect:sdl2.Rect:nil):void

sdl2.ConvertPixels(width:number, height:number, src_format:number, dst_format:number):void

sdl2.ConvertSurface(src:sdl2.Surface, fmt:sdl2.PixelFormat, flags:number) {block?}

sdl2.ConvertSurfaceFormat(src:sdl2.Surface, pixel_format:number, flags:number) {block?}

sdl2.CreateRGBSurface(flags:number, width:number, height:number, depth:number, Rmask:number, Gmask:number,

sdl2.CreateRGBSurfaceFrom(pixels:array:nomap, width:number, height:number, depth:number, pitch:number, Rma

sdl2.CreateRGBSurfaceFromImage(image:image) {block?}

sdl2.FillRect(dst:sdl2.Surface, rect:sdl2.Rect:nil, color:number):void

sdl2.FillRects(dst:sdl2.Surface, rects[]:sdl2.Rect, color:number):void

sdl2.FreeSurface(surface:sdl2.Surface):void

sdl2.GetClipRect(surface:sdl2.Surface) {block?}

sdl2.GetColorKey(surface:sdl2.Surface) {block?}

sdl2.GetSurfaceAlphaMod(surface:sdl2.Surface) {block?}

sdl2.GetSurfaceBlendMode(surface:sdl2.Surface) {block?}

sdl2.GetSurfaceColorMod(surface:sdl2.Surface) {block?}

sdl2.LoadBMP(src:stream) {block?}

sdl2.LoadBMP_RW():void

sdl2.LockSurface(surface:sdl2.Surface):void

sdl2.LowerBlit(src:sdl2.Surface, srcrect:sdl2.Rect:nil, dst:sdl2.Surface, dstrect:sdl2.Rect:nil):void

sdl2.LowerBlitScaled(src:sdl2.Surface, srcrect:sdl2.Rect:nil, dst:sdl2.Surface, dstrect:sdl2.Rect:nil):vo

sdl2.MUSTLOCK(surface:sdl2.Surface) {block?}

sdl2.SaveBMP(surface:sdl2.Surface, dst:stream) {block?}

sdl2.SaveBMP_RW():void

sdl2.SetClipRect(surface:sdl2.Surface, rect:sdl2.Rect) {block?}

sdl2.SetColorKey(surface:sdl2.Surface, flag:number, key:number):void

sdl2.SetSurfaceAlphaMod(surface:sdl2.Surface, alpha:number):void

sdl2.SetSurfaceBlendMode(surface:sdl2.Surface, blendMode:number):void

sdl2.SetSurfaceColorMod(surface:sdl2.Surface, r:number, g:number, b:number):void

sdl2.SetSurfacePalette(surface:sdl2.Surface, palette:sdl2.Palette):void

sdl2.SetSurfaceRLE(surface:sdl2.Surface, flag:number):void sdl2.UnlockSurface(surface:sdl2.Surface):void

sdl2.GetClipboardText() {block?}

sdl2.HasClipboardText() {block?}

sdl2.SetClipboardText(text:string):void

sdl2.AddEventWatch():void

sdl2.DelEventWatch():void

sdl2.EventState(type:number, state:number) {block?}

sdl2.FilterEvents():void

sdl2.FlushEvent(type:number):void

sdl2.FlushEvents(minType:number, maxType:number):void

sdl2.GetEventFilter():void

sdl2.GetNumTouchDevices() {block?}

sdl2.GetNumTouchFingers(touchId:number) {block?}

sdl2.GetTouchDevice(index:number) {block?}

sdl2.GetTouchFinger(touchId:number, index:number) {block?}

sdl2.HasEvent(type:number) {block?}

sdl2.HasEvents(minType:number, maxType:number) {block?}

sdl2.LoadDollarTemplates(touchId:number, src:stream) {block?}

sdl2.AddEvents(events[]:sdl2.Event) {block?}

sdl2.PeekEvents(numevents:number, minType:number, maxType:number) {block?}

sdl2.GetEvents(numevents:number, minType:number, maxType:number) {block?}

sdl2.PollEvent() {block?}

sdl2.PumpEvents():void

sdl2.PushEvent(event:sdl2.Event) {block?}

sdl2.QuitRequested() {block?}

sdl2.RecordGesture(touchId:number) {block?}

sdl2.RegisterEvents(numevents:number) {block?}

sdl2.SaveAllDollarTemplates(dst:stream) {block?}

sdl2.SaveDollarTemplate(gestureId:number, dst:stream):void

sdl2.SetEventFilter():void

sdl2.WaitEvent() {block?}

sdl2.WaitEventTimeout(timeout:number) {block?}

sdl2.CheckKeyboardState(scancode:number) {block?}

sdl2.GetKeyFromName(name:string) {block?}

sdl2.GetKeyFromScancode(scancode:number) {block?}

sdl2.GetKeyName(key:number) {block?}

sdl2.GetKeyboardFocus() {block?}

sdl2.GetKeyboardState() {block?}

sdl2.GetModState() {block?}

sdl2.GetScancodeFromKey(key:number) {block?}

sdl2.GetScancodeFromName(name:string) {block?}

sdl2.GetScancodeName(scancode:number) {block?}

sdl2.HasScreenKeyboardSupport() {block?}

sdl2.IsScreenKeyboardShown(window:sdl2.Window) {block?}

sdl2.IsTextInputActive() {block?}

sdl2.SetModState(modstate:number):void

sdl2.SetTextInputRect(rect:sdl2.Rect):void

sdl2.StartTextInput():void

sdl2.StopTextInput():void

sdl2.CaptureMouse(enalbed:boolean):void

sdl2.CreateColorCursor(surface:sdl2.Surface, hot_x:number, hot_y:number) {block?}

sdl2.CreateCursor(data:array@uint8:nomap, mask:array@uint8:nomap, w:number, h:number, hot_x:number, hot_y:

sdl2.CreateSystemCursor(id:number) {block?}

sdl2.FreeCursor(cursor:sdl2.Cursor):void sdl2.GetCursor() {block?}

sdl2.GetDefaultCursor() {block?}

sdl2.GetGlobalMouseState():void

sdl2.GetMouseFocus() {block?}

sdl2.GetMouseState() {block?}

sdl2.GetRelativeMouseMode() {block?}

sdl2.GetRelativeMouseState() {block?}

sdl2.SetCursor(cursor:sdl2.Cursor):void

sdl2.SetRelativeMouseMode(enabled:boolean):void

sdl2.ShowCursor(toggle:number):void

sdl2.WarpMouseGlobal(x:number, y:number):void

sdl2.WarpMouseInWindow(window:sdl2.Window, x:number, y:number):void

sdl2.JoystickClose(joystick:sdl2.Joystick):void

sdl2.JoystickEventState(state:number) {block?}

sdl2.JoystickGetAttached(joystick:sdl2.Joystick) {block?}

sdl2.JoystickGetAxis(joystick:sdl2.Joystick, axis:number) {block?}

sdl2.JoystickGetBall(joystick:sdl2.Joystick, ball:number) {block?}

sdl2.JoystickGetButton(joystick:sdl2.Joystick, button:number) {block?}

sdl2.JoystickGetDeviceGUID(device_index:number) {block?}

sdl2.JoystickGetGUID(joystick:sdl2.Joystick) {block?}

sdl2.JoystickGetGUIDFromString(pchGUID:string) {block?}

sdl2.JoystickGetGUIDString(guid:sdl2.JoystickGUID) {block?}

sdl2.JoystickGetHat(joystick:sdl2.Joystick, hat:number) {block?}

sdl2.JoystickInstanceID(joystick:sdl2.Joystick) {block?}

sdl2.JoystickName(joystick:sdl2.Joystick) {block?}

sdl2.JoystickNameForIndex(device_index:number) {block?}

sdl2.JoystickNumAxes(joystick:sdl2.Joystick) {block?}

sdl2.JoystickNumBalls(joystick:sdl2.Joystick) {block?}

sdl2.JoystickNumButtons(joystick:sdl2.Joystick) {block?}

sdl2.JoystickNumHats(joystick:sdl2.Joystick) {block?}

sdl2.JoystickOpen(device_index:number) {block?}

sdl2.JoystickUpdate():void

sdl2.NumJoysticks() {block?}

sdl2.GameControllerAddMapping(mappingString:string) {block?}

sdl2.GameControllerAddMappingsFromFile(file:stream) {block?}

sdl2.GameControllerAddMappingsFromRW():void

sdl2.GameControllerClose(gamecontroller:sdl2.GameController):void

sdl2.GameControllerEventState(state:number) {block?}

sdl2.GameControllerGetAttached(gamecontroller:sdl2.GameController) {block?}

sdl2.GameControllerGetAxis(gamecontroller:sdl2.GameController, axis:number) {block?}

sdl2.GameControllerGetAxisFromString(pchString:string) {block?}

sdl2.GameControllerGetBindForAxis(gamecontroller:sdl2.GameController, axis:number) {block?}

sdl2.GameControllerGetBindForButton(gamecontroller:sdl2.GameController, button:number) {block?}

sdl2.GameControllerGetButton(gamecontroller:sdl2.GameController, button:number) {block?}

sdl2.GameControllerGetButtonFromString(pchString:string) {block?}

sdl2.GameControllerGetJoystick(gamecontroller:sdl2.GameController) {block?}

sdl2.GameControllerGetStringForAxis(axis:number) {block?}

sdl2.GameControllerGetStringForButton(button:number) {block?}

sdl2.GameControllerMapping(gamecontroller:sdl2.GameController) {block?}

sdl2.GameControllerMappingForGUID(guid:sdl2.JoystickGUID) {block?}

sdl2.GameControllerName(gamecontroller:sdl2.GameController) {block?}

sdl2.GameControllerNameForIndex(joystick_index:number) {block?}

sdl2.GameControllerOpen(joystick_index:number) {block?}

sdl2.GameControllerUpdate():void

sdl2.IsGameController(joystick_index:number) {block?}

sdl2.HapticClose(haptic:sdl2.Haptic):void

sdl2.HapticDestroyEffect(haptic:sdl2.Haptic, effect:number):void

sdl2.HapticEffectSupported(haptic:sdl2.Haptic, effect:sdl2.HapticEffect) {block?}

sdl2.HapticGetEffectStatus(haptic:sdl2.Haptic, effect:number) {block?}

sdl2.HapticIndex(haptic:sdl2.Haptic) {block?}

sdl2.HapticName(device_index:number) {block?}

sdl2.HapticNewEffect(haptic:sdl2.Haptic, effect:sdl2.HapticEffect) {block?}

sdl2.HapticNumAxes(haptic:sdl2.Haptic) {block?}

sdl2.HapticNumEffects(haptic:sdl2.Haptic) {block?}

sdl2.HapticNumEffectsPlaying(haptic:sdl2.Haptic) {block?}

sdl2.HapticOpen(device_index:number) {block?}

sdl2.HapticOpenFromJoystick(joystick:sdl2.Joystick) {block?}

sdl2.HapticOpenFromMouse() {block?}

sdl2.HapticOpened(device_index:number) {block?}

sdl2.HapticPause(haptic:sdl2.Haptic):void

sdl2.HapticQuery(haptic:sdl2.Haptic) {block?}

sdl2.HapticRumbleInit(haptic:sdl2.Haptic):void

sdl2.HapticRumblePlay(haptic:sdl2.Haptic, strength:number, length:number):void

sdl2.HapticRumbleStop(haptic:sdl2.Haptic):void

sdl2.HapticRumbleSupported(haptic:sdl2.Haptic) {block?}

sdl2.HapticRunEffect(haptic:sdl2.Haptic, effect:number, iterations:number):void

sdl2.HapticSetAutocenter(haptic:sdl2.Haptic, autocenter:number):void

sdl2.HapticSetGain(haptic:sdl2.Haptic, gain:number):void

sdl2.HapticStopAll(haptic:sdl2.Haptic):void

sdl2.HapticStopEffect(haptic:sdl2.Haptic, effect:number):void

sdl2.HapticUnpause(haptic:sdl2.Haptic):void sdl2.HapticUpdateEffect(haptic:sdl2.Haptic, effect:number, data:sdl2.HapticEffect):void

sdl2.JoystickIsHaptic(joystick:sdl2.Joystick) {block?}

sdl2.MouseIsHaptic() {block?}

sdl2.NumHaptics() {block?}

sdl2.AudioInit(driver_name:string):void

sdl2.AudioQuit():void

sdl2.BuildAudioCVT(cvt:sdl2.AudioCVT, src_format:number, src_channels:number, src_rate:number, dst_format:r

sdl2.ClearQueuedAudio(dev:number):void

sdl2.CloseAudio():void

sdl2.CloseAudioDevice(dev:number):void

sdl2.ConvertAudio(cvt:sdl2.AudioCVT):void

sdl2.FreeWAV(wav:sdl2.Wav):void

sdl2.GetAudioDeviceName(index:number, iscapture:number) {block?}

sdl2.GetAudioDeviceStatus(dev:number) {block?}

sdl2.GetAudioDriver(index:number) {block?}

sdl2.GetAudioStatus() {block?}

sdl2.GetCurrentAudioDriver() {block?}

sdl2.GetNumAudioDevices(iscapture:number) {block?}

sdl2.GetNumAudioDrivers() {block?}

sdl2.GetQueuedAudioSize(dev:number):void

sdl2.LoadWAV(file:stream) {block?}

sdl2.LoadWAV_RW():void

sdl2.LockAudio():void

sdl2.LockAudioDevice(dev:number):void

sdl2.MixAudio(volume:number):void

sdl2.MixAudioFormat(format:number, volume:number):void

sdl2.OpenAudio(desired:sdl2.AudioSpec) {block?}

sdl2.OpenAudioDevice(device:string, iscapture:number, desired:sdl2.AudioSpec, allowed_changes:number):void

sdl2.PauseAudio(pause_on:number):void

sdl2.PauseAudioDevice(dev:number, pause_on:number):void

sdl2.QueueAudio(dev:number):void

sdl2.UnlockAudio():void

sdl2.UnlockAudioDevice(dev:number):void

sdl2.AUDIO_BITSIZE(x:number) {block?}

sdl2.AUDIO_ISFLOAT(x:number) {block?}

sdl2.AUDIO_ISBIGENDIAN(x:number) {block?}

sdl2.AUDIO_ISSIGNED(x:number) {block?}

sdl2.AUDIO_ISINT(x:number) {block?}

sdl2.AUDIO_ISLITTLEENDIAN(x:number) {block?}

sdl2.AUDIO_ISUNSIGNED(x:number) {block?}

sdl2.CreateThread():void

sdl2.DetachThread():void

sdl2.GetThreadID():void

sdl2.GetThreadName():void

sdl2.GetThreadPriority():void

sdl2.TLSCreate():void

sdl2.TLSGet():void

sdl2.TLSSet():void

sdl2.ThreadID():void

sdl2.WaitThread():void

sdl2.CondBroadcast():void

sdl2.CondSignal():void

sdl2.CondWait():void

sdl2.CondWaitTimeout():void

sdl2.CreateCond():void

sdl2.CreateMutex():void

sdl2.CreateSemaphore():void

sdl2.DestroyCond():void

sdl2.DestroyMutex():void

sdl2.DestroySemaphore():void

sdl2.LockMutex():void

sdl2.SemPost():void

sdl2.SemTryWait():void

sdl2.SemValue():void

sdl2.SemWait():void

sdl2.SemWaitTimeout():void

sdl2.TryLockMutex():void

sdl2.UnlockMutex():void

sdl2.AtomicAdd():void

sdl2.AtomicCAS():void

sdl2.AtomicCASPtr():void

sdl2.AtomicDecRef():void

sdl2.AtomicGet():void

sdl2.AtomicGetPtr():void

sdl2.AtomicIncRef():void

sdl2.AtomicLock():void

sdl2.AtomicSet():void

sdl2.AtomicSetPtr():void

sdl2.AtomicTryLock():void

sdl2.AtomicUnlock():void

sdl2.CompilerBarrier():void

sdl2.AddTimer(interval:number):void

sdl2.Delay(ms:number):void

sdl2.GetPerformanceCounter() {block?}

sdl2.GetPerformanceFrequency() {block?}

sdl2.GetTicks() {block?}

sdl2.RemoveTimer(id:number) {block?}

sdl2.TICKS_PASSED(A:number, B:number) {block?}

sdl2.GetBasePath():void

sdl2.GetPrefPath(org:string, app:string):void

sdl2.AllocRW():void

sdl2.FreeRW():void

sdl2.RWFromConstMem():void

sdl2.RWFromFP():void

sdl2.RWFromFile():void

sdl2.RWFromMem():void

sdl2.RWclose():void

sdl2.RWread():void

sdl2.RWseek():void

sdl2.RWtell():void

sdl2.RWwrite():void

sdl2.ReadBE16():void

sdl2.ReadBE32():void

sdl2.ReadBE64():void

sdl2.ReadLE16():void

sdl2.ReadLE32():void

sdl2.ReadLE64():void

sdl2.WriteBE16():void

sdl2.WriteBE32():void

sdl2.WriteBE64():void

sdl2.WriteLE16():void

sdl2.WriteLE32():void

sdl2.WriteLE64():void

sdl2.GetPlatform() {block?}

sdl2.GetCPUCacheLineSize() {block?}

sdl2.GetCPUCount() {block?}

sdl2.GetSystemRAM() {block?}

sdl2.Has3DNow() {block?}

sdl2.HasAVX() {block?}

sdl2.HasAVX2():void

sdl2.HasAltiVec() {block?}

sdl2.HasMMX() {block?}

sdl2.HasRDTSC() {block?}

sdl2.HasSSE() {block?}

sdl2.HasSSE2() {block?}

sdl2.HasSSE3() {block?}

sdl2.HasSSE41() {block?}

sdl2.HasSSE42() {block?}

sdl2.Swap16():void

sdl2.Swap32():void

sdl2.Swap64():void

sdl2.SwapBE16():void

sdl2.SwapBE32():void

sdl2.SwapBE64():void

sdl2.SwapFloat():void

sdl2.SwapFloatBE():void

sdl2.SwapFloatLE():void

sdl2.SwapLE16():void

sdl2.SwapLE32():void

sdl2.SwapLE64():void

sdl2.MostSignificantBitIndex32(x:number):void

sdl2.GetPowerInfo() {block?}

sdl2.AndroidGetActivity():void

sdl2.AndroidGetExternalStoragePath():void

sdl2.AndroidGetExternalStorageState():void

sdl2.AndroidGetInternalStoragePath():void

sdl2.AndroidGetJNIEnv():void

sdl2.acos(x:number) {block?}

45.2 sdl2.Window Class

45.3 sdl2.Renderer Class

45.4 sdl2.Texture Class

45.5 sdl2.Event Class

45.6 sdl2.Point Class

45.7 sdl2.Rect Class

45.8 sdl2.Color Class

45.9 sdl2.Palette Class

45.10 sdl2.PixelFormat Class

45.11 sdl2.Keysym Class

45.12 sdl2.Cursor Class

45.13 sdl2.Joystick Class

45.14 sdl2.JoystickGUID Class

45.15 sdl2.GameController Class

45.16 sdl2.GameControllerButtonBind Class

45.17 sdl2.AudioCVT Class

45.18 sdl2.AudioSpec Class

45.19 sdl2.Wav Class

45.20 sdl2.RendererInfo Class

45.21 sdl2.DisplayMode Class

45.22 sdl2.GLContext Class

45.23 sdl2.HapticEffect Class

45.24 sdl2.Surface Class

http://www.libsdl.org/

Chapter 46

sqlite3 Module

The `sqlite3` module provides measures to access SQLite3 database. To utilize it, import the `sqlite3` module using `import` function.

46.1 sqlite3.db Class

46.1.1 Constructor

`sqlite3.db(filename:string) {block?}`

Opens an sqlite3 database file and returns a connection handle with the database.

If `block` is specified, it would be evaluated with a block parameter `|db:sqlite3|`, where `db` is the created instance. In this case, the block's result would become the function's returned value. The connection handle will be automatically closed when the block finishes.

46.1.2 Method

`sqlite3.db#close()`

Shuts down the connection with an sqlite3 server.

`sqlite3.db#exec(sql:string):map`

Executes an SQL statement and creates an `list` that has `list` instances containing queried result as its elements.

`sqlite3.db#getcolnames(sql:string):map {block?}`

`sqlite3.db#query(sql:string):map {block?}`

Executes an SQL statement and creates an `iterator` that returns `list` instances containing queried result as its elements.

You should use `sqlite3.db#query()` instead of `sqlite3.db#exec()` when it's likely that you get a large size of data as the result.

`sqlite3.db#transaction() {block}`

Executes the block within a transaction. The process is like following:

1. Executes a sqlit3 command 'BEGIN TRANSACTION'.

2. Executes code in the `block`.
3. Executes a sqlite3 command 'END TRANSACTION'.

46.2 Thanks

This module uses SQlite3 library which is distributed in the following site:
http://www.sqlite.org/index.html

Chapter 47

sys Module

The `sys` module provides system-related information. This is a built-in module, so you can use it without being imported.

47.1 Module Variable

- `sys.argv`
- `sys.path`
- `sys.maindir`
- `sys.version`
- `sys.banner`
- `sys.timestamp`
- `sys.build`
- `sys.platform`
- `sys.ps1`
- `sys.ps2`
- `sys.langcode`
- `sys.executable`
- `sys.incdir`
- `sys.libdir`
- `sys.datadir`
- `sys.moddir`
- `sys.localdir`
- `sys.appdir`
- `sys.cfgdir`
- `sys.workdir`

47.2 Module Function

`sys.echo(flag:boolean)`

Enables or disables echo-back functionality according to flag.

`sys.exit(status?:number)`

Terminates the program with a specified status number.

`sys.interactive()`

Enters to interactive mode.

`sys.required_version(major:number, minor:number, patch:number)`

Raises an error if the running interpreter doesn't satisfy the required version.

Chapter 48

tar Module

The `tar` module provides measures to read/write TAR files. To utilize it, import the `tar` module using `import` function.

48.1 tar.reader Class

48.1.1 Function To Create Instance

`tar.reader(stream:stream:r, compression?:symbol) {block?}`

Reads a tar file from `stream` and returns a `tar.reader` instance that is to be used to read contents from the archive.

The argument `compression` specifies the compression format of the tar file and takes one of the following symbols:

- `‘auto` .. determins the format from a suffix name of the stream.
- `‘gzip` .. gzip format
- `‘bzip2` .. bzip2 format

48.1.2 Method

`tar.reader#entries() {block?}`

Creates an iterator that returns stream instances for each entry in the tar file.

48.2 tar.writer Class

48.2.1 Function To Create Instance

`tar.writer(stream:stream:w, compression?:symbol) {block?}`

Creates a tar file on `stream` and returns a `tar.writer` instance that is to be used to write contents to the archive.

The argument `compression` specifies the compression format of the tar file and takes one of the following symbols:

- `'auto` .. determins the format from a suffix name of the stream.
- `'gzip` .. gzip format
- `'bzip2` .. bzip2 format

48.2.2 Method

`tar.writer#add(stream:stream:r, filename?:string):map:reduce`

Adds an entry to the tar archive with a content from `stream` and a name of `filename`.

If the argument `filename` is omitted, an identifier associated with the `stream` would be used as the entry name.

`tar.writer#close():reduce`

Flushes all the unfinished writing processes and invalidates the `tar.writer` instance.

48.3 Thanks

This module uses zlib and bzip2 library which are distributed in the following sites:

- http://zlib.net/
- http://www.bzip.org/

Chapter 49

tiff Module

The `tiff` module provides measures to read/write image data in TIFF format. To utilize it, import the `tiff` module using `import` function.

Below is an example to read a TIFF file:

```
import(tiff)
img = image('foo.tiff')
```

49.1 Exntension to Function's Capability

This module extends the capability of function `image()` and instance method `image#write()` so that they can read/write TIFF files.

When function `image()` is provided with a stream that satisfies the following conditions, it would recognize the stream as a TIFF file.

- The identifier of the stream ends with a suffix ".tif" or ".tiff".

When instance method `image#write()` is provided with a stream that satisfies the following condition, it would write image data in TIFF format.

- The identifier of the stream ends with a suffix ".tif" or ".tiff".

49.2 Extension to image Class

This module extends the `image` class with methods described here.

`image#read@tiff(stream:stream:r):reduce`

Reads a TIFF image from a stream.

49.3 Thanks

This module uses libtiff which is distributed in the following site:

http://www.libtiff.org/

Chapter 50

tokenizer Module

The `tokenizer` module ...

50.1 Module Function

Chapter 51

units Module

The `units` module provides functions to convert physical units into another.

51.1 Module Function

`units.inch$mm(inch:number):map`

Converts **inch** to **mm**.

`units.mm$inch(mm:number):map`

Converts **mm** to **inch**.

`units.mm$pt(mm:number):map`

Converts **mm** to **pt**.

`units.pt$mm(pt:number):map`

Converts **pt** to **mm**.

Chapter 52

uuid Module

The `uuid` module provides functions to generate UUIDs. To utilize it, import the `uuid` module using `import` function.

52.1 Module Function

`uuid.generate():[upper]`

Generates a Universal Unique Identifier (UUID). In default, results are output in lower-case characters. Specifying :upper would generates it in upper-case characters.

Chapter 53

wav Module

53.1 Module Function

53.2 Extension to audio Class

This module extends the `audio` class with methods described here.

`audio#read@wav(stream:stream:r):reduce`

Reads WAV audio from a stream.

`audio#write@wav(stream:stream:w):reduce`

Writes WAV audio to a stream.

Chapter 54

wx Module

The `wx` module provides functions and methods of wxWidgets library.

54.1 Module Function

54.2 Thanks

This module uses wxWidgets library which is distributed in the following site:
http://www.wxwidgets.org/

Chapter 55

xml Module

The `xml` module provides measures to parse or compose XML documents.

There are two ways to parse an XML document as follows.

One is to create an `xml.document` instance from a stream that contains all the XML elements with a tree structure. This is an easy way to parse an XML document but consumes much memory. Below is an example to read an XML file `test.xml`:

```
doc = xml.document('test.xml')
// doc contains all the information of XML document
```

Another one is to create a class inherited `xml.parser` and implements event handlers that respond to tags, comments and texts, and then executes `xml.parser#parse()` method with it. Below is an example to create a class that implements a handler for StartElement event:

```
Parser = class(xml.parser) {
    StartElement(elem) = {
        printf('<%s>\n', elem.tagname)
    }
}
Parser().parse('test.xml')
```

55.1 xml.attribute Class

The `xml.attribute` instance represents a name-value pair of XML's attribute that can be retrieved from `attrs` property in the `xml.element` instance.

55.1.1 Property

Property	Type	R/W	Explanation
`name`	`string`	R	
`value`	`string`	R	

55.2 xml.document Class

55.2.1 Constructor

`xml.document(stream?:stream:r) {block?}`

55.2.2 Property

Property	Type	R/W	Explanation
version	string	R	
encoding	string	R	
root	xml.element	R	

55.2.3 Method

`xml.document#parse(str:string):void`

`xml.document#read(stream:stream:r):void`

`xml.document#textize(fancy?:boolean, tabs?:number)`

`xml.document#write(stream:stream:w, fancy?:boolean, tabs?:number):void`

55.3 xml.element Class

55.3.1 Constructor

`xml.element(_tagname_:string, attrs%):map {block?}`

`xml.comment(comment:string)`

55.3.2 Property

Property	Type	R/W	Explanation
tagname	string	R	A tag name of this element.
text	string	R	The text string if the element is TEXT. Otherwise, this value would be `nil`.
comment	string	R	The comment string if the element is COMMENT. Otherwise, this value would be `nil`.
children	iterator	R	An iterator to return `xml.element` instances that represent children contained in this element. This value would be `nil` if the element has no children.
attrs	iterator	R	An iterator to return `xml.attribute` instances that represent attributes contained in this element. This value would be `nil` if the element has no attributes.

55.3.3 Method

`xml.element#addchild(value):map:void`

`xml.element#gettext()`

`xml.element#textize(fancy?:boolean, indentLevel?:number, tabs?:number)`

`xml.element#write(stream:stream:w, fancy?:boolean, indentLevel?:number, tabs?:number):void`

55.4 xml.parser Class

The `xml.parser` class is a base class from which you can implement a inheritance class that has methods corresponding to events associated with XML elements. Below are methods that you can implement in the class for event handling:

- `StartElement(elem:xml.element)`
- `EndElement(name:string)`
- `CharacterData(text:string)`
- `ProcessingInstruction(target:string, data:string)`
- `Comment(data:string)`
- `StartCdataSection()`
- `EndCdataSection()`
- `Default(text:string)`
- `DefaultExpand(text:string)`
- `ExternalEntityRef()`
- `SkippedEntity(entityName:string, isParameterEntity:boolean)`
- `StartNamespaceDecl(prefix:string, uri:string)`
- `EndNamespaceDecl(prefix:string)`
- `XmlDecl(version:string, encoding:string, standalone:boolean)`
- `StartDoctypeDecl(doctypeName:strng, systemId:string, publicId:string, hasInternalSubset:bo`
- `EndDoctypeDecl()`
- `ElementDecl()`
- `AttlistDecl(elemName:string, attName:string, attType:string, defaultValue:string, isRequired:boolean)`
- `EntityDecl(entityName:string, isParameterEntity:boolean, value:string, base:string, systemId:string, publicId:string, notationName:string)`
- `NotationDecl(notationName:string, base:string, systemId:string, publicId:string)`
- `NotStandalone()`

55.4.1 Constructor

`xml.parser() {block?}`

55.4.2 Method

`xml.parser#parse(stream:stream:r):void`

55.5 Thanks

This module uses expat library which is distributed in the following site:
http://expat.sourceforge.net/

Chapter 56

xpm Module

The `xpm` module provides measures to write image data in XPM format and to parse a list of strings that is described in the format. To utilize it, import the `xpm` module using `import` function.

Below is an example to parse a list of strings described in XPM format.

```
import(xpm)
foo_xpm = @{
"13 13 2 2 0 0",
"   c #000000",
"#  c #ffffff",
"          #               ",
"          #               ",
"  # # # # # # # # # #     ",
"          #               ",
"          #     #         ",
"        # # # # # #       ",
"      #   #     #   #     ",
"    #     #   #       #   ",
"  #       #   #       #   ",
"  #       #   #       #   ",
"  #       # #         #   ",
"    # # #           #     ",
"                # #       ",
}
img = image(`rgba).xpmdata(foo_xpm)
```

56.1 Extension to image Class

This module extends the `image` class with methods described here.

`image#write@xpm(stream:stream:w):reduce`

Writes a xpm image to a stream.

`image#xpmdata(xpm[]:string):reduce`

Read xpm data from a string list.

Chapter 57

yaml Module

The `yaml` module provides measures to read/write YAML files. You can use this module as a measure to serialize and deserialize objects that consists of `list`, `dict` and `string` instances.

Below is an example to reconstruct values from YAML text:

```
txt = '''
key1:
  - item-A
  - item-B
  - item-C
key2:
  - item-D
  - item-E
  - item-F
'''
x = yaml.parse(txt)
// x has the following value:
// %{
//   'key1' => ['item-A', 'item-B', 'item-C']
//   'key2' => ['item-D', 'item-E', 'item-F']
// }
```

57.1 Correspondance of Data Object

The below table shows how YAML data types correspond to Gura's value types each other:

YAML Data Type	Gura's Value Type
sequence	`list`
mapping	`dict`
scalar	`string`

57.2 Module Function

`yaml.compose(obj)`

Composes YAML text to represent the content of `obj` that consists of `list`, `dict` and `string` instances.

`yaml.parse(str:string)`

Parses YAML text in `str` and returns a composition of `list`, `dict` and `string` instances.

`yaml.read(stream:stream:r)`

Parses YAML text from `stream` and returns a composition of `list`, `dict` and `string` instances.

`yaml.write(stream:stream:w, obj):reduce`

Composes YAML text to represent the content of `obj` that consists of `list`, `dict` and `string` instances and writes the result to `stream`.

57.3 Thanks

This module uses yaml library which is distributed in the following site:

http://pyyaml.org/wiki/LibYAML

Chapter 58

zip Module

The `zip` module provides measures to read/write ZIP files.

58.1 zip.reader Class

The `zip.reader` class provides methods to read contents and to get information in a ZIP file through `stream` instance. An instance of `stream` class created by the methods includes a property named `stat`, a `zip.stat` instance, which provides information such as filename and created time stamp that are contained in the ZIP file.

Below is an example to list filenames in a ZIP file:

```
import(zip)
zip.reader('foo.zip') {|r|
    println(r.entries():*stat:*filename)
}
```

Below is an example to print a content of a text file that is stored in a ZIP file:

```
import(zip)
zip.reader('foo.zip') {|r|
    print(r.entry('README.txt').readlines())
}
```

58.1.1 Constructor

`zip.reader(stream:stream:r) {block?}`

Creates `zip.reader` instance from the specified stream.

If `block` is specified, it would be evaluated with a block parameter `|reader:zip.reader|`, where `reader` is the created instance. In this case, the block's result would become the function's returned value.

58.1.2 Method

`zip.reader#entry(name:string) {block?}`

Seeks entry in the zip file that matches the specified name and returns a `stream` instance associated with the entry.

If `block` is specified, it would be evaluated with a block parameter `|s:stream|`, where `s` is the created instance. In this case, the block's result would become the function's returned value.

`zip.reader#entries() {block?}`

Creates an `iterator` instance that returns `stream` instances associated with each entry in the ZIP file.

In default, this returns an iterator as its result value. Specifying the following attributes would customize the returned value:

- `:iter` .. An iterator. This is the default behavior.
- `:xiter` .. An iterator that eliminates `nil` from its elements.
- `:list` .. A list.
- `:xlist` .. A list that eliminates `nil` from its elements.
- `:set` .. A list that eliminates duplicated values from its elements.
- `:xset` .. A list that eliminates duplicated values and `nil` from its elements.

See the chapter of Mapping Process in Gura Language Manual for the detail.

If a block is specified, it would be evaluated repeatingly with block parameters `|value, idx:number|` where `value` is the iterated value and `idx` the loop index starting from zero. In this case, the last evaluated value of the block would be the result value. If one of the attributes listed above is specified, an iterator or a list of the evaluated value would be returned.

58.2 zip.writer Class

The `zip.writer` class provides methods to add entries to a ZIP file. When an instance of `zip.writer` is created, a new ZIP file would be created.

Below is an exapmple to create a ZIP archive file that contains three entries:

```
import(zip)
zip.writer('foo.zip') {|w|
    w.add('file1.txt')
    w.add('file2.txt')
    w.add('file3.txt')
    w.close()
}
```

58.2.1 Constructor

`zip.writer(stream:stream:w, compression?:symbol) {block?}`

Creates `zip.writer` instance from the stream.

Argument `compression` specifies the compression method and takes one of the following symbol.

- `` `store ``
- `` `deflate ``

- `bzip2

If `block` is specified, it would be evaluated with a block parameter `|writer:zip.writer|`, where `writer` is the created instance. In this case, the block's result would become the function's returned value.

58.2.2 Method

`zip.writer#add(stream:stream:r, filename?:string, compression?:symbol):map:reduce`

Reads data from `stream` and adds it to the zip file. Entry name is decided by the file name associated with the stream unless it's specified by argument `filename`.

Argument `compression` specifies the compression method and takes one of the following symbol.

- `store
- `deflate
- `bzip2

`zip.writer#close():void`

Closes the zip file after flushing cached data.

58.3 zip.stat Class

The `zip.stat` class provides information of entries in a ZIP file.

58.3.1 Property

Property	Type	R/W	Explanation
filename	string	R	
comment	string	R	
mtime	datetime	R	
crc32	number	R	
compression_method	number	R	
size	number	R	
compressed_size	number	R	
attributes	number	R	

58.4 Thanks

This module uses zlib and bzip2 library which are distributed in the following sites:

- http://zlib.net/
- http://www.bzip.org/