RingoJS API 0.11

Modules: Ringo Modules

assert

Assertion library for unit testing. It implements the CommonJS Unit Testing specification and adds some additional convenience methods.

binary

When dealing with network sockets or binary files, it’s necessary to read and write into byte streams. JavaScript itself does not provide a native representation of binary data, so this module provides two classes addressing this shortcoming. The implementation follows the CommonJS Binary/B proposal.

ByteArray implements a modifiable and resizable byte buffer.

ByteString implements an immutable byte sequence.

Both classes share a common base class Binary. The base class can't be instantiated. It exists only to affirm that ByteString and ByteArray instances of Binary.

When passed to a Java method that expects a byte[], instances of these classes are automatically unwrapped.

console

This module provides functions to write on the standard error stream stderr for error logging and quick debugging. It’s similar to the console object implemented in most web browsers.

fs

This module provides a file system API for the manipulation of paths, directories, files, links, and the construction of input and output streams. It follows the CommonJS Filesystem/A proposal.

Some file system manipulations use a wrapper around standard POSIX functions. Their functionality depends on the concrete file system and operating system. Others use the java.io package and work cross-platform.

globals

RingoJS adopts some of the global properties from the Rhino shell and adds a few of its own.

Note that this module must and can not be imported like an ordinary module. It is evaluated only once upon RingoJS startup.

io

This module provides functions for reading and writing streams of raw bytes. It implements the Stream and TextStream classes as per the CommonJS IO/A proposal.

Streams are closely related with two other modules. Low-level byte manipulation is provided by the binary module and uses the ByteArray or ByteString class. The fs module returns io streams for reading and writing files.

net

This module provides support for networking using TCP and UDP sockets. A socket represents a connection between a client and a server program over a network. The underlying native binding is provided by the java.net package.

system

This module provides an implementation of the system module compliant to the CommonJS System/1.0 specification. Beyond the standard a print() function is provided.

test

A test runner compliant to the CommonJS Unit Testing specification. It manages the execution of unit tests and processes test results. The runner reports the total number of failures as exit status code.

The runner treats a module like a test case. A test case defines the fixture to run multiple tests. Test cases can provide optional `setUp()` and `tearDown()` functions to initialize and destroy the fixture. The test runner will run these methods prior to / after each test.

The following example test case `testDatabase.js` starts a new test runner if executed with `ringo testDatabase.js`

ringo/args

A parser for command line options. This parser supports various option formats:

- `-a -b -c` (multiple short options)
- `-abc` (multiple short options combined into one)
- `-a value` (short option with value)
- `-avalue` (alternative short option with value)
- `--option value` (long option with value)
- `--option=value` (alternative long option with value)

ringo/base64

Base64 encoding and decoding for binary data and strings.

ringo/buffer

A simple text Buffer class for composing strings.

ringo/concurrent

Utilities for working with multiple concurrently running threads.

ringo/daemon

The daemon control script invoked by the init script.

This module interprets the first command line argument as module ID, load the module and try to invoke the life cycle functions on it.

For HTTP servers it is generally more convenient to directly use ringo/httpserver which will create a new server instance and pass it to as argument to the application life cycle functions.

ringo/encoding

Low-level support for character encoding and decoding.

ringo/engine

Provides access to the Rhino JavaScript engine.

ringo/events

Exports an EventEmitter classes that provide methods to emit events and register event listener functions.

ringo/httpclient

A module for sending HTTP requests and receiving HTTP responses.

ringo/httpserver

A wrapper for the Jetty HTTP server.

ringo/jsdoc

Low level support for parsing JSDoc-style comments from JavaScript files.

ringo/logging

This module provides generic logging support for Ringo applications. It uses SLF4J or Apache log4j if either is detected in the classpath, and will fall back to java.util.logging otherwise.

If the first argument passed to any of the logging methods is a string containing any number of curly bracket pairs ({}), the logger will interpret it as format string and use any following arguments to replace the curly bracket pairs. If an argument is an Error or Java Exception object, the logger will render a stack trace for it and append it to the log message.

This module's exports object implements the EventEmitter interface and emits logged messages using the log level name as event type.

ringo/markdown

A fast and extensible Markdown formatter.

ringo/mime

This module provides functionality for determining the MIME type for a given file extension.

ringo/mustache

CommonJS-compatible mustache.js module.

This version of mustache.js adds filters. If a tag or section name consists of several space-separated items, the items are evaluated one at a time, starting with the right-most item. If an item evaluates to a function, the result of the previous item is passed to it as argument.

ringo/parser

This module provides an interface to the Rhino parser.

ringo/profiler

A profiler for measuring execution time of JavaScript functions. Note that you need to run with optimization level -1 for profiling to work. Running the profiler on optimized code will produce no data.

ringo/promise

Allows to work with deferred values that will be resolved in the future.

ringo/shell

Provides functions to work with the Ringo shell.

ringo/subprocess

A module for spawning processes, connecting to their input/output/errput and returning their response codes. It uses the current JVM's runtime provided by java.lang.Runtime.getRuntime(). The exact behavior of this module is highly system-dependent.

ringo/term

A module for printing ANSI terminal escape sequences. This module provides a number of useful color and style constants, and a replacement for the print function optimized for styled output.

ringo/worker

A Worker API based on the W3C Web Workers.

ringo/zip

This module provides classes to uncompress zip files and streams.

ringo/jsgi/connector

Low-level JSGI adapter implementation.

ringo/jsgi/response

This module provides response helper functions for composing JSGI response objects. For more flexibility the JsgiResponse is chainable.

ringo/utils/arrays

Provides utility functions for working with JavaScript Arrays.

ringo/utils/dates

Adds useful functions for working with JavaScript Date objects.

ringo/utils/files

A collection of file related utilities not covered by the fs module.

ringo/utils/http

Provides utility functions to work with HTTP requests and responses.

ringo/utils/numbers

Provides utility functions for working with JavaScript numbers.

ringo/utils/objects

Adds utility functions for working with JavaScript Objects

ringo/utils/strings

Adds useful methods to the JavaScript String type.

Index: Ringo Modules

Module assert

deepEqual (Object, Object)

equal (Object, Object)

fail (Object|String)

isFalse (Object)

isNaN (Object)

isNotNaN (Object)

isNotNull (Object)

isNotUndefined (Object)

isNull (Object)

isTrue (Object)

isUndefined (Object)

matches (String, RegExp)

notDeepEqual (Object, Object)

notEqual (Object, Object)

notStrictEqual (Object, Object)

ok (Object)

strictEqual (Object, Object)

stringContains (String, String)

throws (Object, Object)

Class ArgumentsError(String)

message

stackTrace

Class AssertionError(Object)

actual

expected

message

name

stackTrace

Module binary

parse (Array, Object)

Module ringo/base64

decode (String, String)
encode (String|Binary, String)

Module ringo/buffer

Class Buffer()

digest ()
forEach ()
reset ()
toString ()
write ()
writeln ()
length

Module ringo/concurrent

Class Semaphore()

signal ()
tryWait (,)
wait ()

Module ringo/daemon

destroy ()
init ()
start ()
stop ()

Module ringo/encoding

Class Decoder(, ,)

Module ringo/shell

printError (Exception, Array, Boolean)

printResult (,)

quit (Number)

read ()

readln (String, String)

start ()

write ()

writeln ()

Module ringo/subprocess

command (String, String, Object)

createProcess (Object)

status (String, String, Object)

system (String, String, Object)

Class Process()

connect (Stream, Stream, Stream)

kill ()

wait ()

stderr

stdin

stdout

Module ringo/term

write ()

writeln ()

BLACK

BLUE

BOLD

CYAN

GREEN

INVERSE

MAGENTA

ONBLACK

Module ringo/utils/files

Module assert

Assertion library for unit testing. It implements the CommonJS Unit Testing specification and adds some additional convenience methods.

Example

```
var assert = require('assert');
assert.deepEqual({b: 2, a: 1}, {a: 1, b: 2});
assert.isFalse(100 != 100);
assert.isNotNull(undefined);
```

See

The test module is a test runner for unit tests. It manages the execution of tests and provides the outcome to the user.

Functions

- deepEqual (actual, expected)
- equal (actual, expected)
- fail (options)
- isFalse (val)
- isNaN (val)
- isNotNaN (val)
- isNotNull (val)
- isNotUndefined (val)
- isNull (val)
- isTrue (val)
- isUndefined (val)
- matches (value, expr)
- notDeepEqual (actual, expected)
- notEqual (actual, expected)
- notStrictEqual (actual, expected)
- ok (value)
- strictEqual (actual, expected)
- stringContains (value, pattern)
- throws (func, expectedError)

Ringo Modules

- Modules
- Overview

assert
binary
console
fs
globals
io
net
system
test
ringo/args
ringo/base64
ringo/buffer
ringo/concurrent
ringo/daemon
ringo/encoding
ringo/engine
ringo/events
ringo/httpclient
ringo/httpserver
ringo/jsdoc
ringo/logging
ringo/markdown
ringo/mime
ringo/mustache
ringo/parser
ringo/profiler
ringo/promise
ringo/shell
ringo/subprocess
ringo/term
ringo/worker
ringo/zip
ringo/jsgi/connector
ringo/jsgi/response
ringo/utils/arrays
ringo/utils/dates
ringo/utils/files
ringo/utils/http
ringo/utils/numbers
ringo/utils/objects
ringo/utils/strings

Class ArgumentsError

Instance Properties

- message
- stackTrace

Class AssertionError

Instance Properties

- actual
- expected
- message
- name
- stackTrace

ArgumentsError (message)

Creates a new ArgumentsError instance

Parameters

String	**message**	The exception message

Returns

A newly created ArgumentsError instance

ArgumentsError.prototype.message

ArgumentsError.prototype.stackTrace

AssertionError (options)

Constructs a new AssertionError instance

Parameters

Object	**options**	An object containing error details

AssertionError.prototype.actual

AssertionError.prototype.expected

AssertionError.prototype.message

AssertionError.prototype.name

AssertionError.prototype.stackTrace

deepEqual (actual, expected)

Performs a deep recursive comparison of objects. It is equivalent to equal(). If an object's property holds a non-object type, it performs a non-strict comparison. Instances of Date are compared with getTime() according to universal time.

Example

```
// passing assertions
assert.deepEqual(5, "5");
assert.deepEqual(
  { time: new Date(2010, 5, 14) },
  { "time": new Date(2010, 5, 14) }
);
assert.deepEqual([1, 2, 3], ["1", "2", "3"]);
assert.deepEqual({"one": 1, "two": 2}, {"two": "2", "one": "1"});
```

Parameters

Object	**actual**	The actual value
Object	**expected**	The expected value

Throws

ArgumentsError, AssertionError

equal (actual, expected)

Performs a non-strict comparison with the simple comparison operator == to check if the values are equal. When they are equal, the assertion passes, otherwise it fails.

Example

```
// truthy conditionals
assert.equal(true, true);
assert.equal(true, "1");

// falsy conditionals
assert.equal(false, false);
assert.equal(false, "");
assert.equal(false, "0");
assert.equal(null, undefined);
```

Parameters

Object	**actual**	The actual value
Object	**expected**	The expected value

Throws

ArgumentsError, AssertionError

---

fail (options)

Basic failure method. Fails an assertion without checking any preconditions.

Example

```
// a complex condition
if (a === true && (b === "complex" || ...)) {
  assert.fail("This should not be reached!");
}
```

Parameters

Object|String	**options**	An object containing optional "message", "actual" and "expected" properties, or alternatively a message string

Throws

AssertionError

isFalse (val)

Checks if the value passed as argument is strict boolean false using ===.

Example

```
// passing assertion
assert.isFalse(100 != 100);

// failing assertion
assert.isFalse(100 == 100);
```

Parameters

Object	**val**	The value that should be boolean false.

Throws

ArgumentsError, AssertionError

---

isNaN (val)

Asserts that the value passed as argument is NaN. Uses global.isNaN() for the check.

Parameters

Object	**val**	The value that should be NaN.

Throws

ArgumentsError, AssertionError

---

isNotNaN (val)

Checks if the value passed as argument is not NaN. Uses global.isNaN() for the check.

Parameters

Object	**val**	The value that should be not NaN.

Throws

ArgumentsError, AssertionError

---

isNotNull (val)

Checks if the value passed as argument is strict not null using `===`.

Example

```
// passing assertions
assert.isNotNull(undefined);
assert.isNotNull("passes");

// failing assertion
assert.isNotNull(null);
```

Parameters

Object	**val**	The value that should be not null.

Throws

ArgumentsError, AssertionError

---

isNotUndefined (val)

Checks if the value passed as argument is not undefined using `===`.

Example

```
// passing assertions
assert.isNotUndefined(null);
assert.isNotUndefined("passes");

// failing assertion
assert.isNotUndefined(undefined);
```

Parameters

Object	**val**	The value that should be not undefined.

Throws

ArgumentsError, AssertionError

---

isNull (val)

Checks if the value passed as argument is strict null using `===`.

Example

```
// passing assertion
assert.isNull(null);

// failing assertions
assert.isNull(undefined);
assert.isNull("");
```

Parameters

Object	**val**	The value that should be null.

Throws

ArgumentsError, AssertionError

---

isTrue (val)

Checks if the value passed as argument is boolean true using `===`.

Example

```
// passing assertion
assert.isTrue(100 == 100);

// failing assertion
assert.isTrue(100 != 100);
```

Parameters

Object	**val**	The value that should be boolean true.

Throws

ArgumentsError, AssertionError

---

isUndefined (val)

Checks if the value passed as argument is strict undefined using `===`.

Example

```
// passing assertion
assert.isUndefined(undefined);

// failing assertions
assert.isUndefined(null);
assert.isUndefined("");
```

Parameters

Object	**val**	The value that should be undefined.

Throws

ArgumentsError, AssertionError

matches (value, expr)

Checks if the regular expression matches the string.

Example

```
assert.matches("this will pass", /p.?[s]{2}/);
assert.matches("this will fail", /[0-9]+/);
```

Parameters

String	**value**	The string that should contain the regular expression pattern
RegExp	**expr**	The regular expression that should match the value

Throws

ArgumentsError, AssertionError

notDeepEqual (actual, expected)

Performs a deep recursive comparison of objects. The comparison is equivalent to notEqual().

Example

```
// passing assertions
assert.notDeepEqual(
  { "time": new Date(2010, 5, 14) },
  { "time": new Date(2010, 5, 15) }
);
assert.notDeepEqual([1, 2, 3, 4], ["1", "2", "3"]);
assert.notDeepEqual({"one": 1, "two": 2}, {"three": "3", "one": "1"});
```

Parameters

Object	**actual**	The actual value
Object	**expected**	The expected value

Throws

Throws

ArgumentsError, AssertionError

---

notEqual (actual, expected)

Performs a non-strict comparison with the simple comparison operator != to check if the values are not equal. When they are not equal, the assertion passes, otherwise it fails.

Example

```
// passing assertions
assert.notEqual(true, false);
assert.notEqual(1, 2);
assert.notEqual(false, NaN);
assert.notEqual(null, NaN);
assert.notEqual(undefined, NaN);
```

Parameters

Object	**actual**	The actual value
Object	**expected**	The expected value

Throws

ArgumentsError, AssertionError

---

notStrictEqual (actual, expected)

Performs a strict comparison with the strict inequality operator !==. When the values are inequal in type and value, the assertion passes, otherwise it fails.

Example

```
// passing assertions
assert.notStrictEqual(null, undefined);
assert.notStrictEqual(1, "1");
assert.notStrictEqual(true, false);
```

Parameters

Object	**actual**	The actual value
Object	**expected**	The expected value

Throws

ArgumentsError, AssertionError

ok (value)

Checks if the value passed as argument is truthy.

Example

```
// passing assertions
assert.ok(true);
assert.ok("1");
assert.ok([]);
assert.ok({});
assert.ok(new Boolean(false));
assert.ok(Infinity);

// failing assertions
assert.ok(0);
assert.ok(false);
assert.ok(null);
assert.ok(undefined);
assert.ok("");
```

Parameters

Object	**value**	The value to check for truthiness

Throws

ArgumentsError, AssertionError

strictEqual (actual, expected)

Performs a strict comparison with the strict equality operator `===`. When the values are equal in type and value, the assertion passes, otherwise it fails.

Example

```
// passing assertions
assert.strictEqual(null, null);
assert.strictEqual(undefined, undefined);
assert.strictEqual(1, 1);
assert.strictEqual("1", "1");
assert.strictEqual(true, true);

// passing assertion
var obj = {};
assert.strictEqual(obj, obj);

// failing assertions
assert.strictEqual(null, undefined);
assert.strictEqual(true, "1");
assert.strictEqual(false, "");
assert.strictEqual(false, "0");
```

Parameters

Object	**actual**	The actual value
Object	**expected**	The expected value

Throws

ArgumentsError, AssertionError

---

stringContains (value, pattern)

Checks if the value passed as argument contains the pattern specified.

Example

```
assert.stringContains("this will pass", "pass");
assert.stringContains("this will fail", "pass");
```

Parameters

String	**value**	The string that should contain the pattern
String	**pattern**	The string that should be contained

Throws

ArgumentsError, AssertionError

---

throws (func, expectedError)

Checks if the function passed as argument throws a defined exception. It can also assert certain Java exceptions thrown by the function.

Example

```
var foo = function() { throw "foo"; };
var bar = function() { (new java.util.Vector()).get(0); }

// passes
assert.throws(foo, "foo");

// fails
assert.throws(foo, "bar");

// checks for a Java runtime exception, passes
```

```
assert.throws(bar, java.lang.ArrayIndexOutOfBoundsException);
```

Parameters

Object	**func**	The function to call
Object	**expectedError**	Optional object expected to be thrown when executing the function

Throws

ArgumentsError, AssertionError

Module binary

When dealing with network sockets or binary files, it’s necessary to read and write into byte streams. JavaScript itself does not provide a native representation of binary data, so this module provides two classes addressing this shortcoming. The implementation follows the CommonJS Binary/B proposal.

ByteArray implements a modifiable and resizable byte buffer.

ByteString implements an immutable byte sequence.

Both classes share a common base class Binary. The base class can't be instantiated. It exists only to affirm that ByteString and ByteArray instances of Binary.

When passed to a Java method that expects a byte[], instances of these classes are automatically unwrapped.

Example

```
// raw network streams only accept Binary as input
var stream = socket.getStream();
stream.write(new ByteArray([0xFA, 0xF0, 0x10, 0x58, 0xFF]));

// network protocols like HTTP/1.1 require ASCII
const CRLF = new ByteString("\r\n", "ASCII");
const EMPTY_LINE = new ByteString("\r\n\r\n", "ASCII");

// saves a java.security.Key to a file;
// the method getEncoded() returns a Java byte[]
fs.write("id_dsa.pub", ByteArray.wrap(publicKey.getEncoded()));
```

See

http://wiki.commonjs.org/wiki/Binary/B

Class Binary

Class ByteArray

Instance Methods

- byteAt (offset)
- charAt (offset)
- charCodeAt (offset)
- concat (arg...)
- copy (start, end, target, targetOffset)

Ringo Modules

- Modules
- Overview

assert
binary
console
fs
globals
io
net
system
test
ringo/args
ringo/base64
ringo/buffer
ringo/concurrent
ringo/daemon
ringo/encoding
ringo/engine
ringo/events
ringo/httpclient
ringo/httpserver
ringo/jsdoc
ringo/logging
ringo/markdown
ringo/mime
ringo/mustache
ringo/parser
ringo/profiler
ringo/promise
ringo/shell
ringo/subprocess
ringo/term
ringo/worker
ringo/zip
ringo/jsgi/connector
ringo/jsgi/response
ringo/utils/arrays
ringo/utils/dates
ringo/utils/files
ringo/utils/http
ringo/utils/numbers
ringo/utils/objects
ringo/utils/strings

decodeToString (encoding)
every (callback, thisObj)
filter (callback, thisObj)
forEach (fn, thisObj)
get (offset)
indexOf (sequence, start, stop)
lastIndexOf (sequence, start, stop)
map (callback, thisObj)
pop ()
push (num...)
reduce (callback, initialValue)
reduceRight (callback, initialValue)
reverse ()
set (offset, value)
shift ()
slice (begin, end)
some (callback, thisObj)
sort (comparator)
splice (index, howMany, elements...)
split (delimiter, options)
toArray ()
toByteArray ()
toByteString ()
toString ()
unshift (num...)
unwrap ()

Instance Properties

length

Static Methods

wrap (bytes)

Class ByteString

Instance Methods

byteAt (offset)
charAt (offset)
charCodeAt (offset)
concat (arg...)
copy (start, end, target, targetStart)
decodeToString (charset)

get (offset)
indexOf (sequence, start, stop)
lastIndexOf (sequence, start, stop)
slice (begin, end)
split (delimiter, options)
toArray ()
toByteArray ()
toByteString ()
toString ()
unwrap ()

Instance Properties

length

Static Methods

wrap (bytes)

Class String

Instance Methods

toByteArray (charset)
toByteString (charset)

Binary ()

Abstract base class for ByteArray and ByteString. The Binary type exists only to affirm that ByteString and ByteArray instances of Binary.

ByteArray (contentOrLength, [charset])

Constructs a writable and growable byte array.

If the first argument to this constructor is a number, it specifies the initial length of the ByteArray in bytes.

Else, the argument defines the content of the ByteArray. If the argument is a String, the constructor requires a second argument containing the name of the String's encoding. If called without arguments, an empty ByteArray is returned.

Parameters

Binary|Array|String|Number	**contentOrLength**	content or length of the

String	**[charset]**	ByteArray. the encoding name if the first argument is a String.

ByteArray.prototype.byteAt (offset)

Returns the byte at the given offset as ByteArray.

Parameters

Number	**offset**

Returns

ByteArray

ByteArray.prototype.charAt (offset)

Returns the byte at the given offset as ByteArray.

Parameters

Number	**offset**

Returns

ByteArray

ByteArray.prototype.charCodeAt (offset)

Returns charcode at the given offset.

Parameters

Number	**offset**

Returns

Number

ByteArray.prototype.concat (arg...)

Returns a ByteArray composed of itself concatenated with the given ByteString, ByteArray, and Array values.

Parameters

Binary|Array	**arg...**	one or more elements to

concatenate

Returns

ByteArray	a new ByteArray

ByteArray.prototype.copy (start, end, target, targetOffset)

Copy a range of bytes between start and stop from this object to another ByteArray at the given target offset.

Parameters

Number	**start**
Number	**end**
ByteArray	**target**
Number	**targetOffset**

ByteArray.prototype.decodeToString (encoding)

Returns the ByteArray decoded to a String using the given encoding

Parameters

String	**encoding**	the name of the encoding to use

ByteArray.prototype.every (callback, thisObj)

Tests whether all elements in the array pass the test implemented by the provided function.

Parameters

Function	**callback**	the callback function
Object	**thisObj**	optional this-object for callback

Returns

Boolean	true if every invocation of callback returns true

ByteArray.prototype.filter (callback, thisObj)

Return a ByteArray containing the elements of this ByteArray for which the callback function returns true.

Parameters

Function	**callback**	the filter function
Object	**thisObj**	optional this-object for callback

Returns

ByteArray	a new ByteArray

ByteArray.prototype.forEach (fn, thisObj)

Apply a function for each element in the ByteArray.

Parameters

Function	**fn**	the function to call for each element
Object	**thisObj**	optional this-object for callback

ByteArray.prototype.get (offset)

Returns the byte at the given offset as integer. get(offset) is analogous to indexing with brackets [offset].

Example

```
var ba = new ByteArray([0,255]);
print(ba[0]); // prints 0
```

Parameters

Number	**offset**

Returns

Number

ByteArray.prototype.indexOf (sequence, start, stop)

Returns the index of the first occurrence of sequence (a Number or a ByteString or ByteArray of any length) or -1 if none was found. If start and/or stop are specified, only elements between the indexes start and stop are searched.

Parameters

Number|Binary	**sequence**	the number or binary to look for
Number	**start**	optional index position at which to start searching
Number	**stop**	optional index position at which to stop searching

Returns

Number	the index of the first occurrence of sequence, or -1

ByteArray.prototype.lastIndexOf (sequence, start, stop)

Returns the index of the last occurrence of sequence (a Number or a ByteString or ByteArray of any length) or -1 if none was found. If start and/or stop are specified, only elements between the indexes start and stop are searched.

Parameters

Number|Binary	**sequence**	the number or binary to look for
Number	**start**	optional index position at which to start searching
Number	**stop**	optional index position at which to stop searching

Returns

Number	the index of the last occurrence of sequence, or -1

ByteArray.prototype.length

The length in bytes. This property is writable. Setting it to a value higher than the current value fills the new slots with 0, setting it to a lower value truncates the byte array.

ByteArray.prototype.map (callback, thisObj)

Returns a new ByteArray whose content is the result of calling the provided function with every element of the original ByteArray

Parameters

Function	**callback**	the callback
Object	**thisObj**	optional this-object for callback

Returns

ByteArray	a new ByteArray

ByteArray.prototype.pop ()

Removes the last element from an array and returns that element.

Returns

Number

ByteArray.prototype.push (num...)

Appends the given elements and returns the new length of the array.

Parameters

Number	**num...**	one or more numbers to append

Returns

Number	the new length of the ByteArray

ByteArray.prototype.reduce (callback, initialValue)

Apply a function to each element in this ByteArray as to reduce its content to a single value.

Parameters

Function	**callback**	the function to call with each element of the ByteArray
Object	**initialValue**	optional argument to be used as the first argument to the first call to the callback

Returns

the return value of the last callback invocation

See

https://developer.mozilla.org/en/Core_JavaScript_1.5_Reference/Global_Objects/Array/reduce

---

ByteArray.prototype.reduceRight (callback, initialValue)

Apply a function to each element in this ByteArray starting at the last element as to reduce its content to a single value.

Parameters

Function	**callback**	the function to call with each element of the ByteArray
Object	**initialValue**	optional argument to be used as the first argument to the first call to the callback

Returns

the return value of the last callback invocation

See

ByteArray.prototype.reduce
https://developer.mozilla.org/en/Core_JavaScript_1.5_Reference/Global_Objects/Array/reduceRight

---

ByteArray.prototype.reverse ()

Reverses the content of the ByteArray in-place

Returns

ByteArray	this ByteArray with its elements reversed

ByteArray.prototype.set (offset, value)

Sets the byte at the given offset. `set(offset, value)` is analogous to indexing with brackets `[offset]=value`.

Example

```
var ba = new ByteArray([0,255]);
ba[0] = 64;
print(ba[0]); // prints 64
```

Parameters

Number	**offset**
Number	**value**

ByteArray.prototype.shift ()

Removes the first element from the ByteArray and returns that element. This method changes the length of the ByteArray

Returns

Number	the removed first element

ByteArray.prototype.slice (begin, end)

Returns a new ByteArray containing a portion of this ByteArray.

Parameters

Number	**begin**	Zero-based index at which to begin extraction. As a negative index, begin indicates an offset from the end of the sequence.
Number	**end**	Zero-based index at which to end extraction. slice extracts up to but not including end. As a negative index, end indicates an offset from the end of the sequence. If end is omitted, slice extracts to the end of the sequence.

Returns

ByteArray	a new ByteArray

ByteArray.prototype.some (callback, thisObj)

Tests whether some element in the array passes the test implemented by the provided function.

Parameters

Function	**callback**	the callback function
Object	**thisObj**	optional this-object for callback

Returns

Boolean	true if at least one invocation of callback returns true

---

ByteArray.prototype.sort (comparator)

Sorts the content of the ByteArray in-place.

Parameters

Function	**comparator**	the function to compare entries

Returns

ByteArray	this ByteArray with its elements sorted

ByteArray.prototype.splice (index, howMany, elements...)

Changes the content of the ByteArray, adding new elements while removing old elements.

Parameters

Number	**index**	the index at which to start changing the ByteArray
Number	**howMany**	The number of elements to remove at the given position
Number	**elements...**	the new elements to add at the given position

---

ByteArray.prototype.split (delimiter, options)

Split at delimiter, which can by a Number, a ByteString, a ByteArray or an Array of the prior (containing multiple delimiters, i.e., "split at any of these delimiters"). Delimiters can have arbitrary size.

Parameters

Number|Binary	**delimiter**	one or more delimiter items
Object	**options**	optional object parameter with the following optional properties: • count - Maximum number of elements (ignoring delimiters) to return. The last returned element may contain delimiters. • includeDelimiter - Whether the delimiter should be included in the result.

ByteArray.prototype.toArray ()

Returns an array containing the bytes as numbers.

ByteArray.prototype.toByteArray ()

ByteArray.prototype.toByteString ()

ByteArray.prototype.toString ()

Returns a String representation of the ByteArray.

ByteArray.prototype.unshift (num...)

Adds one or more elements to the beginning of the ByteArray and returns its new length.

Parameters

Number	**num...**	one or more numbers to append

Returns

Number	the new length of the ByteArray

ByteArray.prototype.unwrap ()

Unwraps the underlying Java byte[] from ByteArray. It can be passed to a Java method that expects a byte array.

Returns

byte[]	a native Java byte array

ByteArray.wrap (bytes)

Create a ByteArray wrapper for a Java byte array without creating a new copy as the ByteArray constructor does. Any changes made on the ByteArray instance will be applied to the original byte array.

Parameters

Binary	**bytes**	a Java byte array or Binary instance

Returns

ByteArray	a ByteArray wrapping the argument

ByteString (content, charset)

Constructs an immutable byte string.

If the first argument is a String, the constructor requires a second argument containing the name of the String's encoding. If called without arguments, an empty ByteString is returned.

Parameters

Binary|Array|String	**content**	the content of the ByteString.
String	**charset**	the encoding name if the first argument is a String.

ByteString.prototype.byteAt (offset)

Returns the byte at the given offset as ByteString.

Parameters

Number	**offset**

Returns

ByteString

ByteString.prototype.charAt (offset)

Returns the byte at the given offset as ByteString.

Parameters

Number	**offset**

Returns

ByteString

ByteString.prototype.charCodeAt (offset)

Returns charcode at the given offset.

Parameters

Number	**offset**

Returns

Number

ByteString.prototype.concat (arg...)

Returns a ByteString composed of itself concatenated with the given ByteString, ByteArray, and Array values.

Parameters

Binary|Array	**arg...**	one or more elements to concatenate

Returns

ByteString	a new ByteString

ByteString.prototype.copy (start, end, target, targetStart)

Copy a range of bytes between start and stop from this ByteString to a target ByteArray at the given targetStart offset.

Parameters

Number	**start**
Number	**end**
ByteArray	**target**
Number	**targetStart**

ByteString.prototype.decodeToString (charset)

Returns this ByteString as string, decoded using the given charset.

Parameters

String	**charset**	the name of the string encoding

ByteString.prototype.get (offset)

Returns the byte at the given offset as a ByteString. get(offset) is analogous to indexing with brackets [offset].

Parameters

Number	**offset**

Returns

ByteString

ByteString.prototype.indexOf (sequence, start, stop)

Returns the index of the first occurrence of sequence (a Number or a ByteString or ByteArray of any length), or -1 if none was found. If start and/or stop are specified, only elements between the indexes start and stop are searched.

Parameters

Number|Binary	**sequence**	the number or binary to look for
Number	**start**	optional index position at which to start searching
Number	**stop**	optional index position at which to stop searching

Returns

Number	the index of the first occurrence of sequence, or -1

ByteString.prototype.lastIndexOf (sequence, start, stop)

Returns the index of the last occurrence of sequence (a Number or a ByteString or ByteArray of any length) or -1 if none was found. If start and/or stop are specified, only elements between the indexes start and stop are searched.

Parameters

Number|Binary	**sequence**	the number or binary to look for

Number	**start**	optional index position at which to start searching
Number	**stop**	optional index position at which to stop searching

Returns

Number	the index of the last occurrence of sequence, or -1

ByteString.prototype.length

The length in bytes. This property is read-only. Setting it to a value silently fails.

ByteString.prototype.slice (begin, end)

Returns a new ByteString containing a portion of this ByteString.

Parameters

Number	**begin**	Zero-based index at which to begin extraction. As a negative index, begin indicates an offset from the end of the sequence.
Number	**end**	Zero-based index at which to end extraction. slice extracts up to but not including end. As a negative index, end indicates an offset from the end of the sequence. If end is omitted, slice extracts to the end of the sequence.

Returns

ByteString	a new ByteString

ByteString.prototype.split (delimiter, options)

Split at delimiter, which can by a Number, a ByteString, a ByteArray or an Array of the prior (containing multiple delimiters, i.e., "split at any of these delimiters"). Delimiters can have arbitrary size.

Parameters

Number|Binary	**delimiter**	one or more delimiter items
Object	**options**	optional object parameter with the following optional properties: • count - Maximum number of elements (ignoring delimiters) to return. The last returned element may contain delimiters. • includeDelimiter - Whether the

delimiter should be included in the result.

ByteString.prototype.toArray ()

Returns an array containing the bytes as numbers.

ByteString.prototype.toByteArray ()

Returns a byte for byte copy of this immutable ByteString as a mutable ByteArray.

Returns

ByteArray	

ByteString.prototype.toByteString ()

Returns this ByteString itself.

ByteString.prototype.toString ()

Returns a debug representation such as "[ByteSTring 10]" where 10 is the length of this ByteString.

ByteString.prototype.unwrap ()

Unwraps the underlying Java byte[] from ByteString. It can be passed to a Java method that expects a byte array.

Returns

byte[]	a native Java byte array

ByteString.wrap (bytes)

Create a ByteString wrapper for a Java byte array without creating a new copy as the ByteString constructor does.

Parameters

Binary	**bytes**	a Java byte array or Binary instance

Returns

ByteString	a ByteString wrapping the argument

String

Not exported as constructor by this module.

String.prototype.toByteArray (charset)

Converts the String to a mutable ByteArray using the specified encoding.

Parameters

String	**charset**	the name of the string encoding. Defaults to 'UTF-8'

Returns

a ByteArray representing the string

String.prototype.toByteString (charset)

Converts the String to an immutable ByteString using the specified encoding.

Parameters

String	**charset**	the name of the string encoding. Defaults to 'UTF-8'

Returns

a ByteArray representing the string

Module console

This module provides functions to write on the standard error stream stderr for error logging and quick debugging. It's similar to the console object implemented in most web browsers.

Functions

- assert (expression, msg...)
- dir (obj)
- error (msg...)
- info (msg...)
- log (msg...)
- time (name)
- timeEnd (name)
- trace (msg...)
- warn (msg...)

assert (expression, msg...)

Tests that an expression is true and throws an AssertionError exception if not. It uses the ECMAScript toBoolean() convertion.

Example

```
>> var x = 10;
>> console.assert(x > 0, 'failed!'); // passes
>> console.assert(x < 0, 'failed!'); // fails
  AssertionError: failed! at <stdin>:12

>> console.assert(false, 'failed!'); // fails
  AssertionError: failed! at <stdin>:13

>> // passes; any Object expression is true
>> console.assert(new Boolean(false), 'failed!');
```

Parameters

expression	the expression to test

...	**msg...**	one or more error messages

---

dir (obj)

Prints a list of all properties of an object.

Example

```
>> var obj = { foo: "bar", baz: 12345 };
>> console.dir(obj);
{ foo: 'bar', baz: 12345 }
>> console.dir(global);
{ setTimeout: [Function], setInterval: [Function] }
```

Parameters

Object	**obj**	the object whose properties should be output

---

error (msg...)

Logs a message with the visual "error" representation, including the file name and line number of the calling code.

Example

```
>> console.error('Hello World!');
[error] Hello World! (<stdin>:1)
>> console.error('A: %s, B: %s, C: %s', 'a', 'b', 'c');
[error] A: a, B: b, C: c (<stdin>:3)
>> console.error('Current nanoseconds: %d', java.lang.System.nanoTime());
[error] Current nanoseconds: 9228448561643 (<stdin>:5)
```

Parameters

	msg...	one or more message arguments

---

info (msg...)

Logs a message with the visual "info" representation, including the file name and line number of the calling code.

Example

```
>> console.info('Hello World!');
```

```
[info] Hello World! (<stdin>:1)
>> console.info('A: %s, B: %s, C: %s', 'a', 'b', 'c');
[info] A: a, B: b, C: c (<stdin>:3)
>> console.info('Current nanoseconds: %d', java.lang.System.nanoTime());
[info] Current nanoseconds: 9677228481391 (<stdin>:5)
```

Parameters

...	**msg...**	one or more message arguments

---

log (msg...)

Logs a message to the console.

The first argument to log may be a string containing printf-like placeholders. Otherwise, multipel arguments will be concatenated separated by spaces.

Example

```
>> console.log('Hello World!');
Hello World!
>> console.log('A: %s, B: %s, C: %s', 'a', 'b', 'c');
A: a, B: b, C: c
>> console.log('Current nanoseconds: %d', java.lang.System.nanoTime());
Current nanoseconds: 9607196939209
```

Parameters

msg...	one or more message arguments

---

time (name)

Creates a new timer under the given name. Call console.timeEnd(name) with the same name to stop the timer and log the time elapsed.

Example

```
>> console.time('timer-1');
>> // Wait some time ...
>> console.timeEnd('timer-1');
timer-1: 15769ms
```

Parameters

String	**name**	the timer name

timeEnd (name)

Stops a timer created by a call to console.time(name) and logs the time elapsed.

Example

```
>> console.time('timer-1');
>> // Wait some time ...
>> console.timeEnd('timer-1');
timer-1: 15769ms
```

Parameters

String	**name**	the timer name

trace (msg...)

Prints a stack trace of JavaScript execution at the point where it is called.

Parameters

...	**msg...**	optional message arguments

warn (msg...)

Logs a message with the visual "warn" representation, including the file name and line number of the calling code.

Example

```
>> console.warn('Hello World!');
[warn] Hello World! (<stdin>:1)
>> console.warn('A: %s, B: %s, C: %s', 'a', 'b', 'c');
[warn] A: a, B: b, C: c (<stdin>:3)
>> console.warn('Current nanoseconds: %d', java.lang.System.nanoTime());
[warn] Current nanoseconds: 9294672097821 (<stdin>:5)
```

Parameters

	msg...	one or more message arguments

Module fs

This module provides a file system API for the manipulation of paths, directories, files, links, and the construction of input and output streams. It follows the CommonJS Filesystem/A proposal.

Some file system manipulations use a wrapper around standard POSIX functions. Their functionality depends on the concrete file system and operating system. Others use the java.io package and work cross-platform.

Functions

- absolute (path)
- base (path, ext)
- canonical (path)
- changeGroup (path, group)
- changeOwner (path, owner)
- changePermissions (path, permissions)
- changeWorkingDirectory (path)
- copy (from, to)
- copyTree (from, to)
- directory (path)
- exists (path)
- extension (path)
- group (path)
- hardLink (source, target)
- isAbsolute (path)
- isDirectory (path)
- isFile (path)
- isLink (path)
- isReadable (path)
- isRelative (path)
- isWritable (path)
- iterate (path)
- join ()

lastModified (path)

list (path)

listDirectoryTree (path)

listTree (path)

makeDirectory (path, permissions)

makeTree (path)

move (source, target)

normal (path)

open (path, options)

openRaw (path, options)

owner (path)

path ()

permissions (path)

read (path, options)

readLink (path)

relative (source, target)

remove (path)

removeDirectory (path)

removeTree (path)

resolve (paths...)

same (pathA, pathB)

sameFilesystem (pathA, pathB)

size (path)

split (path)

symbolicLink (source, target)

touch (path, mtime)

workingDirectory ()

write (path, content, options)

Class Path

Instance Methods

from (target)

join ()

listPaths ()

resolve ()

to (target)

toString ()
valueOf ()

Class Permissions

Instance Methods

toNumber ()
update (permissions)

Path ()

Path constructor. Path is a chainable shorthand for working with paths.

Path.prototype.from (target)

Return the relative path from the given source path to this path. Equivalent to fs.Path(fs.relative(source, this)).

Parameters

target

Path.prototype.join ()

Join a list of paths to this path.

Path.prototype.listPaths ()

Return the names of all files in this path, in lexically sorted order and wrapped in Path objects.

Path.prototype.resolve ()

Resolve against this path.

Path.prototype.to (target)

Return the relative path from this path to the given target path. Equivalent to fs.Path(fs.relative(this, target)).

Parameters

target

Path.prototype.toString ()

Path.prototype.valueOf ()

This is a non-standard extension, not part of CommonJS Filesystem/A.

Permissions (permissions, constructor)

The Permissions class describes the permissions associated with a file.

Parameters

Number|Object	**permissions**	a number or object representing the permissions.
	constructor	

Permissions.prototype.toNumber ()

Permissions.prototype.update (permissions)

Parameters

Number|Object	**permissions**

---

absolute (path)

Make the given path absolute by resolving it against the current working directory.

Example

```
>> fs.absolute('foo/bar/test.txt');
'/Users/username/Desktop/working-directory/foo/bar/test.txt'
```

Parameters

path	the path to resolve

Returns

String	the absolute path

---

base (path, ext)

Return the basename of the given path. That is the path with any leading directory components removed. If specified, also remove a trailing extension.

Example

```
>> fs.base('/a/b/c/foosomeext', 'someext');
'foo'
```

Parameters

String	**path**	the full path
String	**ext**	an optional extension to remove

Returns

String	the basename

---

canonical (path)

Returns the canonical path to a given abstract path. Canonical paths are both absolute and intrinsic, such that all paths that refer to a given file (whether it exists or not) have the same corresponding canonical path.

Parameters

String	**path**	a file path

Returns

String	the canonical path

---

changeGroup (path, group)

Changes the group of the specified file. This function wraps the POSIX chown() function. Supports group name string as well as gid number input.

Parameters

String	**path**	
String|Number	**group**	group name string or gid number

See

POSIX chown

---

changeOwner (path, owner)

Changes the owner of the specified file. This function wraps the POSIX chown() function. Supports user name string as well as uid number input.

Parameters

String	**path**	
String|Number	**owner**	the user name string or uid number

See

POSIX chown

---

changePermissions (path, permissions)

Changes the permissions of the specified file. This function wraps the POSIX chmod() function.

Parameters

String	**path**
Number|Object	**permissions**

See

POSIX chmod

---

changeWorkingDirectory (path)

Set the current working directory to path.

Parameters

String	**path**	the new working directory

---

copy (from, to)

Read data from one file and write it into another using binary mode.

Example

```
// Copies file from a temporary upload directory into /var/www
fs.copy('/tmp/uploads/fileA.txt', '/var/www/fileA.txt');
```

Parameters

String	**from**	original file
String	**to**	copy to create

---

copyTree (from, to)

Copy files from a source path to a target path. Files of the below the source path are copied to the corresponding locations relative to the target path, symbolic links to directories are copied but not traversed into.

Example

```
Before:
 └──── foo
     ├──── bar
     │   └──── example.m4a
     └──── baz

// Copy foo
fs.copyTree('./foo', './foo2');
```

```
After:
├─── foo
│    ├─── bar
│    │    └─── example.m4a
│    └─── baz
└─── foo2
     ├─── bar
     │    └─── example.m4a
     └─── baz
```

Parameters

String	**from**	the original tree
String	**to**	the destination for the copy

---

directory (path)

Return the dirname of the given path. That is the path with any trailing non-directory component removed.

Example

```
>> fs.directory('/Users/username/Desktop/example/test.txt');
'/Users/username/Desktop/example'
```

Parameters

String	**path**

Returns

String	the parent directory path

---

exists (path)

Return true if the file denoted by path exists, false otherwise.

Parameters

String	**path**	the file path.

---

extension (path)

Return the extension of a given path. That is everything after the last dot in the basename of the given path, including the last dot. Returns an empty string if no valid extension exists.

Example

```
>> fs.extension('test.txt');
'.txt'
```

Parameters

String	**path**

Returns

String	the file's extension

---

group (path)

Parameters

String	**path**

---

hardLink (source, target)

Creates a hard link at the target path that refers to the source path. The concrete implementation depends on the file system and the operating system.

This function wraps the POSIX link() function, which may not work on Microsoft Windows platforms.

Parameters

String	**source**	the source file
String	**target**	the target file

See

POSIX link

---

isAbsolute (path)

Check whether the given pathname is absolute. This is a non-standard extension, not part of CommonJS Filesystem/A.

Example

```
>> fs.isAbsolute('../../');
false
>> fs.isAbsolute('/Users/username/Desktop/example.txt');
true
```

Parameters

path	the path to check

Returns

Boolean	true if path is absolute, false if not

---

isDirectory (path)

Returns true if the file specified by path exists and is a directory.

Parameters

String	**path**	the file path

Returns

Boolean	whether the file exists and is a directory

---

isFile (path)

Returns true if the file specified by path exists and is a regular file.

Parameters

String	**path**	the file path

Returns

Boolean	whether the file exists and is a file

---

isLink (path)

Return true if target file is a symbolic link, false otherwise.

This function wraps the POSIX lstat() function to get the symbolic link status.

Parameters

String	**path**	the file path

Returns

Boolean	true if the given file exists and is a symbolic link

See

POSIX lstat

---

isReadable (path)

Returns true if the file specified by path exists and can be opened for reading.

Parameters

String	**path**	the file path

Returns

Boolean	whether the file exists and is readable

---

isRelative (path)

Check whether the given pathname is relative (i.e. not absolute). This is a non-standard extension, not part of CommonJS Filesystem/A.

Parameters

	path	the path to check

Returns

Boolean	true if path is relative, false if not

---

isWritable (path)

Returns true if the file specified by path exists and can be opened for writing.

Parameters

String	**path**	the file path

Returns

Boolean	whether the file exists and is writable

iterate (path)

Returns a generator that produces the file names of a directory.

Parameters

String	**path**	a directory path

join ()

Join a list of paths using the local file system's path separator. The result is not normalized, so join("..", "foo") returns "../foo".

See

http://wiki.commonjs.org/wiki/Filesystem/Join

lastModified (path)

Returns the time a file was last modified as a Date object.

Parameters

String	**path**	the file path

Returns

Date	the date the file was last modified

list (path)

Returns an array with all the names of files contained in the direcory path.

Parameters

String	**path**	the directory path

Returns

Array	a list of file names

listDirectoryTree (path)

Return an array with all directories below (and including) the given path, as discovered by depth-first traversal. Entries are in lexically sorted order within directories. Symbolic links to directories are not traversed into.

Example

```
// File system tree of the current working directory:
.
└── foo
    └── bar
        └── baz

fs.listDirectoryTree('.');
// returned array:
[ '', 'foo', 'foo/bar', 'foo/bar/baz' ]
```

Parameters

path	the path to discover

Returns

Array	array of strings with all directories lexically sorted

---

listTree (path)

Return an array with all paths (files, directories, etc.) below (and including) the given path, as discovered by depth-first traversal. Entries are in lexically sorted order within directories. Symbolic links to directories are returned but not traversed into.

Example

```
// File system tree of the current working directory:
.
├── foo
│   └── bar
│       └── baz
├── musicfile.m4a
└── test.txt

fs.listTree('.');
// returned array:
['', 'foo', 'foo/bar', 'foo/bar/baz', 'musicfile.m4a', 'test.txt']
```

Parameters

path	the path to list

Returns

Array	array of strings with all discovered paths

---

makeDirectory (path, permissions)

Create a single directory specified by path. If the directory cannot be created for any reason an error is thrown. This includes if the parent directories of path are not present. If a permissions argument is passed to this function it is used to create a Permissions instance which is applied to the given path during directory creation.

This function wraps the POSIX mkdir() function.

Parameters

String	**path**	the file path
Number|Object	**permissions**	optional permissions

See

POSIX mkdir

---

makeTree (path)

Create the directory specified by path including any missing parent directories.

Example

```
Before:
 └── foo

fs.makeTree('foo/bar/baz/');

After:
 └── foo
    └── bar
       └── baz
```

Parameters

path	the path of the tree to create

move (source, target)

Move a file from source to target.

Example

```
// Moves file from a temporary upload directory into /var/www
fs.move('/tmp/uploads/fileA.txt', '/var/www/fileA.txt');
```

Parameters

String	**source**	the source path
String	**target**	the target path

Throws

Error

normal (path)

Normalize a path by removing '.' and simplifying '..' components, wherever possible.

Example

```
>> fs.normal('../redundant/../foo/./bar.txt');
'../foo/bar.txt'
```

Parameters

path

Returns

String	the normalized path

open (path, options)

Open the file corresponding to path for reading or writing, depending on the options argument. Returns a binary stream or a text stream.

The options argument may contain the following properties:

- **read** *(boolean)* open the file in read-only mode.
- **write** *(boolean)* open the file in write mode starting at the beginning of the file.
- **append** *(boolean)* open the file in write mode starting at the end of the file.
- **binary** *(boolean)* open the file in binary mode.
- **charset** *(string)* open the file in text mode using the given encoding. Defaults to UTF-8.

Instead of an options object, a string with the following modes can be provided:

- **r** *(string)* equivalent to read-only
- **w** *(string)* equivalent to write
- **a** *(string)* equivalent to append
- **b** *(string)* equivalent to binary

So an options object { read: true, binary: true }and the mode string 'rb' are functionally equivalent. *Note: The options canonical and exclusive proposed by CommonJS are not supported.*

Example

```
// Opens a m4a file in binary mode
var m4aStream = fs.open('music.m4a', {
  binary: true,
  read: true
});

// The equivalent call with options as string
var m4aStream = fs.open('music.m4a', 'br');

// Opens a text file
var textStream = fs.open('example.txt', { read: true });

// The equivalent call with options as string
var textStream = fs.open('example.txt', 'r');
```

Parameters

String	**path**	the file path
Object|String	**options**	options as object properties or as mode string

Returns

Stream|TextStream	a Stream object in binary mode, otherwise a TextStream

openRaw (path, options)

Opens the file corresponding to path for reading or writing in binary mode. The options argument may contain the following properties:

- **read** *(boolean)* open the file in read-only mode. (default)
- **write** *(boolean)* open the file in write mode starting at the beginning of the file.
- **append** *(boolean)* open the file in write mode starting at the end of the file.

Parameters

String	**path**	the file path
Object	**options**	options

Returns

Stream

See

open

owner (path)

Parameters

String	**path**

path ()

A shorthand for creating a new Path without the new keyword.

permissions (path)

Parameters

String	**path**

read (path, options)

Read the content of the file corresponding to path. Returns a

String or ByteString object depending on the options argument. This function supports the same options as open().

Parameters

String	**path**	the file path
Object	**options**	optional options

Returns

String|Binary	the content of the file

---

readLink (path)

Returns the immediate target of the symbolic link at the given path.

This function wraps the POSIX readlink() function, which may not work on Microsoft Windows platforms.

Parameters

String	**path**	a file path

See

POSIX readlink

---

relative (source, target)

Establish the relative path that links source to target by strictly traversing up ('..') to find a common ancestor of both paths. If the target is omitted, returns the path to the source from the current working directory.

Example

```
>> fs.relative('foo/bar/', 'foo/baz/');
'../baz/'
>> fs.relative('foo/bar/', 'foo/bar/baz/');
'baz/'
```

Parameters

String	**source**
String	**target**

Returns

String the path needed to change from source to target

---

remove (path)

Remove a file at the given path. Throws an error if path is not a file or a symbolic link to a file.

Parameters

String	**path**	the path of the file to remove.

Throws

Error if path is not a file or could not be removed.

---

removeDirectory (path)

Remove a file or directory identified by path. Throws an error if path is a directory and not empty.

Parameters

String	**path**	the directory path

Throws

Error if the file or directory could not be removed.

---

removeTree (path)

Remove the element pointed to by the given path. If path points to a directory, all members of the directory are removed recursively.

Example

```
// File system tree of the current working directory:
├── foo
│   └── bar
│       └── baz
├── musicfile.m4a
└── test.txt

fs.removeTree('foo');
```

```
After:
├─── musicfile.m4a
└─── test.txt
```

Parameters

path	the element to delete recursively

---

resolve (paths...)

Join a list of paths by starting at an empty location and iteratively "walking" to each path given. Correctly takes into account both relative and absolute paths.

Example

```
>> fs.resolve('../.././foo/file.txt', 'bar/baz/', 'test.txt');
'../../foo/bar/baz/test.txt'
```

Parameters

paths...	the paths to resolve

Returns

String	the joined path

---

same (pathA, pathB)

Returns whether two paths refer to the same storage (file or directory), either by virtue of symbolic or hard links, such that modifying one would modify the other.

This function uses the POSIX stat() function to compare two files or links.

Parameters

String	**pathA**	the first path
String	**pathB**	the second path

Returns

Boolean	true if identical, otherwise false

See

POSIX stat

sameFilesystem (pathA, pathB)

Returns whether two paths refer to an entity of the same file system.

This function uses the POSIX stat() function to compare two paths by checking if the associated devices are identical.

Parameters

String	**pathA**	the first path
String	**pathB**	the second path

Returns

Boolean	true if same file system, otherwise false

See

POSIX stat

size (path)

Returns the size of a file in bytes, or throws an exception if the path does not correspond to an accessible path, or is not a regular file or a link.

Parameters

String	**path**	the file path

Returns

Number	the file size in bytes

Throws

Error if path is not a file

split (path)

Split a given path into an array of path components.

Example

```
>> fs.split('/Users/someuser/Desktop/subdir/test.txt');
```

```
[ '', 'Users', 'someuser', 'Desktop', 'subdir', 'test.txt' ]
```

Parameters

String	**path**

Returns

Array	the path components

---

symbolicLink (source, target)

Creates a symbolic link at the target path that refers to the source path. The concrete implementation depends on the file system and the operating system.

This function wraps the POSIX symlink() function, which may not work on Microsoft Windows platforms.

Parameters

String	**source**	the source file
String	**target**	the target link

See

POSIX symlink

---

touch (path, mtime)

Sets the modification time of a file or directory at a given path to a specified time, or the current time. Creates an empty file at the given path if no file or directory exists, using the default permissions.

Parameters

String	**path**	the file path
Date	**mtime**	optional date

---

workingDirectory ()

Return the path name of the current working directory.

Returns

String	the current working directory

---

write (path, content, options)

Open, write, flush, and close a file, writing the given content. If content is a ByteArray or ByteString from the binary module, binary mode is implied.

Parameters

String	**path**
ByteArray|ByteString|String	**content**
Object	**options**

See

ByteArray or ByteString for binary data

Module globals

RingoJS adopts some of the global properties from the Rhino shell and adds a few of its own.

Note that this module must and can not be imported like an ordinary module. It is evaluated only once upon RingoJS startup.

Functions

- addToClasspath (path)
- clearInterval (id)
- clearTimeout (id)
- defineClass (clazz)
- export (name...)
- gc ()
- getRepository (path)
- getResource (path)
- include (moduleId)
- load (filename...)
- module.resolve (path)
- module.singleton (id, factory)
- print (arg...)
- privileged (func)
- quit ()
- require (moduleId)
- seal (obj)
- setInterval (callback, delay, args)
- setTimeout (callback, delay, [args])
- spawn (func)
- sync (func, [obj])

Properties

- arguments
- console

addToClasspath (path)

Adds path to the RingoJS application classpath at runtime. This is necessary if libraries and their classes are not in the default Java classpath.

Calling addToClasspath() will invoke an org.ringojs.engine.AppClassLoader, which is a subclass of java.net.URLClassLoader. It checks if the URL has been already loaded and if not, adds it to the resource search path. If the given URL ends with /, it will be treated as directory, otherwise it's assumed to refer to a jar file. The function throws an exception if it could not load a path or if it fails.

Example

```
// Adds Apache Lucene text search engine to the classpath
addToClasspath("../jars/lucene-core.jar");
```

Parameters

String|Resource|Repository	**path**	a directory or jar path; or a single resource; or a repository

See

Ringo Java Integration

arguments

The arguments array contains the command line arguments RingoJS was started with.

Note that this variable is shadowed by the arguments object inside functions which is why it is usually safer to use system.args instead.

---

clearInterval (id)

Cancel a timeout previously scheduled with setInterval().

Parameters

object	**id**	the id object returned by setInterval()

See

setInterval

---

clearTimeout (id)

Cancel a timeout previously scheduled with setTimeout().

Parameters

object	**id**	the id object returned by setTimeout()

See

setTimeout

---

console

Debug console to print messages on stderr. It’s similar to the console object implemented in most web browsers.

Example

```
console.log('Hello World!');
```

See

console

defineClass (clazz)

Loads a custom Java-based host object into the global scope. This is useful to provide transparent access to Java classes inside the JavaScript environment. It uses ScriptableObject.defineClass() to define the extension.

Example

```
// org.somejavalib.Foo extends org.mozilla.javascript.ScriptableObject
defineClass(org.somejavalib.Foo);
var x = new Foo();
```

Parameters

java.lang.Class	**clazz**	the host object's Java class

See

ECMAScript 5 host object definition
Rhino's ScriptableObject.defineClass()
The Java package org.ringojs.wrappers includes typical host objects like Binary, EventAdapter and Stream.

environment

The environment object contains the Java system properties.

See

Java System Properties

export (name...)

Takes any number of top-level names to be exported from this module.

This is a non-standard alternative to the exports object for exporting values in a less verbose and intrusive way.

Example

```
// equivalent to exports.foo = function() { ... }
// and exports.bar = function() { ... }
```

```
export(
  "foo",
  "bar"
);

function foo() { ... };
function bar() { ... };
```

Parameters

name...	one or more names of exported properties

---

exports

The exports object as defined in the CommonJS Modules 1.1.1 specification.

Define properties on the exports object to make them available to other modules requiring this module.

Example

```
exports.multiply = function(x, y) { return x * y; }
```

---

gc ()

Runs the garbage collector.

See

java.lang.Runtime.gc()

---

getRepository (path)

Resolve path following the same logic require uses for module ids and return an instance of org.ringojs.repository.Repository representing the resolved path.

Parameters

String	**path**	the repository path

Returns

org.ringojs.repository.Repository	a repository

See

getResource

---

getResource (path)

Resolve path following the same logic require uses for module ids and return an instance of org.ringojs.repository.Resource representing the resolved path.

Parameters

String	**path**	the resource path

Returns

org.ringojs.repository.Resource	a resource

See

getRepository

---

global

A reference to the global object itself.

When a module is evaluated in RingoJS it uses its own private module scope which in turn uses this shared global object as prototype. Therefore, properties of the global object are visible in every module.

Since the global object is hidden in the prototype chain of module scopes it cannot normally be accessed directly. This reference allows you to do so, defining real global variables if you want to do so.

Example

```
global.foo = "bar";
```

---

include (moduleId)

Load a module and include all its properties in the calling scope.

Example

```
include('fs');
// calls fs.isReadable()
if (isReadable('essay.txt') ) { ... }
```

Parameters

String	**moduleId**	the id or path of the module to load

---

load (filename...)

Load JavaScript source files named by string arguments. If multiple arguments are given, each file is read in and executed in turn.

Parameters

String	**filename...**	one or more file names

---

module

The module object as defined in the CommonJS Modules 1.1.1 specification.

The RingoJS module object has the following properties:

- directory
- exports
- id
- path
- uri
- resolve
- singleton

---

module.directory

The directory that contains this module.

---

module.exports

By default, module.exports refers to exports object. Setting this property to a different value will cause that value to be used as exports object instead.

---

module.id

The module id of this module.

---

module.path

The absolute path of this module's source.

---

module.resolve (path)

Resolve path relative to this module, like when calling require with a moduleId starting with './' or '../'.

This returns an absolute path if the current module is a regular file. For other types of modules such as those residing in a .jar file it returs a relative path relative to the module's module path root.

Parameters

String	**path**

Returns

String	the resolved path

---

module.singleton (id, factory)

module.singleton enables the creation of singletons across all workers using the same module. This means that a value within a module will be instantiated at most once for all concurrent worker threads even though workers usually operate on their own private scopes and variables.

The id argument identifies the singleton within the module.

When module.singleton is called with an id that has not been initialized yet and the factory argument is defined, factory is invoked and its return value is henceforth used as singleton value for the given id.

Once the value of a singleton has been set, the factory function is never called again and all calls to module.singleton with that id return that original value.

module.singleton supports lazy initialization. A singleton can remain undefined if module.singleton is called without factory argument. In this case module.singleton returns undefined until it is first called with a factory argument.

Parameters

String	**id**	the singleton id
Function	**factory**	(optional) factory function for the singleton

Returns

the singleton value

---

module.uri

This module's URI.

---

print (arg...)

Converts each argument to a string and prints it.

Parameters

arg...	one ore more arguments

---

privileged (func)

Calls func with the privileges of the current code base instead of the privileges of the code in the call stack.

This is useful when running with Java security manager enabled using the -P or --policy command line switches.

Parameters

Function	**func**	a function

Returns

Object	the return value of the function

---

quit ()

Quit the RingoJS shell. The shell will also quit in interactive mode if an end-of-file character (CTRL-D) is typed at the prompt.

---

require (moduleId)

The require function as defined in the CommonJS Modules 1.1.1 specification.

moduleId is resolved following these rules:

- If moduleId starts with './' or '../' it is resolved relative to the current module.
- If moduleId is relative (starting with a file or directory name), it is resolve relative to the module search path.
- If path is absolute (e.g. starting with '/') it is interpreted as absolute file name.

The RingoJS require function has the following properties:

- extensions
- main
- paths

Parameters

String	**moduleId**	the id or path of the module to load

Returns

Object	the exports object of the required module

---

require.extensions

An object used to extend the way require loads modules.

Use a file extension as key and a function as value. The function should accept a Resource object as argument and return either a string to be used as JavaScript module source or an object which will be directly returned by require.

For example, the following one-liner will enable require() to load XML files as E4X modules:

```
require.extensions['.xml'] = function(r) new XML(r.content);
```

---

require.main

If RingoJS was started with a command line script, require.main contains the module object of the main module. Otherwise, this property is defined but has the value undefined.

Example

```
// is the current module is the main module?
 if (require.main == module.id) {
  // Start up actions like in a Java public static void main() method
  var server = new Server();
  server.start();
}
```

---

require.paths

An array that contains the module search path. You can add or remove paths items to or from this array in order to change the places where RingoJS will look for modules.

---

seal (obj)

Seal the specified object so any attempt to add, delete or modify its properties would throw an exception.

Parameters

Object	**obj**	a JavaScript object

---

setInterval (callback, delay, args)

Calls a function repeatedly, with a fixed time delay between each call to that function. The function will be called in the thread of the local event loop. This means it will only run after the currently executing code and other code running before it have terminated.

Parameters

function	**callback**	a function
number	**delay**	the delay in milliseconds
...	**args**	optional arguments to pass to the function

Returns

object	an id object useful for cancelling the scheduled invocation

See

clearInterval

---

setTimeout (callback, delay, [args])

Executes a function after specified delay. The function will be called in the thread of the local event loop. This means it will only run after the currently executing code and other code running before it have terminated.

Parameters

function	**callback**	a function
number	**delay**	the delay in milliseconds
...	**[args]**	optional arguments to pass to the function

Returns

object	an id object useful for cancelling the scheduled invocation

See

clearTimeout

spawn (func)

Calls func in a new thread from an internal thread pool and returns immediately.

Parameters

Function	**func**	a function

sync (func, [obj])

Returns a wrapper around a function that synchronizes on the original function or, if provided, on the second argument.

When multiple threads call functions that are synchronized on the same object, only one function call is allowed to execute at a time.

Example

```
exports.synchronizedFunction = sync(function() {
  // no two threads can execute this code in parallel
});
```

Parameters

Function	**func**	a function
Object	**[obj]**	optional object to synchronize on

Returns

Function	a synchronized wrapper around the function

Module io

This module provides functions for reading and writing streams of raw bytes. It implements the Stream and TextStream classes as per the CommonJS IO/A proposal.

Streams are closely related with two other modules. Low-level byte manipulation is provided by the binary module and uses the ByteArray or ByteString class. The fs module returns io streams for reading and writing files.

Class MemoryStream

Instance Methods

- close ()
- closed ()
- flush ()
- read (maxBytes)
- readInto (buffer, begin, end)
- readable ()
- seekable ()
- writable ()
- write (source, begin, end)

Instance Properties

- content
- length
- position

Class Stream

Instance Methods

- close ()
- closed ()
- copy (output)
- flush ()
- forEach (fn, [thisObj])
- read (maxBytes)

readInto (buffer, begin, end)

readable ()

seekable ()

skip (num)

unwrap ()

writable ()

write (source, begin, end)

Instance Properties

inputStream

outputStream

Class TextStream

Instance Methods

close ()

copy (output)

flush ()

forEach (callback, [thisObj])

iterator ()

next ()

print ()

read ()

readInto ()

readLine ()

readLines ()

readable ()

seekable ()

writable ()

write ()

writeLine (line)

writeLines (lines)

Instance Properties

content

raw

MemoryStream (binaryOrNumber)

A binary stream that reads from and/or writes to an in-memory byte array.

If the constructor is called with a Number argument, a ByteArray with the given length is allocated and the length of the stream is set to zero.

If the argument is a binary object it will be used as underlying buffer and the stream length set to the length of the binary object. If argument is a ByteArray, the resulting stream is both readable, writable, and seekable. If it is a ByteString, the resulting stream is readable and seekable but not writable.

If called without argument, a ByteArray of length 1024 is allocated as buffer.

Parameters

Binary|Number	**binaryOrNumber**	the buffer to use, or the initial capacity of the buffer to allocate.

MemoryStream.prototype.close ()

Closes the stream, freeing the resources it is holding.

MemoryStream.prototype.closed ()

Returns true if the stream is closed, false otherwise.

Returns

Boolean	true if the stream has been closed

MemoryStream.prototype.content

The wrapped buffer.

MemoryStream.prototype.flush ()

Flushes the bytes written to the stream to the underlying medium.

MemoryStream.prototype.length

The number of bytes in the stream's underlying buffer.

MemoryStream.prototype.position

The current position of this stream in the wrapped buffer.

MemoryStream.prototype.read (maxBytes)

Read up to maxBytes bytes from the stream, or until the end of the stream has been reached. If maxBytes is not specified, the full stream is read until its end is reached. Reading from a stream where the end has already been reached returns an empty ByteString.

Parameters

Number	**maxBytes**	the maximum number of bytes to read

Returns

ByteString

See

Stream.prototype.read

MemoryStream.prototype.readInto (buffer, begin, end)

Read bytes from this stream into the given buffer. This method does *not* increase the length of the buffer.

Parameters

ByteArray	**buffer**	the buffer
Number	**begin**	optional begin index, defaults to 0.
Number	**end**	optional end index, defaults to

buffer.length - 1.

Returns

Number	The number of bytes read or -1 if the end of the stream has been reached

See

Stream.prototype.readInto

MemoryStream.prototype.readable ()

Returns true if the stream supports reading, false otherwise. Always returns true for MemoryStreams.

Returns

Boolean	true if stream is readable

See

Stream.prototype.readable

MemoryStream.prototype.seekable ()

Returns true if the stream is randomly accessible and supports the length and position properties, false otherwise. Always returns true for MemoryStreams.

Returns

Boolean	true if stream is seekable

See

Stream.prototype.seekable

MemoryStream.prototype.writable ()

Returns true if the stream supports writing, false otherwise. For MemoryStreams this returns true if the wrapped binary is an instance of ByteArray.

Returns

Boolean	true if stream is writable

See

Stream.prototype.writable

MemoryStream.prototype.write (source, begin, end)

Write bytes from b to this stream. If begin and end are specified, only the range starting at begin and ending before end is written.

Parameters

Binary	**source**	The source to be written from
Number	**begin**	optional
Number	**end**	optional

See

Stream.prototype.write

Stream ()

This class implements an I/O stream used to read and write raw bytes.

Stream.prototype.close ()

Closes the stream, freeing the resources it is holding.

Stream.prototype.closed ()

Returns true if the stream has been closed, false otherwise.

Returns

Boolean	true if the stream has been closed

Stream.prototype.copy (output)

Reads all data available from this stream and writes the result to the given output stream, flushing afterwards. Note that this function does not close this stream or the output stream after copying.

Parameters

Stream	**output**	The target Stream to be written to.

Stream.prototype.flush ()

Flushes the bytes written to the stream to the underlying medium.

Stream.prototype.forEach (fn, [thisObj])

Read all data from this stream and invoke function fn for each chunk of data read. The callback function is called with a ByteArray as single argument. Note that the stream is not closed after reading.

Parameters

Function	**fn**	the callback function
Object	**[thisObj]**	optional this-object to use for callback

---

Stream.prototype.inputStream

The wrapped java.io.InputStream.

Stream.prototype.outputStream

The wrapped java.io.OutputStream.

Stream.prototype.read (maxBytes)

Read up to maxBytes bytes from the stream, or until the end of the stream has been reached. If maxBytes is not specified, the full stream is read until its end is reached. Reading from a stream where the end has already been reached returns an empty ByteString.

Parameters

Number	**maxBytes**	the maximum number of bytes to read

Returns

ByteString

Stream.prototype.readInto (buffer, begin, end)

Read bytes from this stream into the given buffer. This method does *not* increase the length of the buffer.

Parameters

ByteArray	**buffer**	the buffer
Number	**begin**	optional begin index, defaults to 0.
Number	**end**	optional end index, defaults to buffer.length - 1.

Returns

Number	The number of bytes read or -1 if the end of the stream has been reached

---

Stream.prototype.readable ()

Returns true if the stream supports reading, false otherwise.

Returns

Boolean	true if stream is readable

Stream.prototype.seekable ()

Returns true if the stream is randomly accessible and supports the length and position properties, false otherwise.

Returns

Boolean	true if stream is seekable

Stream.prototype.skip (num)

Try to skip over num bytes in the stream. Returns the number of acutal bytes skipped or throws an error if the operation could not be completed.

Parameters

Number	**num**	bytes to skip

Returns

Number	actual bytes skipped

Stream.prototype.unwrap ()

Get the Java input or output stream instance wrapped by this Stream.

Stream.prototype.writable ()

Returns true if the stream supports writing, false otherwise.

Returns

Boolean	true if stream is writable

Stream.prototype.write (source, begin, end)

Write bytes from b to this stream. If begin and end are specified, only the range starting at begin and ending before end is written.

Parameters

Binary	**source**	The source to be written from
Number	**begin**	optional
Number	**end**	optional

TextStream (io, options, buflen)

A TextStream implements an I/O stream used to read and write strings. It wraps a raw Stream and exposes a similar interface.

Parameters

Stream	**io**	The raw Stream to be wrapped.
Object	**options**	the options object. Supports the following properties: • charset: string containing the name of the encoding to use. Defaults to "utf8". • newline: string containing the newline character sequence to use in writeLine() and writeLines(). Defaults to "\n". • delimiter: string containing the delimiter to use in print(). Defaults to " ".
number	**buflen**	optional buffer size. Defaults to 8192.

TextStream.prototype.close ()

See

Stream.prototype.close

TextStream.prototype.content

If the wrapped stream is a MemoryStream this contains its content decoded to a String with this streams encoding. Otherwise contains an empty String.

TextStream.prototype.copy (output)

Reads from this stream with readLine, writing the results to the target stream and flushing, until the end of this stream is reached.

Parameters

output

Returns

TextStream	this stream

TextStream.prototype.flush ()

See

Stream.prototype.flush

TextStream.prototype.forEach (callback, [thisObj])

Calls callback with each line in the input stream.

Example

```
var txtStream = fs.open('./browserStats.csv', 'r');
txtStream.forEach(function(line) {
  console.log(line); // Print one single line
});
```

Parameters

Function	**callback**	the callback function
Object	**[thisObj]**	optional this-object to use for callback

TextStream.prototype.iterator ()

Returns this stream.

Returns

TextStream	this stream

TextStream.prototype.next ()

Returns the next line of input without the newline. Throws StopIteration if the end of the stream is reached.

Example

```
var fs = require('fs');
var txtStream = fs.open('./browserStats.csv', 'r');
try {
  while (true) {
     console.log(txtStream.next());
  }
} catch (e) {
  console.log("EOF");
}
```

Returns

String	the next line

TextStream.prototype.print ()

Writes all argument values as a single line, delimiting the values using a single blank.

Example

```
>> var fs = require('fs');
>> var txtOutStream = fs.open('./demo.txt', 'w');
>> txtOutStream.print('foo', 'bar', 'baz');

// demo.txt content:
foo bar baz
```

Returns

TextStream	this stream

TextStream.prototype.raw

The wrapped binary stream.

TextStream.prototype.read ()

Read the full stream until the end is reached and return the data read as string.

Returns

String

TextStream.prototype.readInto ()

Not implemented for TextStream. Calling this method will raise an error.

TextStream.prototype.readLine ()

Reads a line from this stream. If the end of the stream is reached before any data is gathered, returns an empty string. Otherwise, returns the line including only the newline character. Carriage return will be dropped.

Returns

String	the next line

TextStream.prototype.readLines ()

Returns an Array of Strings, accumulated by calling readLine until it returns an empty string. The returned array does not include the final empty string, but it does include a trailing newline at the end of every line.

Example

```
>> var fs = require('fs');
>> var txtStream = fs.open('./sampleData.csv', 'r');
>> var lines = txtStream.readLines();
>> console.log(lines.length + ' lines');
6628 lines
```

Returns

Array	an array of lines

TextStream.prototype.readable ()

See

Stream.prototype.readable

TextStream.prototype.seekable ()

Always returns false, as a TextStream is not randomly accessible.

TextStream.prototype.writable ()

See

Stream.prototype.writable

TextStream.prototype.write ()

Writes all arguments to the stream.

Example

```
>> var fs = require('fs');
>> var txtOutStream = fs.open('./demo.txt', 'w');
>> txtOutStream.write('foo', 'bar', 'baz');

// demo.txt content:
foobarbaz
```

Returns

TextStream	this stream

TextStream.prototype.writeLine (line)

Writes the given line to the stream, followed by a newline.

Parameters

line

Returns

TextStream	this stream

TextStream.prototype.writeLines (lines)

Writes the given lines to the stream, terminating each line with a newline.

This is a non-standard extension, not part of CommonJS IO/A.

Parameters

lines

Returns

TextStream	this stream

Module net

This module provides support for networking using TCP and UDP sockets. A socket represents a connection between a client and a server program over a network. The underlying native binding is provided by the java.net package.

Example

```
// A simple TCP server
var io = require('io');
var net = require('net');

var server = new net.ServerSocket();
server.bind('127.0.0.1', 6789);

var socket = server.accept();
var stream = new io.TextStream(socket.getStream(), {
  'charset': 'US-ASCII'
});

var line;
do {
  // Read one line from the client
  line = stream.readLine();
  console.log(line);

  // Write back to the client
  stream.writeLine("Received: " + line);
} while (line.indexOf("END") < 0);

stream.close();
socket.close();
server.close();
```

Class DatagramSocket

Instance Methods

- bind (host, port)
- close ()
- connect (host, port)
- disconnect ()
- getTimeout ()
- isBound ()
- isClosed ()

isConnected ()

localAddress ()

receive (length, buffer)

receiveFrom (length, buffer)

remoteAddress ()

send (data)

sendTo (host, port, data)

setTimeout (timeout)

Class ServerSocket

Instance Methods

accept ()

bind (host, port)

close ()

getTimeout ()

isBound ()

isClosed ()

localAddress ()

setTimeout (timeout)

Class Socket

Instance Methods

bind (host, port)

close ()

connect (host, port, [timeout])

getStream ()

getTimeout ()

isBound ()

isClosed ()

isConnected ()

localAddress ()

remoteAddress ()

setTimeout (timeout)

DatagramSocket ()

The DatagramSocket class is used to create a UDP socket.

DatagramSocket.prototype.bind (host, port)

Binds the socket to a local address and port. If address or port are omitted the system will choose a local address and port to bind the socket.

Parameters

String	**host**	address (interface) to which the socket will be bound.
Number	**port**	port number to bind the socket to.

DatagramSocket.prototype.close ()

Close the socket immediately

DatagramSocket.prototype.connect (host, port)

Connect the socket to a remote address. If a DatagramSocket is connected, it may only send data to and receive data from the given address. By default DatagramSockets are not connected.

Parameters

String	**host**	IP address or hostname
Number	**port**	port number or service name

DatagramSocket.prototype.disconnect ()

Disconnects the socket.

DatagramSocket.prototype.getTimeout ()

Return the current timeout of this DatagramSocket. A value of zero implies that timeout is disabled, i.e. receive() will never time out.

Returns

Number	the current timeout

DatagramSocket.prototype.isBound ()

Returns whether this socket is bound to an address.

Returns

Boolean	true if the socket has been bound to an address

DatagramSocket.prototype.isClosed ()

Returns whether the socket is closed or not.

Returns

Boolean	true if the socket has been closed

DatagramSocket.prototype.isConnected ()

Returns whether the socket is connected or not.

Returns

Boolean	true if the socket has been connected to a remote address

---

DatagramSocket.prototype.localAddress ()

Get the local address to which this socket is bound. This returns an object with a property `address` containing the IP address as string and a property `port` containing the port number, e.g. `{address: '127.0.0.1', port: 8080}`.

Returns

Object	an address descriptor

DatagramSocket.prototype.receive (length, buffer)

Receive a datagram packet from this socket. This method does not return the sender's IP address, so it is meant to be in conjunction with connect().

Parameters

Number	**length**	the maximum number of bytes to receive
ByteArray	**buffer**	optional buffer to store bytes in

Returns

ByteArray	the received data

DatagramSocket.prototype.receiveFrom (length, buffer)

Receive a datagram packet from this socket. This method returns an object with the following properties:

- address: the sender's IP address as string
- port: the sender's port number
- data: the received data

Parameters

Number	**length**	the maximum number of bytes to receive
ByteArray	**buffer**	optional buffer to store bytes in

Returns

Object	the received packet

DatagramSocket.prototype.remoteAddress ()

Get the remote address to which this socket is connected. This returns an object with a property address containing the IP address as string and a property port containing the port number, e.g. {address: '127.0.0.1', port: 8080}.

Returns

Object	an address descriptor

DatagramSocket.prototype.send (data)

Send a datagram packet from this socket. This method does not allow the specify the recipient's IP address, so it is meant to be in conjunction with connect().

Parameters

Binary	**data**	the data to send

DatagramSocket.prototype.sendTo (host, port, data)

Send a datagram packet from this socket to the specified address.

Parameters

String	**host**	the IP address of the recipient
Number	**port**	the port number
Binary	**data**	the data to send

DatagramSocket.prototype.setTimeout (timeout)

Enable/disable timeout with the specified timeout, in milliseconds. With this option set to a non-zero timeout, a call to receive() for this DatagramSocket will block for only this amount of time.

Parameters

Number	**timeout**	timeout in milliseconds

ServerSocket ()

This class implements a server socket. Server sockets wait for requests coming in over the network.

ServerSocket.prototype.accept ()

Listens for a connection to be made to this socket and returns a new Socket object. The method blocks until a connection is made.

Returns

Socket	a newly connected socket object

ServerSocket.prototype.bind (host, port)

Binds the socket to a local address and port. If address or port are omitted the system will choose a local address and port to bind the socket.

Parameters

String	**host**	address (interface) to which the socket will be bound.
Number	**port**	port number to bind the socket to.

ServerSocket.prototype.close ()

Close the socket immediately

ServerSocket.prototype.getTimeout ()

Return the current timeout of this ServerSocket. A value of zero implies that timeout is disabled, i.e. accept() will never time out.

Returns

Number	the current timeout

ServerSocket.prototype.isBound ()

Returns whether this socket is bound to an address.

Returns

Boolean	true if the socket has been bound to an address

ServerSocket.prototype.isClosed ()

Returns whether the socket is closed or not.

Returns

Boolean	true if the socket has been closed

ServerSocket.prototype.localAddress ()

Get the local address to which this socket is bound. This returns an object with a property address containing the IP address as string and a property port containing the port number, e.g. {address: '127.0.0.1', port: 8080}

Returns

Object	an address descriptor

ServerSocket.prototype.setTimeout (timeout)

Enable/disable timeout with the specified timeout, in milliseconds. With this option set to a non-zero timeout, a call to accept() for this ServerSocket will block for only this amount of time.

Parameters

Number	**timeout**	timeout in milliseconds

Socket ()

The Socket class is used to create a TCP socket. Newly created sockets must be connected to a remote address before being able to send and receive data.

Socket.prototype.bind (host, port)

Binds the socket to a local address and port. If address or port are omitted the system will choose a local address and port to bind the socket.

Parameters

String	**host**	address (interface) to which the socket will be bound.
Number	**port**	port number to bind the socket to.

Socket.prototype.close ()

Close the socket immediately

Socket.prototype.connect (host, port, [timeout])

Initiate a connection on a socket. Connect to a remote port on the specified host with a connection timeout. Throws an exception in case of failure.

Parameters

String	**host**	IP address or hostname
Number	**port**	port number or service name
Number	**[timeout]**	optional timeout value in milliseconds

Socket.prototype.getStream ()

Get the I/O stream for this socket.

Returns

Stream	a binary stream

See

io.Stream

Socket.prototype.getTimeout ()

Return the current timeout of this Socket. A value of zero implies that timeout is disabled, i.e. read() will never time out.

Returns

Number	the current timeout

Socket.prototype.isBound ()

Returns whether this socket is bound to an address.

Returns

true if the socket has been bound to an address

Socket.prototype.isClosed ()

Returns whether the socket is closed or not.

Returns

true if the socket has been closed

Socket.prototype.isConnected ()

Returns whether the socket is connected or not.

Returns

true if the socket has been connected to a remote address

Socket.prototype.localAddress ()

Get the local address to which this socket is bound. This returns an object with a property `address` containing the IP address as string and a property `port` containing the port number, e.g. `{address: '127.0.0.1', port: 8080}`.

Returns

Object	an address descriptor

Socket.prototype.remoteAddress ()

Get the remote address to which this socket is connected. This returns an object with a property address containing the IP address as string and a property port containing the port number, e.g. {address: '127.0.0.1', port: 8080}.

Returns

Object	an address descriptor

Socket.prototype.setTimeout (timeout)

Enable/disable timeout with the specified timeout, in milliseconds. With this option set to a non-zero timeout, a read() on this socket's stream will block for only this amount of time.

Parameters

Number	**timeout**	timeout in milliseconds

Module system

This module provides an implementation of the system module compliant to the CommonJS System/1.0 specification. Beyond the standard a print() function is provided.

Functions

- exit (status)
- print ()

Properties

- args
- env
- stderr
- stdin
- stdout

args

An array of strings representing the command line arguments passed to the running script.

Example

```
>> ringo .\myScript.js foo bar baz 12345
system.args -> ['.\myScript.js', 'foo', 'bar', 'baz', '12345']
```

---

env

An object containing of the current system environment.

Example

```
{
  USERPROFILE: 'C:\Users\username',
  JAVA_HOME: 'C:\Program Files\Java\jdk1.7.0_07\',
  SystemDrive: 'C:',
  Path: '%System%/...',
  PROCESSOR_REVISION: '1a05',
  USERDOMAIN: 'EXAMPLE',
  SESSIONNAME: 'Console',
  TMP: 'C:\Temp',
  PROMPT: '$P$G',
  PROCESSOR_LEVEL: '6',
  LOCALAPPDATA: 'C:\Local',
  ...
}
```

See

java.lang.System.getenv()

---

exit (status)

Terminates the current process.

Parameters

number	**status**	The exit status, defaults to 0.

---

print ()

A utility function to write to stdout.

---

stderr

A TextStream to write to stderr.

---

stdin

A TextStream to read from stdin.

---

stdout

A TextStream to write to stdout.

Module test

A test runner compliant to the CommonJS Unit Testing specification. It manages the execution of unit tests and processes test results. The runner reports the total number of failures as exit status code.

The runner treats a module like a test case. A test case defines the fixture to run multiple tests. Test cases can provide optional setUp() and tearDown() functions to initialize and destroy the fixture. The test runner will run these methods prior to / after each test.

The following example test case testDatabase.js starts a new test runner if executed with ringo testDatabase.js

Example

```
// testDatabase.js
exports.setUp = function() { ... open db connection ... }
exports.tearDown = function() { ... close db connection ... }

exports.testCreateTable = function() { ... }
exports.testInsertData = function() { ... }
exports.testTransactions = function() { ... }
exports.testDeleteTable = function() { ... }

if (require.main == module.id) {
  // Get a runner and run on the current module
  require("test").run(exports);
}
```

See

The assert module is an assertion library to write unit tests.

Functions

getStackTrace (trace)

getType (obj)

jsDump (value, lvl)

run (scope, name, writer)

getStackTrace (trace)

Creates a stack trace and parses it for display.

Parameters

java.lang.StackTraceElement	**trace**	The trace to parse. If not given a stacktrace will be generated

Returns

String	The parsed stack trace

---

getType (obj)

Returns the type of the object passed as argument.

Parameters

	obj

Returns

String	The type of the object passed as argument

---

jsDump (value, lvl)

Converts the value passed as argument into a nicely formatted and indented string

Parameters

Object	**value**	The value to convert into a string
Number	**lvl**	Optional indentation level (defaults to zero)

Returns

String	The string representation of the object passed as argument

---

run (scope, name, writer)

The main runner method. This method can be called with one, two or three arguments: run(scope), run(scope, nameOfTest), run(scope, writer) or run(scope, nameOfTest, writer)

Parameters

String|Object	**scope**	Either the path to a module containing unit tests to execute, or an object containing the exported test methods or nested scopes.
String	**name**	Optional name of a test method to execute
Object	**writer**	Optional writer to use for displaying the test results. Defaults to TermWriter.

Module ringo/args

A parser for command line options. This parser supports various option formats:

- -a -b -c (multiple short options)
- -abc (multiple short options combined into one)
- -a value (short option with value)
- -avalue (alternative short option with value)
- --option value (long option with value)
- --option=value (alternative long option with value)

Example

```
// ringo parserExample.js -v --size 123 -p 45678

include('ringo/term');
var system = require('system');
var {Parser} = require('ringo/args');

var parser = new Parser();
parser.addOption('s', 'size', 'SIZE', 'Sets the size to SIZE');
parser.addOption('p', 'pid', 'PID', 'Kill the process with the PID');
parser.addOption('v', 'verbose', null, 'Verbosely do something');
parser.addOption('h', 'help', null, 'Show help');

var options = parser.parse(system.args.slice(1));
if (options.help) {
  writeln(parser.help());
} else {
  if (options.size) {
     writeln('Set size to ' + parseInt(options.size));
  }

  if (options.pid) {
     writeln('Kill process ' + options.pid);
  }

  if (options.verbose) {
     writeln('Verbose!');
  }
}

if (!Object.keys(options).length) {
  writeln("Run with -h/--help to see available options");
}
```

Class Parser

Ringo Modules

- Modules
- Overview

Instance Methods

addOption (shortName, longName, argument, helpText)

help ()

parse (args, result)

Parser ()

Create a new command line option parser.

Parser.prototype.addOption (shortName, longName, argument, helpText)

Add an option to the parser.

Parameters

String	**shortName**	the short option name (without leading hyphen)
String	**longName**	the long option name (without leading hyphens)
String	**argument**	the name of the argument if the option requires one, or null
String	**helpText**	the help text to display for the option

Returns

Object	this parser for chained invocation

Parser.prototype.help ()

Get help text for the parser's options suitable for display in command line scripts.

Returns

String	a string explaining the parser's options

Parser.prototype.parse (args, result)

Parse an arguments array into an option object. If a long option name is defined, it is converted to camel-case and used as property name. Otherwise, the short option name is used as property name.

Passing an result object as second argument is a convenient way to define default options:

Example

```
parser.parse(system.args.slice(1), {myOption: "defaultValue"});
```

Parameters

Array	**args**	the argument array. Matching options are removed.
Object	**result**	optional result object. If undefined, a new Object is created.

Returns

Object	the result object

See

toCamelCase()

Module ringo/base64

Base64 encoding and decoding for binary data and strings.

Example

```
>> var base64 = require('ringo/base64');
>> var enc = base64.encode('Hello World!', 'ISO-8859-15');
>> print(enc);
'SGVsbG8gV29ybGQh'
>> print(base64.decode(enc, 'ISO-8859-15'));
Hello World!
```

Functions

decode (str, encoding)

encode (str, encoding)

decode (str, encoding)

Decodes a Base64 encoded string to a string or byte array.

Parameters

String	**str**	the Base64 encoded string
String	**encoding**	the encoding to use for the return value. Defaults to 'utf8'. Use 'raw' to get a ByteArray instead of a string.

Returns

the decoded string or ByteArray

encode (str, encoding)

Encode a string or binary to a Base64 encoded string

Parameters

String|Binary	**str**	a string or binary
String	**encoding**	optional encoding to use if first argument is a string. Defaults to

Ringo Modules

- Modules
- Overview

assert
binary
console
fs
globals
io
net
system
test
ringo/args
ringo/base64
ringo/buffer
ringo/concurrent
ringo/daemon
ringo/encoding
ringo/engine
ringo/events
ringo/httpclient
ringo/httpserver
ringo/jsdoc
ringo/logging
ringo/markdown
ringo/mime
ringo/mustache
ringo/parser
ringo/profiler
ringo/promise
ringo/shell
ringo/subprocess
ringo/term
ringo/worker
ringo/zip
ringo/jsgi/connector
ringo/jsgi/response
ringo/utils/arrays
ringo/utils/dates
ringo/utils/files
ringo/utils/http
ringo/utils/numbers
ringo/utils/objects
ringo/utils/strings

'utf8'.

Returns

the Base64 encoded string

Module ringo/buffer

A simple text Buffer class for composing strings.

Class Buffer

Instance Methods

- digest (algorithm)
- forEach (fn)
- reset ()
- toString ()
- write (...)
- writeln (...)

Instance Properties

- length

Buffer (...)

A Buffer class for composing strings. This is implemented as a simple wrapper around a JavaScript array.

Parameters

...	initial parts to write to the buffer

Buffer.prototype.digest (algorithm)

Get a message digest on the content of this buffer.

Parameters

algorithm	the algorithm to use, defaults to MD5

Buffer.prototype.forEach (fn)

Call function fn with each content part in this buffer.

Parameters

fn a function to apply to each buffer part

Buffer.prototype.length

A read-only property containing the number of characters currently contained by this buffer.

Buffer.prototype.reset ()

Reset the buffer discarding all its content.

Returns

this buffer object

Buffer.prototype.toString ()

Return the content of this buffer as string.

Buffer.prototype.write (...)

Append all arguments to this buffer.

Parameters

... variable arguments to append to the buffer

Returns

this buffer object

Buffer.prototype.writeln (...)

Append all arguments to this buffer terminated by a carriage return/newline sequence.

Parameters

... variable arguments to append to the buffer

Returns

this buffer object

Module ringo/concurrent

Utilities for working with multiple concurrently running threads.

Class Semaphore

Instance Methods

- signal (permits)
- tryWait (timeout, permits)
- wait (permits)

Semaphore (permits)

A counting semaphore that can be used to coordinate and synchronize cooperation between synchronous threads. A semaphore keeps a number of permits.

Note that Worker events are usually run in the single thread of the local event loop and thus don't require synchronization provided by semaphores. The only case when you may want to use a semaphore with workers is when setting the syncCallbacks flag as second argument to Worker.postMessage() since this will cause callbacks from the worker to be run in their own thread instead of the event loop.

To synchronize threads using a semaphore, a threads may ask for one or more permits using the wait and tryWait methods. If the requested number of permits is available, they are subtracted from the number of permits in the semaphore and the method returns immediately.

If the number of requested permits is not available, the wait and tryWait methods block until another thread adds the required permits using the signal method or, in the case of tryWait, the specified timeout expires.

Parameters

permits	the number of initial permits, defaults to 0

Semaphore.prototype.signal (permits)

Add one or more permits to the semaphore.

Parameters

permits	the number of permits to give, defaults to 1

Semaphore.prototype.tryWait (timeout, permits)

Wait for one or more permits for the given span of time. Returns true if the requested permits could be acquired before the timeout elapsed.

Parameters

timeout	The span of time to wait, in milliseconds
permits	the number of permits to wait for, defaults to 1

Returns

true if the requested permits could be acquired, false if the timeout elapsed

---

Semaphore.prototype.wait (permits)

Wait for one or more permits.

Parameters

permits	the number of permits to wait for, defaults to 1

Module ringo/daemon

The daemon control script invoked by the init script.

This module interprets the first command line argument as module ID, load the module and try to invoke the life cycle functions on it.

For HTTP servers it is generally more convenient to directly use ringo/httpserver which will create a new server instance and pass it to as argument to the application life cycle functions.

Functions

- destroy ()
- init ()
- start ()
- stop ()

destroy ()

Called when the daemon is destroyed.

---

init ()

Called when the daemon instance is created.

This function can be run with superuser id to perform privileged actions before the daemon is started.

---

start ()

Called when the daemon instance is started.

---

stop ()

Called when the daemon is stopped.

Module ringo/encoding

Low-level support for character encoding and decoding.

Class Decoder

Instance Methods

- clear ()
- close ()
- decode (bytes, start, end)
- hasPendingInput ()
- read ()
- readFrom (source)
- readLine (includeNewline)
- toString ()

Instance Properties

- length

Class Encoder

Instance Methods

- clear ()
- close ()
- encode (string, start, end)
- toByteArray ()
- toByteString ()
- toString ()
- writeTo (sink)

Instance Properties

- length

Decoder (charset, strict, capacity)

Parameters

charset
strict

capacity

Decoder.prototype.clear ()

Decoder.prototype.close ()

Decoder.prototype.decode (bytes, start, end)

Decode bytes from the given buffer.

Parameters

binary.Binary	**bytes**	a ByteString or ByteArray
Number	**start**	The start index, or 0 if undefined
Number	**end**	the end index, or bytes.length if undefined

Decoder.prototype.hasPendingInput ()

Decoder.prototype.length

Decoder.prototype.read ()

Decoder.prototype.readFrom (source)

Parameters

binary.Binary	**source**

Decoder.prototype.readLine (includeNewline)

Parameters

Boolean	**includeNewline**

Decoder.prototype.toString ()

Encoder (charset, strict, capacity)

Parameters

- **charset**
- **strict**
- **capacity**

Encoder.prototype.clear ()

Encoder.prototype.close ()

Encoder.prototype.encode (string, start, end)

Parameters

String	**string**
Number	**start**
Number	**end**

Encoder.prototype.length

Encoder.prototype.toByteArray ()

Encoder.prototype.toByteString ()

Encoder.prototype.toString ()

Encoder.prototype.writeTo (sink)

Parameters

sink

Module ringo/engine

Provides access to the Rhino JavaScript engine.

Functions

- addHostObject (javaClass)
- addRepository (repo)
- addShutdownHook (funcOrObject, sync)
- asJavaObject (object)
- asJavaString (object)
- createSandbox (modulePath, globals, options)
- getCurrentWorker (obj)
- getErrors ()
- getOptimizationLevel ()
- getRepositories ()
- getRhinoContext ()
- getRhinoEngine ()
- getRingoHome ()
- getWorker ()
- setOptimizationLevel (level)

Properties

- version

addHostObject (javaClass)

Define a class as Rhino host object.

Parameters

JavaClass	**javaClass**	the class to define as host object

---

addRepository (repo)

Add a repository to the module search path

Parameters

Repository	**repo**	a repository

---

addShutdownHook (funcOrObject, sync)

Register a callback to be invoked when the current RingoJS instance is terminated.

Parameters

Function|Object	**funcOrObject**	Either a JavaScript function or a JavaScript object containing properties called \`module\` and \`name\` specifying a function exported by a RingoJS module.
Boolean	**sync**	(optional) whether to invoke the callback synchronously (on the main shutdown thread) or asynchronously (on the worker's event loop thread)

---

asJavaObject (object)

Get a wrapper for an object that exposes it as Java object to JavaScript.

Parameters

Object	**object**	an object

Returns

Object	the object wrapped as native java object

---

asJavaString (object)

Get a wrapper for a string that exposes the java.lang.String methods to JavaScript This is useful for accessing strings as java.lang.String without the cost of creating a new instance.

Parameters

Object	**object**	an object

Returns

Object	the object converted to a string and wrapped as native java object

---

createSandbox (modulePath, globals, options)

Create a sandboxed scripting engine with the same install directory as this and the given module paths, global properties, class shutter and sealing

Parameters

Array	**modulePath**	the comma separated module search path
Object	**globals**	a map of predefined global properties (may be undefined)
Object	**options**	an options object (may be undefined). The following options are supported: - systemModules array of system module directories to add to the module search path (may be relative to the ringo install dir) - classShutter a Rhino class shutter, may be null - sealed if the global object should be sealed, defaults to false

Returns

RhinoEngine	a sandboxed RhinoEngine instance

Throws

{FileNotFoundException} if any part of the module paths does not exist

---

getCurrentWorker (obj)

Get the worker instance associated with the current thread or the given scope or function object.

Parameters

obj	{Object} optional scope or function to get the worker from.

Returns

org.ringojs.engine.RingoWorker	the current worker

---

getErrors ()

Get a list containing the syntax errors encountered in the current worker.

Returns

ScriptableList	a list containing the errors encountered in the current worker

---

getOptimizationLevel ()

Get the Rhino optimization level for the current thread and context. The optimization level is an integer between -1 (interpreter mode) and 9 (compiled mode, all optimizations enabled). The default level is 0.

Returns

Number	level an integer between -1 and 9

---

getRepositories ()

Get the app's module search path as list of repositories.

Returns

ScriptableList	a list containing the module search path repositories

---

getRhinoContext ()

Get the org.mozilla.javascript.Context associated with the current thread.

getRhinoEngine ()

Get the org.ringojs.engine.RhinoEngine associated with this application.

Returns

org.ringojs.engine.RhinoEngine	the current RhinoEngine instance

---

getRingoHome ()

Get the RingoJS installation directory.

Returns

Repository	a Repository representing the Ringo installation directory

---

getWorker ()

Get a new worker instance.

Returns

org.ringojs.engine.RingoWorker	a new RingoWorker instance

---

setOptimizationLevel (level)

Set the Rhino optimization level for the current thread and context. The optimization level is an integer between -1 (interpreter mode) and 9 (compiled mode, all optimizations enabled). The default level is 0.

Parameters

Number	**level**	an integer between -1 and 9

---

version

The RingoJS version as an array-like object with the major and minor version number as first and second element.

Module ringo/events

Exports an EventEmitter classes that provide methods to emit events and register event listener functions.

Class EventEmitter

Instance Methods

- emit (type, [args...])
- listeners (type)
- on (type, listener)
- removeAllListeners (type)
- removeListener (type, listener)

Class JavaEventEmitter

Instance Methods

- addListener (type, listener)
- addSyncListener (type, listener)
- emit (type, [args...])
- on (type, listener)
- removeAllListeners (type)
- removeListener (type, listener)

Instance Properties

- impl

EventEmitter ()

This class provides methods to emit events and add or remove event listeners.

The EventEmitter function can be used as constructor or as mix-in. Use the new keyword to construct a new EventEmitter:

```
var emitter = new EventEmitter();
```

To add event handling methods to an existing object, call or apply the EventEmitter function with the object as this:

```
EventEmitter.call(object);
```

EventEmitter.prototype.emit (type, [args...])

Emit an event to all listeners registered for this event type

Parameters

string	**type**	type the event type
...	**[args...]**	optional arguments to pass to the listeners

Returns

true if the event was handled by at least one listener, false otherwise

Throws

Error if the event type was "error" and there were no listeners

---

EventEmitter.prototype.listeners (type)

Get an array containing the listener functions for the given event. If no listeners exist for the given event a new array is created. Changes on the return value will be reflected in the EventEmitter instance.

Parameters

string	**type**	the event type

Returns

array	the lister array

EventEmitter.prototype.on (type, listener)

Add a listener function for the given event. This is a shortcut for addListener()

Parameters

string	**type**	the event type
function	**listener**	the listener

Returns

this object for chaining

EventEmitter.prototype.removeAllListeners (type)

Remove all listener function for the given event.

Parameters

string	**type**	the event type

Returns

this object for chaining

EventEmitter.prototype.removeListener (type, listener)

Remove a listener function for the given event.

Parameters

string	**type**	the event type
function	**listener**	the listener

Returns

this object for chaining

JavaEventEmitter (classOrInterface, eventMapping)

An adapter for dispatching Java events to Ringo. This class takes a Java class or interface as argument and creates a Java object that extends or implements the class or interface and forwards method calls to event listener functions registered using the EventEmitter methods.

Like EventEmitter, JavaEventEmitter can be used as constructor or as mix-in. Use the new keyword to construct a new JavaEventEmitter:

```
var emitter = new JavaEventEmitter(JavaClassOrInterface);
```

To add event handling methods to an existing object, call or apply the JavaEventEmitter function with the object as this:

```
JavaEventEmitter.call(object, JavaClassOrInterface);
```

JavaEventEmitter accepts an object as optional second argument that maps Java methods to event names. If the first argument is a Java class this mapping also allows to select which methods should be overridden. If called without event mapping the method name is used as event name, except for methods like onFoo which will trigger event foo.

Parameters

classOrInterface	a Java class or interface, or an Array containing multiple Java interfaces.
eventMapping	optional object mapping method names to event names. If this parameter is defined only methods whose name is a property key in the object will be overridden, and the event type will be set to the property value instead of the method name.

JavaEventEmitter.prototype.addListener (type, listener)

Add a listener function for the given event. The function will be called asynchronously on the thread of the local event loop.

Parameters

string	**type**	the event type
function	**listener**	the listener

JavaEventEmitter.prototype.addSyncListener (type, listener)

Add a synchronous listener function for the given event. A synchronous listener will be called by an outside thread instead of the thread of the local event loop. This means it can be called concurrently while this worker is running other code.

Parameters

string	**type**	the event type
function	**listener**	the listener

JavaEventEmitter.prototype.emit (type, [args...])

Emit an event to all listeners registered for this event type

Parameters

string	**type**	type the event type
...	**[args...]**	optional arguments to pass to the listeners

Returns

true if the event was handled by at least one listener, false otherwise

Throws

Error if the event type was "error" and there were no listeners

---

JavaEventEmitter.prototype.impl

The generated Java object. This implements the Java interface passed to the JavaEventEmitter constructor and can be passed to Java code that expects given interface.

JavaEventEmitter.prototype.on (type, listener)

Add a listener function for the given event. This is a shortcut for addListener()

Parameters

string	**type**	the event type
function	**listener**	the listener

JavaEventEmitter.prototype.removeAllListeners (type)

Removes all listener functions for a given event.

Parameters

string	**type**	the event type

JavaEventEmitter.prototype.removeListener (type, listener)

Remove a listener function for the given event.

Parameters

string	**type**	the event type
function	**listener**	the listener

Module ringo/httpclient

A module for sending HTTP requests and receiving HTTP responses.

Example

```
var {request} = require('ringo/httpclient');
var exchange = request({
   method: 'GET',
   url: 'http://ringojs.org/',
   headers: {
      'x-custom-header': 'foobar'
   }
});

if(exchange.status == 200) {
   console.log(exchange.content);
}
```

Functions

- del (url, data, success, error)
- get (url, data, success, error)
- post (url, data, success, error)
- put (url, data, success, error)
- request (options)

Class BinaryPart

Class Exchange

Instance Properties

- connection
- content
- contentBytes
- contentLength
- contentType
- cookies
- done
- encoding

Ringo Modules

- Modules
- Overview

assert
binary
console
fs
globals
io
net
system
test
ringo/args
ringo/base64
ringo/buffer
ringo/concurrent
ringo/daemon
ringo/encoding
ringo/engine
ringo/events
ringo/httpclient
ringo/httpserver
ringo/jsdoc
ringo/logging
ringo/markdown
ringo/mime
ringo/mustache
ringo/parser
ringo/profiler
ringo/promise
ringo/shell
ringo/subprocess
ringo/term
ringo/worker
ringo/zip
ringo/jsgi/connector
ringo/jsgi/response
ringo/utils/arrays
ringo/utils/dates
ringo/utils/files
ringo/utils/http
ringo/utils/numbers
ringo/utils/objects
ringo/utils/strings

headers
message
status
url

Class TextPart

BinaryPart (data, charset, filename)

Parameters

String	**data**	The data
String	**charset**	The charset
String	**filename**	An optional file name

Returns

BinaryPart	A newly constructed BinaryPart instance

Exchange (url, options, callbacks)

Parameters

String	**url**	The URL
Object	**options**	The options
Object	**callbacks**	An object containing success, error and complete callback methods

Returns

Exchange	A newly constructed Exchange instance

Exchange.prototype.connection

The connection used by this Exchange instance

Exchange.prototype.content

The response body as String

Exchange.prototype.contentBytes

The response body as ByteArray

Exchange.prototype.contentLength

The response content length

Exchange.prototype.contentType

The response content type

Exchange.prototype.cookies

The cookies set by the server

Exchange.prototype.done

True if the request has completed, false otherwise

Exchange.prototype.encoding

The response encoding

Exchange.prototype.headers

The response headers

Exchange.prototype.message

The response status message

Exchange.prototype.status

The response status code

Exchange.prototype.url

The URL wrapped by this Exchange instance

TextPart (data, charset, filename)

Parameters

String|TextStream	**data**	The data
String	**charset**	The charset
String	**filename**	An optional file name

Returns

TextPart	A newly constructed TextPart instance

---

del (url, data, success, error)

Executes a DELETE request

Parameters

String	**url**	The URL
Object|String	**data**	The data to append as GET parameters to the URL
Function	**success**	Optional success callback
Function	**error**	Optional error callback

Returns

Exchange	The Exchange instance representing the request and response

---

get (url, data, success, error)

Executes a GET request

Parameters

String	**url**	The URL

Object|String	**data**	The data to append as GET parameters to the URL
Function	**success**	Optional success callback
Function	**error**	Optional error callback

Returns

Exchange	The Exchange instance representing the request and response

---

post (url, data, success, error)

Executes a POST request

Parameters

String	**url**	The URL
Object|String|Stream|Binary	**data**	The data to send to the server
Function	**success**	Optional success callback
Function	**error**	Optional error callback

Returns

Exchange	The Exchange instance representing the request and response

---

put (url, data, success, error)

Executes a PUT request

Parameters

String	**url**	The URL
Object|String|Stream|Binary	**data**	The data send to the server
Function	**success**	Optional success callback
Function	**error**	Optional error callback

Returns

Exchange	The Exchange instance representing the request and response

---

request (options)

Make a generic request.

Generic request options

The options object may contain the following properties:

- url: the request URL
- method: request method such as GET or POST
- data: request data as string, object, or, for POST or PUT requests, Stream or Binary.
- headers: request headers
- username: username for HTTP authentication
- password: password for HTTP authentication
- proxy: proxy-settings as string ("proxy.host:port") or object {host: "hostname.org", port: 3128}
- contentType: the contentType
- binary: if true if content should be delivered as binary, else it will be decoded to string
- followRedirects: whether HTTP redirects (response code 3xx) should be automatically followed; default: true
- readTimeout: setting for read timeout in millis. 0 return implies that the option is disabled (i.e., timeout of infinity); default: 0 (or until impl decides its time)
- connectTimeout: Sets a specified timeout value, in milliseconds, to be used when opening a communications link to the resource referenced by this URLConnection. A timeout of zero is interpreted as an infinite timeout.; default: 0 (or until impl decides its time)

Callbacks

The options object may also contain the following callback functions:

- complete: called when the request is completed
- success: called when the request is completed successfully
- error: called when the request is completed with an error
- beforeSend: called with the Exchange object as argument before the request is sent

The following arguments are passed to the complete, success and part callbacks:

1. content: the content as String or ByteString
2. status: the HTTP status code
3. contentType: the content type
4. exchange: the exchange object

The following arguments are passed to the error callback:

1. message: the error message. This is either the message from an exception thrown during request processing or an HTTP error message
2. status: the HTTP status code. This is 0 if no response was received
3. exchange: the exchange object

Parameters

Object	**options**

Returns

Exchange	exchange object

See

get
post
put
del

Module ringo/httpserver

A wrapper for the Jetty HTTP server.

Functions

destroy ()

init (appPath)

main (appPath)

start ()

stop ()

Class Context

Instance Methods

addServlet (servletPath, servlet, initParams)

addWebSocket (path, onconnect)

serveApplication (app, engine)

serveStatic (dir)

Class Server

Instance Methods

destroy ()

getContext (path, virtualHosts, options)

getDefaultContext ()

getJetty ()

isRunning ()

start ()

stop ()

Class WebSocket

Instance Methods

close ()

isOpen ()

send (msg)

sendBinary (bytearray, offset, length)

Context

Not exported as constructor by this module.

See

Server.prototype.getContext
Server.prototype.getDefaultContext

Context.prototype.addServlet (servletPath, servlet, initParams)

Map a request path within this context to the given servlet.

Parameters

string	**servletPath**	the servlet path
Servlet	**servlet**	a java object implementing the javax.servlet.Servlet interface.
Object	**initParams**	optional object containing servlet init parameters

---

Context.prototype.addWebSocket (path, onconnect)

Start accepting WebSocket connections in this context context.

Parameters

String	**path**	The URL path on which to accept WebSocket connections
Function	**onconnect**	a function called for each new WebSocket connection with the WebSocket object as argument.

See

WebSocket

Context.prototype.serveApplication (app, engine)

Map this context to a JSGI application.

Parameters

function|object	**app**	a JSGI application, either as a function or an object with properties

		appModule and appName defining the application. { appModule: 'main', appName: 'app' }
RhinoEngine	**engine**	optional RhinoEngine instance for multi-engine setups

---

Context.prototype.serveStatic (dir)

Map this context to a directory containing static resources.

Parameters

string	**dir**	the directory from which to serve static resources

---

Server (options)

Create a Jetty HTTP server with the given options. The options may either define properties to be used with the default jetty.xml, or define a custom configuration file.

Parameters

Object	**options**	A javascript object with any of the following properties (default values in parentheses): • jettyConfig ('config/jetty.xml') • port (8080) • host (undefined) • sessions (true) • security (true) • cookieName (null) • cookieDomain (null) • cookiePath (null) • httpOnlyCookies (false) • secureCookies (false) For convenience, the constructor supports the definition of a JSGI application and static resource mapping in the options object using the following properties: • virtualHost (undefined) • mountpoint ('/') • staticDir ('static') • staticMountpoint ('/static')

- appModule ('main')
- appName ('app')

Server.prototype.destroy ()

Destroy the HTTP server, freeing its resources.

Server.prototype.getContext (path, virtualHosts, options)

Get a servlet application context for the given path and virtual hosts, creating it if it doesn't exist.

Parameters

string	**path**	the context root path such as "/" or "/app"
string|array	**virtualHosts**	optional single or multiple virtual host names. A virtual host may start with a "*." wildcard.
Object	**options**	may have the following properties: sessions: true to enable sessions for this context, false otherwise security: true to enable security for this context, false otherwise cookieName: optional cookie name cookieDomain: optional cookie domain cookiePath: optional cookie path httpOnlyCookies: true to enable http-only session cookies secureCookies: true to enable secure session cookies

Returns

a Context object

See

Context

Server.prototype.getDefaultContext ()

Get the server's default context. The default context is the context that is created when the server is created.

Returns

the default Context

See

Context

Server.prototype.getJetty ()

Get the Jetty server instance

Returns

the Jetty Server instance

Server.prototype.isRunning ()

Checks whether this server is currently running.

Returns

true if the server is running, false otherwise.

Server.prototype.start ()

Start the HTTP server.

Server.prototype.stop ()

Stop the HTTP server.

WebSocket

Provides support for WebSockets in the HTTP server.

WebSocket is an event emitter that supports the following events:

- **open**: called when a new websocket connection is accepted
- **message**: Called with a complete text message when all fragments have been received.
- **close**: called when an established websocket connection closes

---

WebSocket.prototype.close ()

Closes the WebSocket connection.

WebSocket.prototype.isOpen ()

Check whether the WebSocket is open.

Returns

Boolean	true if the connection is open

WebSocket.prototype.send (msg)

Send a string over the WebSocket.

Parameters

String	**msg**	a string

WebSocket.prototype.sendBinary (bytearray, offset, length)

Send a byte array over the WebSocket.

Parameters

ByteArray	**bytearray**	The byte array to send
Number	**offset**	Optional offset (defaults to zero)
Number	**length**	Optional length (defaults to the length of the byte array)

---

destroy ()

Daemon life cycle function invoked by init script. Frees any resources occupied by the Server instance. If the application exports a function called destroy, it will be invoked with the server as argument.

Returns

Server	the Server instance.

---

init (appPath)

Daemon life cycle function invoked by init script. Creates a new Server with the application at appPath. If the application exports a function called init, it will be invoked with the new server as argument.

Parameters

appPath	{string} optional application file name or module id. If undefined, the first command line argument will be used as application. If there are no command line arguments, module `main` in the current working directory is used.

Returns

Server	the Server instance.

---

main (appPath)

Main function to start an HTTP server from the command line.

Parameters

String	**appPath**	optional application file name or module id.

Returns

Server	the Server instance.

---

start ()

Daemon life cycle function invoked by init script. Starts the

Server created by init(). If the application exports a function called start, it will be invoked with the server as argument immediately after it has started.

Returns

Server	the Server instance.

---

stop ()

Daemon life cycle function invoked by init script. Stops the Server started by start().

Returns

Server	the Server instance. If the application exports a function called \`stop\`, it will be invoked with the server as argument immediately before it is stopped.

Module ringo/jsdoc

Low level support for parsing JSDoc-style comments from JavaScript files.

Functions

parseResource (resource)

Class ScriptRepository

Instance Methods

exists ()

getPath ()

getScriptResource (path)

getScriptResources (nested)

ScriptRepository (path)

Create a script repository for the given path

Parameters

String	**path**	the base path

Returns

an script repository

ScriptRepository.prototype.exists ()

Check whether this script repository exists.

Returns

boolean	true if the repository exists

ScriptRepository.prototype.getPath ()

Get the absolute path of this script repository.

Returns

string	the absolute repository path

ScriptRepository.prototype.getScriptResource (path)

Get a script resource contained in this repository.

Parameters

String	**path**	the script path

Returns

Resource	the script resource

ScriptRepository.prototype.getScriptResources (nested)

Get a list of script resources (files with a .js extension) in this repository.

Parameters

Boolean	**nested**	whether to return scripts in nested directories

Returns

Array	list of script files as RingoJS Resource objects

---

parseResource (resource)

Parse a script resource and return an array containing the JSDoc items for the properties it exports.

Parameters

Resource	**resource**	a script resource

Returns

Array	an array of objects representing the API documentation for of the resource

Module ringo/logging

This module provides generic logging support for Ringo applications. It uses SLF4J or Apache log4j if either is detected in the classpath, and will fall back to java.util.logging otherwise.

If the first argument passed to any of the logging methods is a string containing any number of curly bracket pairs ({}), the logger will interpret it as format string and use any following arguments to replace the curly bracket pairs. If an argument is an Error or Java Exception object, the logger will render a stack trace for it and append it to the log message.

This module's exports object implements the EventEmitter interface and emits logged messages using the log level name as event type.

Example

```
// Get a Logger for the current module
var log = require('ringo/logging').getLogger(module.id);

log.debug('Connected to ', url, ' [GET]');
log.error('This should not occur');
log.info('Info message');
log.info('User {} accessed {}', username, resource);
log.warn('A warning');
```

Functions

- getJavaStack (error, prefix)
- getLogger (name)
- getScriptStack (error, prefix)
- setConfig (resource, watchForUpdates)

Class Logger

Instance Methods

- debug ()

- error ()
- info ()
- isDebugEnabled ()
- isErrorEnabled ()
- isInfoEnabled ()
- isTraceEnabled ()
- isWarnEnabled ()
- trace ()
- warn ()

Logger (name, impl)

Logger class. This constructor is not exported, use this module's {@link getLogger} to get a logger instance.

Parameters

name	the Logger name
impl	the logger implementation

See

getLogger

Logger.prototype.debug ()

Logger.prototype.error ()

Logger.prototype.info ()

Logger.prototype.isDebugEnabled ()

Logger.prototype.isErrorEnabled ()

Logger.prototype.isInfoEnabled ()

Logger.prototype.isTraceEnabled ()

Logger.prototype.isWarnEnabled ()

Logger.prototype.trace ()

Logger.prototype.warn ()

---

getJavaStack (error, prefix)

Get a rendered JavaScript stack trace from a caught error.

Parameters

Error	**error**	an error object
String	**prefix**	to prepend to result if available

Returns

String	the rendered JavaScript stack trace

---

getLogger (name)

Get a logger for the given name.

Parameters

string	**name**	the name of the logger

Returns

Logger	a logger instance for the given name

---

getScriptStack (error, prefix)

Get a rendered JavaScript stack trace from a caught error.

Parameters

Error	**error**	an error object
String	**prefix**	to prepend to result if available

Returns

String	the rendered JavaScript stack trace

---

setConfig (resource, watchForUpdates)

Configure log4j using the given file resource.

If you plan to update the configuration make sure to set the reset property to true in your configuration file.

Parameters

Resource	**resource**	the configuration resource in XML or properties format
Boolean	**watchForUpdates**	if true a scheduler thread is started that repeatedly checks the resource for updates.

Module ringo/markdown

A fast and extensible Markdown formatter.

Functions

process (text, [extension])

process (text, [extension])

Converts a string of Markdown formatted text to HTML.

Passing in an optional JavaScript object as argument allows the caller to override behaviour in the markdown processor. Specifically, the following methods can be overridden:

- **getLink(id)** called to resolve Markdown link ids. Takes a single string id as argument and must return an array containing the target link and the target link title. If this returns null, the markdown link will not be rendered as HTML link.
- **openTag(tagname, buffer)** called when a HTML tag is opened. tagname is an HTML tag such as pre or div, buffer is a java.lang.StringBuffer to append to. The function can be used to create HTML tags with additional attributes.

Parameters

String	**text**	a Markdown formatted text
Object	**[extension]**	optional object with methods overriding default behaviour in org.ringojs.util.MarkdownProcessor

Returns

String	the Markdown text converted to HTML

Module ringo/mime

This module provides functionality for determining the MIME type for a given file extension.

Example

```
>> var mime = require('ringo/mime');
>> mime.mimeType('photo.jpg');
'image/jpeg'
>> mime.mimeType('video.m4v');
'video/mp4'
>> mime.mimeType('feed.rss');
'application/rss+xml'
```

Functions

mimeType (fileName, fallback)

Properties

MIME_TYPES

MIME_TYPES

A list of common MIME types, keyed by file extension.

---

mimeType (fileName, fallback)

Determines the MIME type for the given file extension. If the file extension is unknown, the fallback argument is returned. If that is undefined, the function returns "application/octet-stream".

Parameters

string	**fileName**	a file name
string	**fallback**	MIME type to return if file extension is unknown

Returns

string	the MIME type for the file name

Module ringo/mustache

CommonJS-compatible mustache.js module.

This version of mustache.js adds filters. If a tag or section name consists of several space-separated items, the items are evaluated one at a time, starting with the right-most item. If an item evaluates to a function, the result of the previous item is passed to it as argument.

Example

```
var template = 'Hello {{upper world}}!';
var data = {
    upper: function(str) { return str.toUpperCase() },
    world: 'world'
};
mustache.to_html(template, data);
// -> 'Hello WORLD!'
```

See

http://github.com/janl/mustache.js
http://github.com/hns/mustache.js

Functions

- to_html (template, data)

Properties

- name
- version

name

The name of this module.

---

to_html (template, data)

Renders template using data as context object.

Parameters

String	**template**	the template.
Object	**data**	the data object.

Returns

String	the formatted text.

---

version

The version of this module.

Module ringo/parser

This module provides an interface to the Rhino parser.

Functions

getName (node)

getTypeName (node)

isName (node)

Properties

Token

Class Parser

Instance Methods

parse (script, [encoding])

visit (script, visitorFunction, [encoding])

Parser (options)

Create a new Parser object. The constructor must be called with the new keyword. It takes an options argument which may contain the following properties:

- languageVersion (number) the JavaScript language version to use. Defaults to 180.
- parseComments (boolean) whether to attach jsdoc-style comments to parsed nodes. Defaults to false.

Parameters

Object	**options**	the parser options

Parser.prototype.parse (script, [encoding])

Parse a script resource and return its AST tree.

Parameters

Resource|String	**script**	a string or org.ringojs.repository.Resource object representing the script.
string	**[encoding]**	optional encoding to use, defaults to UTF-8

Returns

AstNode	the root node of the AST tree, an instance of org.mozilla.javascript.ast.AstRoot

Parser.prototype.visit (script, visitorFunction, [encoding])

Parse a script resource and apply the visitor function to its AST tree. The function takes one argument which is a org.mozilla.javascript.ast.AstNode. The function must return true to visit child nodes of the current node.

Parameters

Resource|String	**script**	a string or org.ringojs.repository.Resource object representing the script.
Function	**visitorFunction**	the visitor function
string	**[encoding]**	optional encoding to use, defaults to UTF-8

Token

The org.mozilla.javascript.Token class. This can be used to easily check find out the types of AST nodes:

```
node.type == Token.NAME
```

getName (node)

Utility function to get the name value of a node, or the empty string if it is not a NAME node.

Parameters

AstNode	**node**	an AST node

Returns

String	the name value of the node

---

getTypeName (node)

Get the type name of the token as string such as "CALL" or "NAME".

Parameters

AstNode	**node**	a AST node

Returns

String	the name of the AST node's type

---

isName (node)

Utility function to test whether a node is a NAME node (a node of type org.mozilla.javascript.ast.Name)

Parameters

Object	**node**	an AST node

Returns

Boolean	true if node is a name node

Module ringo/profiler

A profiler for measuring execution time of JavaScript functions. Note that you need to run with optimization level -1 for profiling to work. Running the profiler on optimized code will produce no data.

Functions

profile (func, maxFrames)

Class Profiler

Instance Methods

formatResult (maxFrames)

getFrames ()

getScriptFrame (cx, script)

toString ()

Profiler ()

A class for measuring the frequency and runtime of function invocations.

Profiler.prototype.formatResult (maxFrames)

Parameters

maxFrames

Profiler.prototype.getFrames ()

Profiler.prototype.getScriptFrame (cx, script)

Parameters

cx

script

Profiler.prototype.toString ()

---

profile (func, maxFrames)

Convenience function for profiling the invocation of a function.

Parameters

Function	**func**	the function to profile
number	**maxFrames**	optional maximal number of frames to include

Returns

Object	an object with the following properties: • result: the value returned by the function, if any • error: the error thrown by the function, if any • profiler: the Profiler instance used to profile the invocation

Module ringo/promise

Allows to work with deferred values that will be resolved in the future.

Class Deferred

Instance Methods

resolve (result, isError)

Instance Properties

promise

Class Promise

Instance Methods

then (callback, errback)

wait (timeout)

Class PromiseList

Deferred ()

Creates an object representing a deferred value. The deferred object has two properties: a promise object and a resolve() function.

The promise object can be used to register a callback to be invoked when the promise is eventually resolved.

The resolve function is used to resolve the promise as either fulfilled or failed.

Example

```
// Example for an asynchronous JSGI response.
// The response is generated after a one second delay.
exports.asyncAction = function(request) {
  var response = new Deferred();
  setTimeout(function() {
      response.resolve({
```

Ringo Modules

- Modules
- Overview

```
        status: 200, headers: {}, body: ["Delayed"]
      });
    }, 1000);
    return response.promise;
  }
```

Deferred.prototype.promise

The promise object can be used to register a callback to be invoked when the promise is eventually resolved.

Deferred.prototype.resolve (result, isError)

Resolve the promise.

Parameters

Object	**result**	the result or error value
boolean	**isError**	if true the promise is resolved as failed

Promise

A promise object. This class is not exported, create a deferred object to create a promise.

Promise.prototype.then (callback, errback)

Register callback and errback functions to be invoked when the promise is resolved.

Parameters

function	**callback**	called if the promise is resolved as fulfilled
function	**errback**	called if the promise is resolved as failed

Returns

Object	a new promise that resolves to the return value of the callback or errback when it is called.

Promise.prototype.wait (timeout)

Wait for the promise to be resolved.

Parameters

Number	**timeout**	optional time in milliseconds to wait for. If timeout is undefined wait() blocks forever.

Returns

Object	the value if the promise is resolved as fulfilled

Throws

Object the error value if the promise is resolved as failed

PromiseList (promise...)

The PromiseList class allows to combine several promises into one. It represents itself a promise that resolves to an array of objects, each containing a value or error property with the value or error of the corresponding promise argument.

A PromiseList resolves successfully even if some or all of the partial promises resolve to an error. It is the responsibility of the handler function to check each individual promise result.

Parameters

promise	**promise...**	any number of promise arguments.

Module ringo/shell

Provides functions to work with the Ringo shell.

Functions

- printError (xcept, errors, verbose)
- printResult (value, writer)
- quit (status)
- read ()
- readln (prompt, echoChar)
- start (engine)
- write ()
- writeln ()

printError (xcept, errors, verbose)

Parameters

Exception	**xcept**
Array	**errors**
Boolean	**verbose**

---

printResult (value, writer)

Parameters

	value
	writer

---

quit (status)

Quit the shell and exit the JVM.

Parameters

Number	**status**	optional integer exit status code (default is 0)

read ()

Read a single character from the standard input.

readln (prompt, echoChar)

Read a single line from the standard input.

Parameters

String	**prompt**	optional prompt to display
String	**echoChar**	character to use as echo, e.g. '*' for passwords or '' for no echo.

start (engine)

Start the shell programmatically. This uses the current thread and thus will not return. You should therefore call this function as the last statement in your script. Terminating the shell will exit the program.

Parameters

engine

write ()

Write 0..n arguments to standard output.

writeln ()

Write 0..n arguments to standard output, followed by a newline.

Module ringo/subprocess

A module for spawning processes, connecting to their input/output/errput and returning their response codes. It uses the current JVM's runtime provided by java.lang.Runtime.getRuntime(). The exact behavior of this module is highly system-dependent.

Functions

- command (command, [arguments...], [options])
- createProcess (args)
- status (command, [arguments...], [options])
- system (command, [arguments...], [options])

Class Process

Instance Methods

- connect (input, output, errput)
- kill ()
- wait ()

Instance Properties

- stderr
- stdin
- stdout

Process

The Process object can be used to control and obtain information about a subprocess started using createProcess().

See

http://docs.oracle.com/javase/8/docs/api/java/lang/Process.html

---

Process.prototype.connect (input, output, errput)

Connects the process's steams to the argument streams and starts threads to copy the data asynchronously.

Parameters

Stream	**input**	output stream to connect to the process's input stream
Stream	**output**	input stream to connect to the process's output stream
Stream	**errput**	input stream to connect to the process's error stream

---

Process.prototype.kill ()

Kills the subprocess.

Process.prototype.stderr

The process's error stream.

Process.prototype.stdin

The process's input stream.

Process.prototype.stdout

The process's output stream.

Process.prototype.wait ()

Wait for the process to terminate and return its exit status.

---

command (command, [arguments...], [options])

Executes a given command and returns the standard output. If the exit status is non-zero, throws an Error. Examples:

```
var {command} = require("ringo/subprocess");

// get PATH environment variable on Unix-like systems
var path = command("/bin/bash", "-c", "echo $PATH");

// a simple ping
var result = command("ping", "-c 1", "ringojs.org");
```

Parameters

String	**command**	command to call in the runtime environment
String	**[arguments...]**	optional arguments as single or multiple string parameters. Each argument is analogous to a quoted argument on the command line.
Object	**[options]**	options object. This may contain a \`dir\` string property specifying the directory to run the process in and a \`env\` object property specifying additional environment variable mappings.

Returns

String the standard output of the command

---

createProcess (args)

Low-level function to spawn a new process. The function takes an object argument containing the following properties where all properties except command are optional:

- command a string or array of strings containing the command to execute. Which string lists represent a valid operating system command is system-dependent.
- dir the directory to run the process in
- env alternative environment variables. If null the process inherits the environment of the current process.
- binary a boolean flag that uses raw binary streams instead of text streams
- encoding the character encoding to use for text streams

Parameters

Object	**args**	an object containing the process command and options.

Returns

a Process object

See

Process

---

status (command, [arguments...], [options])

Executes a given command quietly and returns the exit status.

Parameters

String	**command**	command to call in the runtime environment
String	**[arguments...]**	optional arguments as single or multiple string parameters. Each argument is analogous to a quoted argument on the command line.
Object	**[options]**	options object. This may contain a \`dir\` string property specifying the directory to run the process in and a \`env\` object property specifying additional environment variable mappings.

Returns

Number exit status

---

system (command, [arguments...], [options])

Executes a given command, attached to this process's output and error streams, and returns the exit status.

Parameters

String	**command**	command to call in the runtime environment
String	**[arguments...]**	optional arguments as single or multiple string parameters. Each argument is analogous to a quoted argument on the command line.
Object	**[options]**	options object. This may contain a \`dir\` string property specifying the directory to run the process in and a \`env\` object property

specifying additional environment variable mappings.

Returns

Number exit status

Module ringo/term

A module for printing ANSI terminal escape sequences. This module provides a number of useful color and style constants, and a replacement for the print function optimized for styled output.

Example

```
include('ringo/term')
writeln(YELLOW, "foo", MAGENTA, "bar");
// foo bar
writeln(YELLOW, ONBLUE, "IKEA");
// IKEA
writeln(RED, BOLD, INVERSE, "Red Alert!");
// Red Alert!
```

See

http://en.wikipedia.org/wiki/ANSI_escape_code

Functions

- write (args...)
- writeln (args...)

Properties

- BLACK
- BLUE
- BOLD
- CYAN
- GREEN
- INVERSE
- MAGENTA
- ONBLACK
- ONBLUE
- ONCYAN
- ONGREEN
- ONMAGENTA
- ONRED

- ONWHITE
- ONYELLOW
- RED
- RESET
- UNDERLINE
- WHITE
- YELLOW

Class TermWriter

Instance Methods

- isEnabled ()
- setEnabled (flag)
- write (args...)
- writeln (args...)

BLACK

BLUE

BOLD

CYAN

GREEN

INVERSE

MAGENTA

ONBLACK

ONBLUE

ONCYAN

ONGREEN

ONMAGENTA

ONRED

ONWHITE

ONYELLOW

RED

RESET

TermWriter (out)

Creates a terminal writer that writes to the given text output stream.

Parameters

Stream	**out**	a TextStream

TermWriter.prototype.isEnabled ()

Returns true if ANSI terminal colors are enabled.

Returns

true if ANSI is enabled.

TermWriter.prototype.setEnabled (flag)

Enable or disable ANSI terminal colors for this writer.

Parameters

boolean	**flag**	true to enable ANSI colors.

TermWriter.prototype.write (args...)

Write the arguments to the stream, applying ANSI terminal colors if enabled is true.

Parameters

args...	variable number of arguments to write

TermWriter.prototype.writeln (args...)

Write the arguments to the stream followed by a newline character, applying ANSI terminal colors if enabled is true.

Parameters

args...	variable number of arguments to write

UNDERLINE

WHITE

YELLOW

---

write (args...)

Write the arguments to system.stdout, applying ANSI terminal colors if support has been detected.

Parameters

args...	variable number of arguments to write

---

writeln (args...)

Write the arguments to system.stdout followed by a newline character, applying ANSI terminal colors if support has been detected.

Parameters

args...	variable number of arguments to write

Module ringo/worker

A Worker API based on the W3C Web Workers.

Class Worker

Instance Methods

postMessage (data, [syncCallbacks])

terminate ()

Class WorkerPromise

Instance Methods

then (callback, errback)

wait (timeout)

Worker (moduleId)

A Worker thread loosely modeled after the W3C Web Worker API.

The moduleId argument must be the fully resolved id of the module to load in the worker. In order to be able to send messages to the worker using the postMessage method the module must define (though not necessarily export) a onmessage function.

Workers operate on their own set of modules, so a new instance of the module will be created even if the module is already loaded in the current thread or is the same as the currently executing module. Thus, each worker operates in its private module environment, making concurrent programming much more predictable than with shared state multithreading.

Event listeners for callbacks from the worker can be registered by assigning them to the onmessage and onerror properties of the worker.

To free the worker's thread and other resources once the worker is no longer needed its terminate method should be called.

Parameters

String	**moduleId**	the id of the module to load in the worker.

Worker.prototype.postMessage (data, [syncCallbacks])

Post a message to the worker. This enqueues the message in the worker's input queue and returns immediately. The worker thread will then pick up the message and pass it to its onmessage function.

The argument passed to onmessage is an object with a data property containing the message and a source property containing an object with postMessage and postError methods allowing the worker to post messages or report errors back to the original caller.

If syncCallbacks is true, callbacks from the worker will run on the worker's own thread instead of our local event loop thread. This allows callbacks to be delivered concurrently while the local thread is busy doing something else.

Note that in contrast to the Web Workers specification this worker implementation does not require JSON serialization of messages.

Parameters

Object	**data**	the data to pass to the worker
Boolean	**[syncCallbacks]**	flag that indicates whether callbacks from the worker should be called synchronously in the worker's own thread rather than in our own local event loop thread.

Worker.prototype.terminate ()

Release the worker, returning it to the engine's worker pool. Note that this does not terminate the worker thread, or remove any current or future scheduled tasks from its event loop.

WorkerPromise (moduleId, message, [syncCallbacks])

A Promise backed by a Worker.

This creates a new Worker with the given moduleId and calls its postMessage function with the message argument. The first message or error received back from the worker will be used to resolve the promise.

The worker is terminated immediately after it resolves the promise.

Parameters

String	**moduleId**	the id of the module to load in the worker.
Object	**message**	the message to post to the worker.
Boolean	**[syncCallbacks]**	flag that indicates whether callbacks from the worker should be called synchronously in the worker's own thread rather than in our own local event loop thread.

See

ringo/promise.Promise

WorkerPromise.prototype.then (callback, errback)

Registers callback and errback functions that will be invoked when the promise is resolved by the worker. See documentation for Promise.then().

Parameters

function	**callback**	called if the promise is resolved as fulfilled
function	**errback**	called if the promise is resolved as failed

Returns

Returns

Object	a new promise that resolves to the return value of the callback or errback when it is called.

WorkerPromise.prototype.wait (timeout)

Wait for the worker to resolve the promise. See documentation for Promise.wait().

Parameters

Number	**timeout**	optional time in milliseconds to wait for. If timeout is undefined wait() blocks forever.

Returns

Object	the value if the promise is resolved as fulfilled

Throws

Object the error value if the promise is resolved as failed

Module ringo/zip

This module provides classes to uncompress zip files and streams.

Functions

ZipIterator (resource)

Class ZipFile

Instance Methods

close ()

getSize (name)

getTime (name)

isDirectory (name)

isFile (name)

open (name)

Instance Properties

entries

ZipFile (path)

A class to read and unpack a local zip file.

Parameters

String	**path**	the location of the zip file

ZipFile.prototype.close ()

Close the zip file.

ZipFile.prototype.entries

An array containing the names of all entries in this zip file.

ZipFile.prototype.getSize (name)

Returns the uncompressed size in bytes in the given entry, or -1 if not known.

Parameters

String	**name**	the entry name

ZipFile.prototype.getTime (name)

Returns the last modification timestamp of the given entry, or -1 if not available.

Parameters

String	**name**	the entry name

ZipFile.prototype.isDirectory (name)

Returns true if the entry with the given name represents a directory.

Parameters

String	**name**	the entry name

ZipFile.prototype.isFile (name)

Returns true if the entry with the given name represents a file.

Parameters

String	**name**	the entry name

ZipFile.prototype.open (name)

Get an input stream to read the entry with the given name.

Parameters

String	**name**	the entry name

ZipIterator (resource)

A streaming iterator over a zip file or stream. Each item yielded by this iterator is an input stream to read a single zip entry. Each entry stream has additional name, isDirectory, isFile, size, and time properties with the same semantics of the corresponding methods in ZipFile.

Parameters

Stream|String	**resource**	an input stream or file name

See

ZipFile

Module ringo/jsgi/connector

Low-level JSGI adapter implementation.

Functions

handleRequest (moduleId, functionObj, request)

Class AsyncResponse

Instance Methods

close ()

flush ()

start (status, headers)

write (data, [encoding])

AsyncResponse (request, timeout, autoflush)

Creates a streaming asynchronous response. The returned response object can be used both synchronously from the current thread or asynchronously from another thread, even after the original thread has finished execution. AsyncResponse objects are threadsafe.

Parameters

Object	**request**	the JSGI request object
Number	**timeout**	the response timeout in milliseconds. Defaults to 30 seconds.
Boolean	**autoflush**	whether to flush after each write.

AsyncResponse.prototype.close ()

Close the response stream, causing all buffered data to be written to the client.

AsyncResponse.prototype.flush ()

Flush the response stream, causing all buffered data to be written to the client.

Returns

this response object for chaining

AsyncResponse.prototype.start (status, headers)

Set the HTTP status code and headers of the response. This method must only be called once.

Parameters

Number	**status**	the HTTP status code
Object	**headers**	the headers

Returns

this response object for chaining

AsyncResponse.prototype.write (data, [encoding])

Write a chunk of data to the response stream.

Parameters

String|Binary	**data**	a binary or string
String	**[encoding]**	the encoding to use

Returns

this response object for chaining

---

handleRequest (moduleId, functionObj, request)

Handle a JSGI request.

Parameters

String	**moduleId**	the module id. Ignored if functionObj is already a function.
Function	**functionObj**	the function, either as function

		object or function name to be imported from the module moduleId.
Object	**request**	the JSGI request object

Returns

Object	the JSGI response object

Module ringo/jsgi/response

This module provides response helper functions for composing JSGI response objects. For more flexibility the JsgiResponse is chainable.

Functions

- addHeaders (headers)
- bad ()
- created ()
- error ()
- forbidden ()
- gone ()
- html (html...)
- json (object)
- jsonp (callback, object)
- notFound ()
- notModified (functionName)
- ok ()
- redirect (location)
- setCharset (charsetName)
- setStatus (code)
- static (resource, contentType)
- text (text...)
- unauthorized ()
- unavailable ()
- xml (xml)

Class JsgiResponse

Instance Methods

- addHeaders (headers)
- bad ()
- created ()
- error ()
- forbidden ()

gone ()

html (html...)

json (object)

jsonp (callback, object)

notFound ()

notModified ()

ok ()

redirect (location)

setCharset (charsetName)

setStatus (code)

text (text...)

unauthorized ()

unavailable ()

xml (xml)

Instance Properties

body

headers

status

JsgiResponse (base)

A wrapper around a JSGI response object. JsgiResponse is chainable.

Example

```
// Using the constructor
var {JsgiResponse} = require('ringo/jsgi/response');
return (new JsgiResponse()).text('Hello World!').setCharset('ISO-8859-1');

// Using a static helper
var response = require('ringo/jsgi/response');
return response.json({'foo': 'bar'}).error();
```

Parameters

Object	**base**	a base object for the new JSGI response with the initial status, headers and body properties.

JsgiResponse.prototype.addHeaders (headers)

Merge the given object into the headers of the JSGI response.

Parameters

Object	**headers**	new header fields to merge with the current ones.

Returns

JSGI response with the new headers

JsgiResponse.prototype.bad ()

Sets the HTTP status to 400.

Returns

a JSGI response object to send back

JsgiResponse.prototype.body

JsgiResponse.prototype.created ()

Sets the HTTP status to 201.

Returns

a JSGI response object to send back

JsgiResponse.prototype.error ()

Sets the HTTP status to 500.

Returns

a JSGI response object to send back

JsgiResponse.prototype.forbidden ()

Sets the HTTP status to 403.

Returns

a JSGI response object to send back

JsgiResponse.prototype.gone ()

Sets the HTTP status to 410.

Returns

a JSGI response object to send back

JsgiResponse.prototype.headers

JsgiResponse.prototype.html (html...)

Set the JSGI response content-type to 'text/html' with the string as response body.

Parameters

String	**html...**	a variable number of strings to send as response body

Returns

JSGI response with content-type 'text/html'

JsgiResponse.prototype.json (object)

Create a JSGI response with content-type 'application/json' with the JSON representation of the given object as response body.

Parameters

Object	**object**	the object whose JSON representation to return

Returns

> JSGI response with content-type 'application/json'

JsgiResponse.prototype.jsonp (callback, object)

Create a JSGI response with content-type 'application/javascript' with the JSONP representation of the given object as response body wrapped by the callback name.

Parameters

String	**callback**	the callback function name for a JSONP request
Object	**object**	the object whose JSON representation to return

Returns

> JSGI response with content-type 'application/javascript'

JsgiResponse.prototype.notFound ()

Sets the HTTP status to 404.

Returns

> a JSGI response object to send back

JsgiResponse.prototype.notModified ()

Create a response with HTTP status code 304 that indicates the document has not been modified

Returns

> a JSGI response object to send back

JsgiResponse.prototype.ok ()

Sets the HTTP status to 200.

Returns

a JSGI response object to send back

JsgiResponse.prototype.redirect (location)

Create a response with HTTP status code 303 that redirects the client to a new location.

Parameters

String	**location**	the new location

Returns

a JSGI response object to send back

JsgiResponse.prototype.setCharset (charsetName)

Set the character encoding used for text responses.

Parameters

String	**charsetName**	the encoding to use.

Returns

JSGI response with the given charset

JsgiResponse.prototype.setStatus (code)

Set the JSGI response status. This does not commit the request and continues the JsgiReponse chain.

Parameters

Number	**code**	the status code to use

Returns

JSGI response with the new status code

JsgiResponse.prototype.status

JsgiResponse.prototype.text (text...)

Set the JSGI response content-type to 'text/plain' with the string as response body.

Parameters

String	**text...**	a variable number of strings to send as response body

Returns

JSGI response with content-type 'text/plain'

JsgiResponse.prototype.unauthorized ()

Sets the HTTP status to 401.

Returns

a JSGI response object to send back

JsgiResponse.prototype.unavailable ()

Sets the HTTP status to 503.

Returns

a JSGI response object to send back

JsgiResponse.prototype.xml (xml)

Create a JSGI response with content-type 'application/xml' with the given XML as response body.

Parameters

XML|String	**xml**	an XML document

Returns

JSGI response with content-type 'application/xml'

---

addHeaders (headers)

Merge the given object into the headers of the JSGI response.

Parameters

Object	**headers**	new header fields to merge with the current ones.

Returns

JSGI response with the new headers

---

bad ()

Sets the HTTP status to 400.

Returns

a JSGI response object to send back

---

created ()

Sets the HTTP status to 201.

Returns

a JSGI response object to send back

---

error ()

Sets the HTTP status to 500.

Returns

a JSGI response object to send back

---

forbidden ()

Sets the HTTP status to 403.

Returns

a JSGI response object to send back

---

gone ()

Sets the HTTP status to 410.

Returns

a JSGI response object to send back

---

html (html...)

Set the JSGI response content-type to 'text/html' with the string as response body.

Parameters

String	**html...**	a variable number of strings to send as response body

Returns

JSGI response with content-type 'text/html'

---

json (object)

Create a JSGI response with content-type 'application/json' with the JSON representation of the given object as response body.

Parameters

Object	**object**	the object whose JSON representation to

return

Returns

JSGI response with content-type 'application/json'

---

jsonp (callback, object)

Create a JSGI response with content-type 'application/javascript' with the JSONP representation of the given object as response body wrapped by the callback name.

Parameters

String	**callback**	the callback function name for a JSONP request
Object	**object**	the object whose JSON representation to return

Returns

JSGI response with content-type 'application/javascript'

---

notFound ()

Sets the HTTP status to 404.

Returns

a JSGI response object to send back

---

notModified (functionName)

Create a response with HTTP status code 304 that indicates the document has not been modified

Parameters

functionName

Returns

a JSGI response object to send back

ok ()

Sets the HTTP status to 200.

Returns

a JSGI response object to send back

redirect (location)

Create a response with HTTP status code 303 that redirects the client to a new location.

Parameters

String	**location**	the new location

Returns

a JSGI response object to send back

setCharset (charsetName)

Set the character encoding used for text responses.

Parameters

String	**charsetName**	the encoding to use.

Returns

JSGI response with the given charset

setStatus (code)

Static helper to create a JsgiResponse with the given status code.

Parameters

Number	**code**	the status code to use

Returns

JSGI response with the new status code

---

static (resource, contentType)

A response representing a static resource.

Parameters

String|Resource	**resource**	the resource to serve
String	**contentType**	optional MIME type. If not defined, the MIME type is detected from the file name extension.

---

text (text...)

Set the JSGI response content-type to 'text/plain' with the string as response body.

Parameters

String	**text...**	a variable number of strings to send as response body

Returns

JSGI response with content-type 'text/plain'

---

unauthorized ()

Sets the HTTP status to 401.

Returns

a JSGI response object to send back

---

unavailable ()

Sets the HTTP status to 503.

Returns

a JSGI response object to send back

---

xml (xml)

Create a JSGI response with content-type 'application/xml' with the given XML as response body.

Parameters

XML|String	**xml**	an XML document

Returns

JSGI response with content-type 'application/xml'

Module ringo/utils/arrays

Provides utility functions for working with JavaScript Arrays.

Functions

contains (array, val)

Check if an array passed as argument contains a specific value (start from end of array).

Parameters

Array	**array**	the array
Object	**val**	the value to check

Returns

boolean	true if the value is contained

intersection (array1,...)

Retrieve the intersection set of a bunch of arrays.

Parameters

Array	**array1,...**	the arrays to intersect

Returns

Array	the intersection set

max (array)

Parameters

Array	**array**	the array

Returns

the maximal element in an array obtained by calling Math.max().

min (array)

Parameters

Array	**array**	the array

Returns

the minimal element in an array obtained by calling Math.min().

partition (fn)

Parameters

fn

peek (array)

Return the last element of the array. This is like pop(), but without modifying the array.

Parameters

Array	**array**	the array

Returns

Object	the last element of the array, or undefined if the array is empty.

remove (array, val)

Remove the first occurrence of the argument value from the array. This method mutates and returns the array on which it is called and does not create a new array instance.

Parameters

Array	**array**	the array
Object	**val**	the value to remove

Returns

Array	the array

---

union (array1,...)

Retrieve the union set of a bunch of arrays.

Parameters

Array	**array1,...**	the arrays to unify

Returns

Array	the union set

Module ringo/utils/dates

Adds useful functions for working with JavaScript Date objects.

Functions

- add (date, delta, unit)
- after (a, b)
- before (a, b)
- checkDate (fullYear, month, day)
- compare (a, b)
- dayOfYear (date)
- daysInFebruary (date)
- daysInMonth (date)
- daysInYear (date)
- diff (a, b, unit)
- firstDayOfWeek (locale)
- format (the, format, locale, timezone)
- fromUTCDate (year, month, date, hour, minute, second)
- inPeriod (date, periodStart, periodEnd, periodStartOpen, periodEndOpen)
- isLeapYear (date)
- overlapping (aStart, aEnd, bStart, bEnd)
- parse (str)
- quarterInFiscalYear (date, fiscalYearStart)
- quarterInYear (date)
- resetDate (date)
- resetTime (date)
- secondOfDay (date)
- toISOString (date, withTime, withTimeZone, withSeconds, withMilliseconds)
- weekOfMonth (date, locale)
- weekOfYear (date, locale)
- yearInCentury (date)

add (date, delta, unit)

Adds delta to the given field or reduces it, if delta is negative. If larger fields are effected, they will be changed accordingly.

Parameters

Date	**date**	base date to add or remove time from.
Number	**delta**	amount of time to add (positive delta) or remove

		(negative delta).
String	**unit**	(optional) field to change. Possible values: year, quarter, month, week, day (default), hour (24-hour clock), minute, second, millisecond.

Returns

Date	date with the calculated date and time

See

http://download.oracle.com/javase/1.5.0/docs/api/java/util/GregorianCalendar.html#add(int,%20int)

---

after (a, b)

Checks if date a is after date b. This is equals to compare(a, b) > 0

Parameters

Date	**a**	first date
Date	**b**	second date

Returns

Boolean true if a is after b, false if not.

---

before (a, b)

Checks if date a is before date b. This is equals to compareTo(a, b) < 0

Parameters

Date	**a**	first date
Date	**b**	second date

Returns

Boolean true if a is before b, false if not.

---

checkDate (fullYear, month, day)

Checks if the date is a valid date. Example: 2007 is no leap year, so checkDate(2007, 1, 29) returns false.

Parameters

Number	**fullYear**	
Number	**month**	between 0 and 11
Number	**day**	between 1 and 31

Returns

Boolean	true, if the date is valid, false if not.

compare (a, b)

Compares the time values of a and b.

Parameters

Date	**a**	first date
Date	**b**	second date

Returns

Number -1 if a is before b, 0 if equals and 1 if a is after b.

See

http://download.oracle.com/javase/1.5.0/docs/api/java/util/Calendar.html#compareTo(java.util.Calendar)

dayOfYear (date)

Gets the day of the year for the given date.

Parameters

Date	**date**	calculate the day of the year.

Returns

Number day of the year

daysInFebruary (date)

Gets the number of the days in february.

Parameters

Date	**date**	of year to find the number of days in february.

Returns

Number days in the february, 28 or 29, if it's a leap year.

daysInMonth (date)

Gets the number of the days in the month.

Parameters

Date	**date**	to find the maximum number of days.

Returns

Number days in the month, between 28 and 31.

daysInYear (date)

Gets the number of the days in the year.

Parameters

Date	**date**	to find the maximum number of days.

Returns

Number days in the year, 365 or 366, if it's a leap year.

---

diff (a, b, unit)

Get the difference between two dates, specified by the unit of time.

Parameters

Date	**a**	first date
Date	**b**	second date
String	**unit**	(optional) of time to return. Possible values: `year`, `quarter`, `month`, `week`, `day` (default), `hour`, `minute`, `second`, `millisecond` and `mixed` (returns an object)

Returns

Number\|Object<{days, hours, minutes, seconds, milliseconds}> difference between the given dates in the specified unit of time.

---

firstDayOfWeek (locale)

Gets the first day of the week.

Parameters

String|java.util.Locale	**locale**	(optional) the locale as java Locale object or lowercase two-letter ISO-639 code (e.g. "en")

Returns

Number the first day of the week; 1 = Sunday, 2 = Monday.

See

http://download.oracle.com/javase/1.5.0/docs/api/constant-values.html#java.util.Calendar.SUNDAY

---

format (the, format, locale, timezone)

Format a Date to a string. For details on the format pattern, see http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html

Parameters

Date	**the**	Date to format

String	**format**	the format pattern
String|java.util.Locale	**locale**	(optional) the locale as java Locale object or lowercase two-letter ISO-639 code (e.g. "en")
String|java.util.TimeZone	**timezone**	(optional) the timezone as java TimeZone object or an abbreviation such as "PST", a full name such as "America/Los_Angeles", or a custom ID such as "GMT-8:00". If the id is not provided, the default timezone is used. If the timezone id is provided but cannot be understood, the "GMT" timezone is used.

Returns

String the formatted Date

See

http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html

---

fromUTCDate (year, month, date, hour, minute, second)

Create new Date from UTC timestamp.

Parameters

Number	**year**
Number	**month**
Number	**date**
Number	**hour**
Number	**minute**
Number	**second**

Returns

Date

---

inPeriod (date, periodStart, periodEnd, periodStartOpen, periodEndOpen)

Look if the date is in the period, using *periodStart <= date <= periodEnd*.

Parameters

Date	**date**	to check, if it's in the period
Date	**periodStart**	the period's start
Date	**periodEnd**	the period's end
Boolean	**periodStartOpen**	start point is open - default false.
Boolean	**periodEndOpen**	end point is open - default false.

Returns

Boolean true if the date is in the period, false if not.

isLeapYear (date)

Checks if the date's year is a leap year.

Parameters

Date	**date**	to check year

Returns

Boolean true if the year is a leap year, false if not.

overlapping (aStart, aEnd, bStart, bEnd)

Look if two periods are overlapping each other.

Parameters

Date	**aStart**	first period's start
Date	**aEnd**	first period's end
Date	**bStart**	second period's start
Date	**bEnd**	second period's end

Returns

Boolean true if the periods are overlapping at some point, false if not.

parse (str)

Parse a string representing a date. For details on the string format, see http://tools.ietf.org/html/rfc3339. Examples include "2010", "2010-08-06", "2010-08-06T22:04:30Z", "2010-08-06T16:04:30-06".

Parameters

String	**str**	The date string. This follows the format specified for timestamps on the internet described in RFC 3339.

Returns

Date	a date representing the given string

See

http://tools.ietf.org/html/rfc3339
http://www.w3.org/TR/NOTE-datetime

quarterInFiscalYear (date, fiscalYearStart)

Gets the quarter in the fiscal year.

Parameters

Date	**date**	to calculate the quarter for.

Date	**fiscalYearStart**	first day in the fiscal year, default is the start of the current year

Returns

Number quarter of the year, between 1 and 4.

---

quarterInYear (date)

Gets the quarter in the year.

Parameters

Date	**date**	to calculate the quarter for.

Returns

Number quarter of the year, between 1 and 4.

---

resetDate (date)

Drops the date values, keeping only hours, minutes, seconds and milliseconds.

Parameters

Date	**date**	to reset

Returns

Date date with the original time values and 1970-01-01 as date.

---

resetTime (date)

Resets the time values to 0, keeping only year, month and day.

Parameters

Date	**date**	to reset

Returns

Date date without any time values

---

secondOfDay (date)

Gets the second of the day for the given date.

Parameters

Date	**date**	calculate the second of the day.

Returns

Number second of the day

---

toISOString (date, withTime, withTimeZone, withSeconds, withMilliseconds)

Create a ISO 8601 compatible string from the date. Note: This is quite similar to Date.toISOString(), which only returns an UTC-based string without the local timezone. If you don't need timezones, Date.toISOString() will be the better choice.

Parameters

Date	**date**	to format
Boolean	**withTime**	if true, the string will contain the time, if false only the date. Default is true.
Boolean	**withTimeZone**	if true, the string will be in local time, if false it's in UTC. Default is true.
Boolean	**withSeconds**	if true, the string will contain also the seconds of the date. Default true.
Boolean	**withMilliseconds**	if true, the string will contain also the milliseconds of the date. Default false.

Returns

String date as ISO 8601 string.

---

weekOfMonth (date, locale)

Gets the week of the month for the given date.

Parameters

Date	**date**	calculate the week of the month.
String|java.util.Locale	**locale**	(optional) the locale as java Locale object or lowercase two-letter ISO-639 code (e.g. "en")

Returns

Number week of the month

---

weekOfYear (date, locale)

Gets the week of the year for the given date.

Parameters

Date	**date**	calculate the week of the year.
String|java.util.Locale	**locale**	(optional) the locale as java Locale object or lowercase two-letter ISO-639 code (e.g. "en")

Returns

Number week of the
year

---

yearInCentury (date)

Gets the year of the century for the given date. *Examples:* 1900 returns 0, 2010 returns 10.

Parameters

Date	**date**	calculate the year of the century.

Returns

Number second of the day

Module ringo/utils/files

A collection of file related utilities not covered by the fs module.

Functions

createTempFile (prefix, suffix, directory)

isHidden (file)

resolveId (parent, child)

resolveUri (arbitrary)

Properties

roots

separator

createTempFile (prefix, suffix, directory)

Create a new empty temporary file in the default directory for temporary files.

Parameters

String	**prefix**	the prefix of the temporary file; must be at least three characters long
String	**suffix**	the suffix of the temporary file; may be undefined or null
String	**directory**	optional directory in which to create the file. Pass undefined to use the system's default location for temporary files

Returns

File	the temporary file

---

isHidden (file)

Tests whether the file represented by this File object is a hidden file. What constitutes a hidden file may depend on the platform we are running on.

Parameters

String	**file**

Returns

Boolean	Boolean true if this File object is hidden

---

resolveId (parent, child)

Resolve path fragment child relative to parent but only if child is a a relative path according to the CommonJS Modules spec, i.e. starts with "." or "..". Otherwise, the child path is returned unchanged.

Parameters

String	**parent**	the parent path
String	**child**	the child path

---

resolveUri (arbitrary)

Resolve an arbitrary number of path elements relative to each other. This is an adapted version of the file module's resolve function that always and strictly uses forward slashes as file separators. This makes it usable for resolving URI paths as well as module ids and resource paths. Originally adapted for helma/file from narwhal's file module.

Parameters

...	**arbitrary**	number of path elements

---

roots

An Array containing the system's file system roots. On UNIX platforms this contains a single "/" directory, while on Windows platforms this contains an element for each mounted drive.

---

separator

The system-dependent file system separator character.

Module ringo/utils/http

Provides utility functions to work with HTTP requests and responses.

Functions

- BufferFactory (data, encoding)
- TempFileFactory (data, encoding)
- getMimeParameter (headerValue, paramName)
- isFileUpload (contentType)
- isUrlEncoded (contentType)
- mergeParameter (params, name, value)
- parseFileUpload (request, params, encoding, streamFactory)
- parseParameters (input, params, encoding)
- setCookie (key, value, days, options)
- urlEncode (object)

Class Headers

Instance Methods

- add (name, value)
- contains (name)
- get (name)
- set (name, value)
- toString ()
- unset (name)

Class ResponseFilter

Instance Methods

- forEach (fn)

BufferFactory (data, encoding)

A stream factory that stores file upload in a memory buffer. This function is not meant to be called directly but to be passed as streamFactory argument to parseFileUpload().

The buffer is stored in the value property of the parameter's data object.

Parameters

Object	**data**
String	**encoding**

Headers (headers)

Returns an object for use as a HTTP header collection. The returned object provides methods for setting, getting, and deleting its properties in a case-insensitive and case-preserving way.

This function can be used as mixin for an existing JavaScript object or as a constructor.

Parameters

Object	**headers**	an existing JS object. If undefined, a new object is created

Headers.prototype.add (name, value)

Add a header with the given name and value.

Parameters

String	**name**	the header name
String	**value**	the header value

Headers.prototype.contains (name)

Queries whether a header with the given name is set

Parameters

String	**name**	the header name

Returns

Boolean	true if a header with this name is set

Headers.prototype.get (name)

Get the value of the header with the given name

Parameters

String	**name**	the header name

Returns

the header value

Headers.prototype.set (name, value)

Set the header with the given name to the given value.

Parameters

String	**name**	the header name
String	**value**	the header value

Headers.prototype.toString ()

Returns a string representation of the headers in MIME format.

Returns

String	a string representation of the headers

Headers.prototype.unset (name)

Unsets any cookies with the given name

Parameters

String	**name**	the header name

ResponseFilter (body, filter)

A utility class for implementing JSGI response filters. Each part of the response is first passed to the filter function. If the filter function returns a value, that value is passed on to the JSGI response stream.

Parameters

Object	**body**	a JSGI response body
Function	**filter**	a filter function

ResponseFilter.prototype.forEach (fn)

forEach function called by the JSGI connector.

Parameters

Function	**fn**	the response handler callback function

TempFileFactory (data, encoding)

A stream factory that stores file uploads in temporary files. This function is not meant to be called directly but to be passed as streamFactory argument to parseFileUpload().

The name of the temporary file is stored in the tempfile property of the parameter's data object.

Parameters

Object	**data**
String	**encoding**

---

getMimeParameter (headerValue, paramName)

Get a parameter from a MIME header value. For example, calling this function with "Content-Type: text/plain; charset=UTF-8" and "charset" will return "UTF-8".

Parameters

String	**headerValue**	a header value
String	**paramName**	a MIME parameter name

isFileUpload (contentType)

Find out whether the content type denotes a format this module can parse.

Parameters

String	**contentType**	a HTTP request Content-Type header

Returns

true if the content type can be parsed as form data by this module

isUrlEncoded (contentType)

Find out whether the content type denotes a format this module can parse.

Parameters

String	**contentType**	a HTTP request Content-Type header

Returns

true if the content type can be parsed as form data by this module

mergeParameter (params, name, value)

Adds a value to a parameter object using a square bracket property syntax. For example, parameter foo[bar][][baz]=hello will result in object structure {foo: {bar: [{baz : "hello"}]}}.

Parameters

Object	**params**	the top level parameter object
String	**name**	the parameter name
String	**value**	the parameter value

parseFileUpload (request, params, encoding, streamFactory)

Parses a multipart MIME input stream. Parses a multipart MIME input stream.

Parameters

Object	**request**	the JSGI request object
Object	**params**	the parameter object to parse into. If not defined a new object is created and returned.
string	**encoding**	optional encoding to apply to non-file parameters. Defaults to "UTF-8".
function	**streamFactory**	factory function to create streams for mime parts

Returns

Object	the parsed parameter object

---

parseParameters (input, params, encoding)

Parse a string or binary object representing a query string or post data into a JavaScript object structure using the specified encoding.

Parameters

Binary|String	**input**	a Binary object or string containing the URL-encoded parameters
Object	**params**	optional parameter object to parse into. If undefined a new object is created and returned.
String	**encoding**	a valid encoding name, defaults to UTF-8

Returns

the parsed parameter object

---

setCookie (key, value, days, options)

Creates value for the Set-Cookie header for creating a cookie with the given name, value, and attributes.

All arguments except for key and value are optional. The days argument specifies the number of days until the cookie expires. To delete a cookie immediately, set the days argument to 0. If days is undefined or negative, the cookie is set for the current browser session.

Example

```
setCookie("username", "michi");
setCookie("password", "strenggeheim", 10,
{path: "/mypath", domain: ".mydomain.org"});
```

Parameters

String	**key**	the cookie name
String	**value**	the cookie value
Number	**days**	optional the number of days to keep the cookie. If this is undefined or -1, the cookie is set for the current session. If this is 0, the cookie will be deleted immediately.
Object	**options**	optional options argument which may contain the following properties: • path - the path on which to set the cookie (defaults to /) • domain - the domain on which to set the cookie (defaults to current domain) • secure - to only use this cookie for secure connections • httpOnly - to make the cookie inaccessible to client side scripts

Returns

String	the Set-Cookie header value

---

urlEncode (object)

Encode an object's properties into an URL encoded string.

Parameters

Object	**object**	an object

Returns

String	a string containing the URL encoded properties of the object

Module ringo/utils/numbers

Provides utility functions for working with JavaScript numbers.

Functions

- format (number, fmt, locale)
- times (num, fun)

format (number, fmt, locale)

Format number using java.text.DecimalFormat.

Parameters

Number	**number**	the number
String	**fmt**	the format to apply
String	**locale**	optional locale

Returns

String	the number formatted as string

---

times (num, fun)

Invoke a function num times, passing 0 .. (this - 1) as argument.

Parameters

Number	**num**	the number
Function	**fun**	the function to call

Module ringo/utils/objects

Adds utility functions for working with JavaScript Objects

Functions

- clone (object, cloned, recursive)
- merge (obj...)

clone (object, cloned, recursive)

copy the properties of an object into a new object

Parameters

Object	**object**	the object to clone
Object	**cloned**	optional clone object
boolean	**recursive**	pass true to create a deep clone. Otherwise a shallow clone is created.

Returns

Object	the clone object

merge (obj...)

Creates a new object as the as the keywise union of the provided objects. Whenever a key exists in a later object that already existed in an earlier object, the according value of the earlier object takes precedence.

Parameters

Object	**obj...**	The objects to merge

Ringo Modules

- Modules
- Overview

Module ringo/utils/strings

Adds useful methods to the JavaScript String type.

Functions

- Sorter (field, order)
- b16decode (str, encoding)
- b16encode (str, encoding)
- b64decode (string, encoding)
- b64encode (string, encoding)
- capitalize (the, amount)
- compose (one)
- contains (string, substring, fromIndex)
- count (string, pattern)
- digest (string, algorithm)
- endsWith (string, substring)
- entitize (string)
- escapeHtml (string)
- escapeRegExp (str)
- format (format)
- getCommonPrefix (str1, str2)
- group (string, interval, string, ignoreWhiteSpace)
- isAlpha (string)
- isAlphanumeric (string)
- isDateFormat (string)
- isEmail (string)
- isFileName (string)
- isFloat (string)
- isHexColor (string)
- isInt (string)
- isLowerCase (string)
- isNumeric (string)
- isUpperCase (string)
- isUrl (string)
- join (the, the, the)

- pad (string, fill, length, mode)
- random (len, mode)
- repeat (string, num)
- startsWith (string, substring)
- stripTags (string)
- titleize (string)
- toAlphanumeric (string)
- toCamelCase (string)
- toDashes (string)
- toDate (string, format, timezone)
- toFileName (string)
- toHexColor (string)
- toUnderscores (string)
- unwrap (flag, replacement)
- y64decode (string, encoding)
- y64encode (string, encoding)

Sorter (field, order)

factory to create functions for sorting objects in an array

Parameters

String	**field**	name of the field each object is compared with
Number	**order**	(ascending or descending)

Returns

Function	ready for use in Array.prototype.sort

---

b16decode (str, encoding)

Decodes a Base16 encoded string to a string or byte array.

Parameters

String	**str**	the Base16 encoded string
String	**encoding**	the encoding to use for the return value. Defaults to 'utf8'. Use 'raw' to get a ByteArray instead of a string.

Returns

the decoded string or ByteArray

---

b16encode (str, encoding)

Encode a string or binary to a Base16 encoded string

Parameters

String|Binary	**str**	a string or binary
String	**encoding**	optional encoding to use if first argument is a string. Defaults to 'utf8'.

Returns

the Base16 encoded string

---

b64decode (string, encoding)

Decodes a Base64 encoded string to a string or byte array.

Parameters

String	**string**	the Base64 encoded string
String	**encoding**	the encoding to use for the return value. Defaults to 'utf8'. Use 'raw' to get a ByteArray instead of a string.

Returns

the decoded string or ByteArray

---

b64encode (string, encoding)

Encode a string or binary to a Base64 encoded string

Parameters

String|Binary	**string**	a string or binary
String	**encoding**	optional encoding to use if first argument is a string. Defaults to 'utf8'.

Returns

the Base64 encoded string

capitalize (the, amount)

transforms the first n characters of a string to uppercase

Parameters

String	**the**	string to capitalize
Number	**amount**	of characters to transform

Returns

String	the resulting string

compose (one)

create a string from a bunch of substrings

Parameters

String	**one**	or more strings as arguments

Returns

String	the resulting string

contains (string, substring, fromIndex)

Returns true if string contains substring.

Parameters

String	**string**	the string to search in
String	**substring**	the string to search for
Number	**fromIndex**	optional index to start searching

Returns

Boolean	true if substring is contained in this string

count (string, pattern)

returns the amount of occurences of one string in another

Parameters

String	**string**
String	**pattern**

digest (string, algorithm)

function calculates a message digest of a string. If no argument is passed, the MD5 algorithm is used.

Parameters

String	**string**	the string to digest
String	**algorithm**	the name of the algorithm to use

Returns

String	base16-encoded message digest of the string

endsWith (string, substring)

Returns true if string ends with the given substring

Parameters

String	**string**	the string to search in
String	**substring**	pattern to search for

Returns

Boolean true in case it matches the end of the string, false otherwise

entitize (string)

translates all characters of a string into HTML entitie

Parameters

String	**string**	the string

Returns

String	translated result

escapeHtml (string)

Escape the string to make it safe for use within an HTML document.

Parameters

String	**string**	the string to escape

Returns

String	the escaped string

---

escapeRegExp (str)

Accepts a string; returns the string with regex metacharacters escaped. the returned string can safely be used within a regex to match a literal string. escaped characters are [,], {, }, (,), -, *, +, ?, ., , ^, $, |, #, [comma], and whitespace.

Parameters

String	**str**	the string to escape

Returns

String	the escaped string

---

format (format)

A simple string formatter. If the first argument is a format string containing a number of curly bracket pairs {} as placeholders, the same number of following arguments will be used to replace the curly bracket pairs in the format string. If the first argument is not a string or does not contain any curly brackets, the arguments are simply concatenated to a string and returned.

Parameters

String	**format**	string, followed by a variable number of values

Returns

String	the formatted string

---

getCommonPrefix (str1, str2)

Get the longest common segment that two strings have in common, starting at the beginning of the string

Parameters

String	**str1**	a string
String	**str2**	another string

Returns

String	the longest common segment

group (string, interval, string, ignoreWhiteSpace)

function inserts a string every number of characters

Parameters

String	**string**	
Number	**interval**	number of characters after which insertion should take place
String	**string**	to be inserted
Boolean	**ignoreWhiteSpace**	definitely insert at each interval position

Returns

String resulting string

isAlpha (string)

function returns true if the string contains only characters a-z

Parameters

string

Returns

Boolean true in case string is alpha, false otherwise

isAlphanumeric (string)

function returns true if the string contains only a-z and 0-9 (case insensitive!)

Parameters

string

Returns

Boolean true in case string is alpha, false otherwise

---

isDateFormat (string)

checks if a date format pattern is correct

Parameters

String	**string**	the string

Returns

Boolean true if the pattern is correct

---

isEmail (string)

returns true if the string looks like an e-mail

Parameters

String	**string**

---

isFileName (string)

function checks if the string passed contains any characters that are forbidden in image- or filenames

Parameters

String	**string**	the string

Returns

Boolean

---

isFloat (string)

returns true if the string is a floating point literal

Parameters

String	**string**	

---

isHexColor (string)

function checks a string for a valid color value in hexadecimal format. it may also contain # as first character

Parameters

String	**string**	the string

Returns

Boolean false, if string length (without #) > 6 or < 6 or contains any character which is not a valid hex value

---

isInt (string)

returns true if the string is an integer literal

Parameters

String	**string**

---

isLowerCase (string)

returns true if the string is lowercase

Parameters

String	**string**

---

isNumeric (string)

function returns true if the string contains only 0-9

Parameters

string

Returns

Boolean true in case string is numeric, false otherwise

---

isUpperCase (string)

returns true if the string is uppercase

Parameters

String	**string**	

---

isUrl (string)

function checks if the string is an URL validating. Only HTTP, HTTPS and FTP are allowed protocols.

Parameters

String	**string**	the string

Returns

Boolean

---

join (the, the, the)

append one string onto another and add some "glue" if none of the strings is empty or null.

Parameters

String	**the**	first string
String	**the**	string to be appended onto the first one
String	**the**	"glue" to be inserted between both strings

Returns

String	the resulting string

---

pad (string, fill, length, mode)

fills a string with another string up to a desired length

Parameters

String	**string**	the string
String	**fill**	the filling string
Number	**length**	the desired length of the resulting string
Number	**mode**	the direction which the string will be

padded in: a negative number means left, 0 means both, a positive number means right

Returns

String the resulting string

---

random (len, mode)

creates a random string (numbers and chars)

Parameters

Number	**len**	length of key
Number	**mode**	determines which letters to use. null or 0 = all letters; 1 = skip 0, 1, l and o which can easily be mixed with numbers; 2 = use numbers only

Returns

random string

---

repeat (string, num)

function repeats a string passed as argument

Parameters

String	**string**	the string
Number	**num**	amount of repetitions

Returns

String	resulting string

---

startsWith (string, substring)

Returns true if string starts with the given substring

Parameters

String	**string**	the string to search in
String	**substring**	pattern to search for

Returns

Boolean	true in case it matches the beginning of the string, false otherwise

---

stripTags (string)

Remove all potential HTML/XML tags from this string

Parameters

String	**string**	the string

Returns

String	the processed string

---

titleize (string)

transforms the first n characters of each word in a string to uppercase

Parameters

String	**string**	the string

Returns

String	the resulting string

---

toAlphanumeric (string)

function cleans a string by throwing away all non-alphanumeric characters

Parameters

	string

Returns

	cleaned string

---

toCamelCase (string)

Transforms string from space, dash, or underscore notation to camel-case.

Parameters

String	**string**	a string

Returns

String	the resulting string

---

toDashes (string)

Transforms string from camel-case to dash notation.

Parameters

String	**string**	a string

Returns

String	the resulting string

---

toDate (string, format, timezone)

parse a timestamp into a date object.

Parameters

String	**string**	the string
String	**format**	date format to be applied
Object	**timezone**	Java TimeZone Object (optional)

Returns

Object	the resulting date

---

toFileName (string)

function cleans the string passed as argument from any characters that are forbidden or shouldn't be used in filenames

Parameters

String	**string**	the string

Returns

Boolean

---

toHexColor (string)

converts a string into a hexadecimal color representation (e.g. "ffcc33"). also knows how to convert a color string like "rgb (255, 204, 51)".

Parameters

String	**string**	the string

Returns

String the resulting hex color (w/o "#")

---

toUnderscores (string)

Transforms string from camel-case to underscore notation.

Parameters

String	**string**	a string

Returns

String	the resulting string

---

unwrap (flag, replacement)

replace all linebreaks and optionally all w/br tags

Parameters

Boolean	**flag**	indicating if html tags should be replaced
String	**replacement**	for the linebreaks / html tags

Returns

String the unwrapped string

---

y64decode (string, encoding)

Decodes a Y64 encoded string to a string or byte array.

Parameters

String	**string**	the Y64 encoded string

String	**encoding**	the encoding to use for the return value. Defaults to 'utf8'. Use 'raw' to get a ByteArray instead of a string.

Returns

the decoded string or ByteArray

---

y64encode (string, encoding)

Encode a string or binary to a Y64 encoded string. Y64 is an URL-safe Base64 encoding and prevents any URL escaping. It replaces the plus (+), slash (/) and equals (=) with dot (.), underscore (_) and dash (-).

Parameters

String|Binary	**string**	a string or binary
String	**encoding**	optional encoding to use if first argument is a string. Defaults to 'utf8'.

Returns

the Y64 encoded string

See

Detailed Y64 description