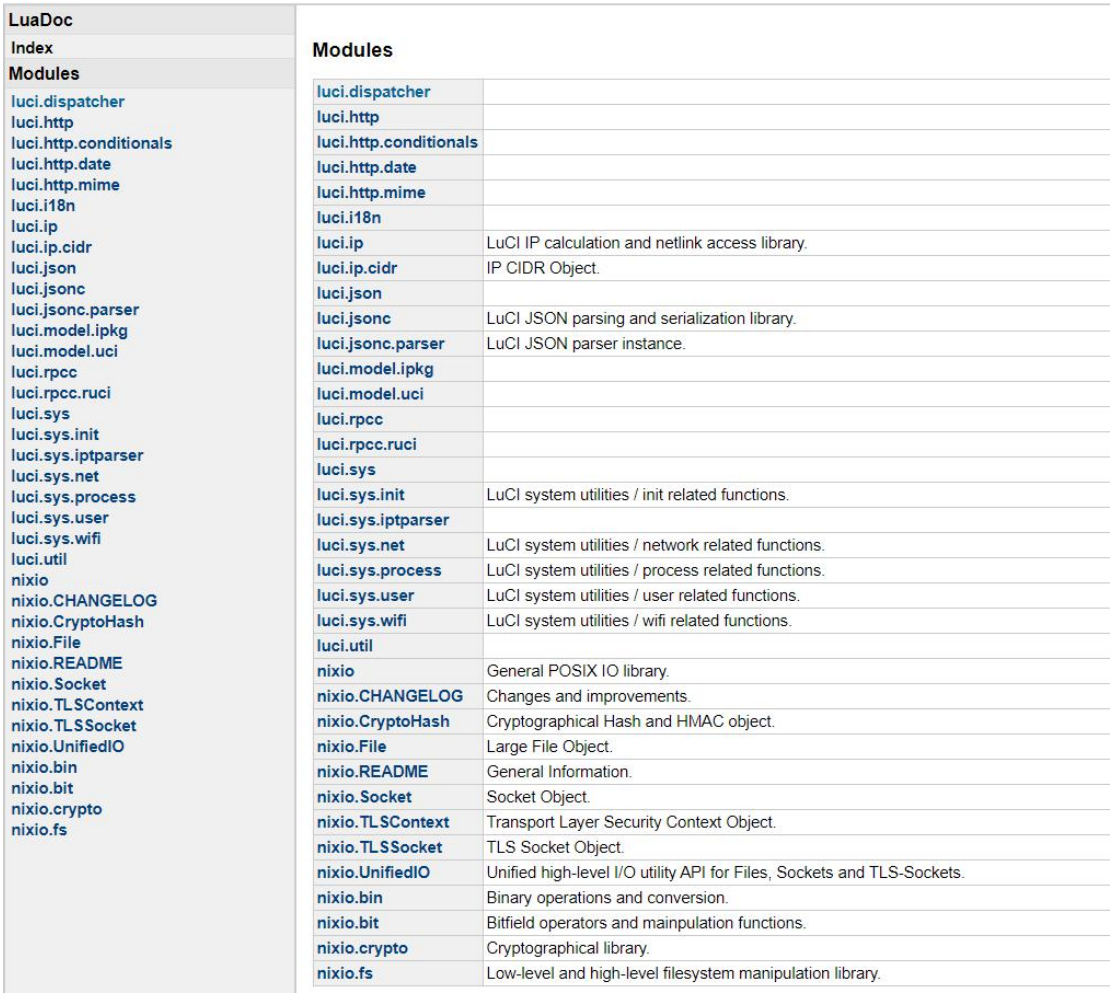
LUCI-APIs-Reference

OverView



LuaDoc

Index

Modules

luci.dispatcher
luci.http
luci.http.conditionals
luci.http.date
luci.http.mime
luci.i18n
luci.ip
luci.ip.cidr
luci.json
luci.jsonc
luci.jsonc.parser
luci.model.ipkg
luci.model.uci
luci.rpcc
luci.rpcc.ruci
luci.sys
luci.sys.init
luci.sys.iptparser
luci.sys.net
luci.sys.process
luci.sys.user
luci.sys.wifi
luci.util
nixio
nixio.CHANGELOG
nixio.CryptoHash
nixio.File
nixio.README
nixio.Socket
nixio.TLSContext
nixio.TLSSocket
nixio.UnifiedIO
nixio.bin
nixio.bit
nixio.crypto
nixio.fs

Modules

luci.dispatcher	
luci.http	
luci.http.conditionals	
luci.http.date	
luci.http.mime	
luci.i18n	
luci.ip	LuCI IP calculation and netlink access library.
luci.ip.cidr	IP CIDR Object.
luci.json	
luci.jsonc	LuCI JSON parsing and serialization library.
luci.jsonc.parser	LuCI JSON parser instance.
luci.model.ipkg	
luci.model.uci	
luci.rpcc	
luci.rpcc.ruci	
luci.sys	
luci.sys.init	LuCI system utilities / init related functions.
luci.sys.iptparser	
luci.sys.net	LuCI system utilities / network related functions.
luci.sys.process	LuCI system utilities / process related functions.
luci.sys.user	LuCI system utilities / user related functions.
luci.sys.wifi	LuCI system utilities / wifi related functions.
luci.util	
nixio	General POSIX IO library.
nixio.CHANGELOG	Changes and improvements.
nixio.CryptoHash	Cryptographical Hash and HMAC object.
nixio.File	Large File Object.
nixio.README	General Information.
nixio.Socket	Socket Object.
nixio.TLSContext	Transport Layer Security Context Object.
nixio.TLSSocket	TLS Socket Object.
nixio.UnifiedIO	Unified high-level I/O utility API for Files, Sockets and TLS-Sockets.
nixio.bin	Binary operations and conversion.
nixio.bit	Bitfield operators and mainpulation functions.
nixio.crypto	Cryptographical library.
nixio.fs	Low-level and high-level filesystem manipulation library.

Class *luci.dispatcher*

Functions

_ ()	No-op function used to mark translation entries for menu labels.
alias (...)	Create a redirect to another dispatching node.
arcombine (trg1, trg2)	Create a combined dispatching target for non argv and argv requests.
assign (path, clone, title, order)	Clone a node of the dispatching tree to another

	position.
build_url (...)	Build the URL relative to the server webroot from given virtual path.
call (name, ...)	Create a function-call dispatching target.
cbi (model)	Create a CBI model dispatching target.
createindex ()	Generate the dispatching index using the native file-cache based strategy.
createtree ()	Create the dispatching tree from the index.
dispatch (request)	Dispatches a LuCI virtual path.
entry (path, target, title, order)	Create a new dispatching node and define common parameters.
error404 (message)	Send a 404 error code and render the "error404" template if available.
error500 (message)	Send a 500 error code and render the "error500" template if available.
firstchild ()	Alias the first (lowest order) page automatically
form (model)	Create a CBI form model dispatching target.
get (...)	Fetch or create a dispatching node without setting the target module or enabling the node.
httpdispatch (request)	Dispatch an HTTP request.
lookup (...)	Lookup node in dispatching tree.
node (...)	Fetch or create a new dispatching node.
node_childs (node)	Return a sorted table of visible children within a given node
node_visible (node)	Check whether a dispatch node shall be visible
rewrite (n, ...)	Rewrite the first x path values of the request.
template (name)	Create a template render dispatching target.
translate (text)	Access the luci.i18n translate() api.

Functions

_ ()

No-op function used to mark translation entries for menu labels. This function does not actually translate the given argument but is used by build/i18n-scan.pl to find translatable entries.

---

alias (...)

Create a redirect to another dispatching node.

Parameters

- ...: Virtual path destination

---

arcombine (trg1, trg2)

Create a combined dispatching target for non argv and argv requests.

Parameters

- trg1: Overview Target
- trg2: Detail Target

---

assign (path, clone, title, order)

Clone a node of the dispatching tree to another position.

Parameters

- path: Virtual path destination
- clone: Virtual path source
- title: Destination node title (optional)
- order: Destination node order value (optional)

Return value:

Dispatching tree node

---

build_url (...)

Build the URL relative to the server webroot from given virtual path.

Parameters

- ...: Virtual path

Return value:

Relative URL

---

call (name, ...)

Create a function-call dispatching target.

Parameters

- name: Target function of local controller
- ...: Additional parameters passed to the function

---

cbi (model)

Create a CBI model dispatching target.

Parameters

- model: CBI model to be rendered

---

createindex ()

Generate the dispatching index using the native file-cache based strategy.

---

createtree ()

Create the dispatching tree from the index. Build the index before if it does not exist yet.

---

dispatch (request)

Dispatches a LuCI virtual path.

Parameters

- request: Virtual path

---

entry (path, target, title, order)

Create a new dispatching node and define common parameters.

Parameters

- path: Virtual path
- target: Target function to call when dispatched.
- title: Destination node title
- order: Destination node order value (optional)

Return value:

Dispatching tree node

---

error404 (message)

Send a 404 error code and render the "error404" template if available.

Parameters

- message: Custom error message (optional)

Return value:

false

---

error500 (message)

Send a 500 error code and render the "error500" template if available.

Parameters

- message: Custom error message (optional)#

Return value:

false

---

firstchild ()

Alias the first (lowest order) page automatically

---

form (model)

Create a CBI form model dispatching target.

Parameters

- model: CBI form model tpo be rendered

---

get (...)

Fetch or create a dispatching node without setting the target module or enabling the node.

Parameters

- ...: Virtual path

Return value:

Dispatching tree node

---

httpdispatch (request)

Dispatch an HTTP request.

Parameters

- request: LuCI HTTP Request object

---

lookup (...)

Lookup node in dispatching tree.

Parameters

- ...: Virtual path

Return value:

Node object, canonical url or nil if the path was not found.

---

node (...)

Fetch or create a new dispatching node.

Parameters

- ...: Virtual path

Return value:

Dispatching tree node

---

node_childs (node)

Return a sorted table of visible children within a given node

Parameters

- node: Dispatch node

Return value:

Ordered table of child node names

---

node_visible (node)

Check whether a dispatch node shall be visible

Parameters

- node: Dispatch node

Return value:

Boolean indicating whether the node should be visible

---

rewrite (n, ...)

Rewrite the first x path values of the request.

Parameters

- n: Number of path values to replace
- ...: Virtual path to replace removed path values with

---

template (name)

Create a template render dispatching target.

Parameters

- name: Template to be rendered

---

translate (text)

Access the luci.i18n translate() api.

Parameters

- text: Text to translate

Class *luci.http*

Functions

build_querystring (table)	Create a querystring out of a table of key - value pairs.
close ()	Close the HTTP-Connection.
content ()	Return the request content if the request was of unknown type.
formvalue (name, noparse)	Get a certain HTTP input value or a table of all input values.
formvaluetable (prefix)	Get a table of all HTTP input values with a certain prefix.
getcookie (name)	Get the value of a certain HTTP-Cookie.
getenv (name)	Get the value of a certain HTTP environment variable or the environment table itself.
header (key, value)	Send a HTTP-Header.
mimedecode_message_body (src, msg, filecb)	Decode a mime encoded http message body with multipart/form-data Content-Type.
parse_message_body (src, msg, filecb)	Try to extract and decode a http message body from the given ltn12 source.
prepare_content (mime)	Set the mime type of following content data.
redirect (url)	Redirects the client to a new URL and closes the connection.
setfilehandler (callback)	Set a handler function for incoming user file uploads.
source ()	Get the RAW HTTP input source
splice (fp, size)	Splice data from a filedescriptor to the client.
status (code, message)	Set the HTTP status code and status message.
urldecode (str, no_plus)	Return the URL-decoded equivalent of a string.
urldecode_message_body (src, msg)	Decode an urlencoded http message body with

	application/x-www-urlencoded Content-Type.
urldecode_params (url, tbl)	Extract and split urlencoded data pairs, separated bei either "&" or ";" from given url or string.
urlencode (str)	Return the URL-encoded equivalent of a string.
urlencode_params (tbl)	Encode each key-value-pair in given table to x-www-urlencoded format, separated by "&".
write (content, src_err)	Send a chunk of content data to the client.
write_json (data)	Send the given data as JSON encoded string.

Functions

`build_querystring` (table)

Create a querystring out of a table of key - value pairs.

Parameters

- table: Query string source table

Return value:

Encoded HTTP query string

`close` ()

Close the HTTP-Connection.

`content` ()

Return the request content if the request was of unknown type.

Return values:

1. HTTP request body
2. HTTP request body length

`formvalue` (name, noparse)

Get a certain HTTP input value or a table of all input values.

Parameters

- name: Name of the GET or POST variable to fetch

- noparse: Don't parse POST data before getting the value

Return value:

HTTP input value or table of all input value

---

formvaluetable (prefix)

Get a table of all HTTP input values with a certain prefix.

Parameters

- prefix: Prefix

Return value:

Table of all HTTP input values with given prefix

---

getcookie (name)

Get the value of a certain HTTP-Cookie.

Parameters

- name: Cookie Name

Return value:

String containing cookie data

---

getenv (name)

Get the value of a certain HTTP environment variable or the environment table itself.

Parameters

- name: Environment variable

Return value:

HTTP environment value or environment table

---

header (key, value)

Send a HTTP-Header.

Parameters

- key: Header key
- value: Header value

---

mimedecode_message_body (src, msg, filecb)

Decode a mime encoded http message body with multipart/form-data Content-Type. Stores all extracted data associated with its parameter name in the params table within the given message object. Multiple parameter values are stored as tables, ordinary ones as strings. If an optional file callback function is given then it is

fed with the file contents chunk by chunk and only the extracted file name is stored within the params table. The callback function will be called subsequently with three arguments: o Table containing decoded (name, file) and raw (headers) mime header data o String value containing a chunk of the file data o Boolean which indicates whether the current chunk is the last one (eof)

Parameters

- src: Ltn12 source function
- msg: HTTP message object
- filecb: File callback function (optional)

Return values:

1. Value indicating successful operation (not nil means "ok")
2. String containing the error if unsuccessful

See also:

- parse_message_header

---

parse_message_body (src, msg, filecb)

Try to extract and decode a http message body from the given ltn12 source. This function will examine the Content-Type within the given message object to select the appropriate content decoder. Currently the application/x-www-urlencoded and application/form-data mime types are supported. If the encountered content encoding can't be handled then the whole message body will be stored unaltered as "content" property within the given message object.

Parameters

- src: Ltn12 source function
- msg: HTTP message object
- filecb: File data callback (optional, see mimedecode_message_body())

Return values:

1. Value indicating successful operation (not nil means "ok")
2. String containing the error if unsuccessful

See also:

- parse_message_header

---

prepare_content (mime)

Set the mime type of following content data.

Parameters

- mime: Mimetype of following content

---

redirect (url)

Redirects the client to a new URL and closes the connection.

Parameters

- url: Target URL

---

setfilehandler (callback)

Set a handler function for incoming user file uploads.

Parameters

- callback: Handler function

---

source ()

Get the RAW HTTP input source

Return value:

HTTP LTN12 source

---

splice (fp, size)

Splice data from a filedescriptor to the client.

Parameters

- fp: File descriptor
- size: Bytes to splice (optional)

---

status (code, message)

Set the HTTP status code and status message.

Parameters

- code: Status code
- message: Status message

---

urldecode (str, no_plus)

Return the URL-decoded equivalent of a string.

Parameters

- str: URL-encoded string

- no_plus: Don't decode + to " "

Return value:

URL-decoded string

See also:

- urlencode

---

urldecode_message_body (src, msg)

Decode an urlencoded http message body with application/x-www-urlencoded Content-Type. Stores all extracted data associated with its parameter name in the params table within the given message object. Multiple parameter values are stored as tables, ordinary ones as strings.

Parameters

- src: Ltn12 source function
- msg: HTTP message object

Return values:

1. Value indicating successful operation (not nil means "ok")
2. String containing the error if unsuccessful

See also:

- parse_message_header

---

urldecode_params (url, tbl)

Extract and split urlencoded data pairs, separated bei either "&" or ";" from given url or string. Returns a table with urldecoded values. Simple parameters are stored as string values associated with the parameter name within the table. Parameters with multiple values are stored as array containing the corresponding values.

Parameters

- url: The url or string which contains x-www-urlencoded form data
- tbl: Use the given table for storing values (optional)

Return value:

Table containing the urldecoded parameters

See also:

- urlencode_params

---

urlencode (str)

Return the URL-encoded equivalent of a string.

Parameters

- str: Source string

Return value:

URL-encoded string

See also:

- urldecode

---

urlencode_params (tbl)

Encode each key-value-pair in given table to x-www-urlencoded format, separated by "&". Tables are encoded as parameters with multiple values by repeating the parameter name with each value.

Parameters

- tbl: Table with the values

Return value:

String containing encoded values

See also:

- urldecode_params

---

write (content, src_err)

Send a chunk of content data to the client. This function is as a valid LTN12 sink. If the content chunk is nil this function will automatically invoke close.

Parameters

- content: Content chunk
- src_err: Error object from source (optional)

See also:

- close

---

write_json (data)

Send the given data as JSON encoded string.

Parameters

- data: Data to send

Class *luci.http.conditionals*

Functions

if_match (req, stat)	14.24 / If-Match Test whether the given message object contains an "If-Match" header and compare it against the given stat object.
if_modified_since (req, stat)	14.25 / If-Modified-Since Test whether the given message object contains an "If-Modified-Since" header and compare it against the given stat object.
if_none_match (req, stat)	14.26 / If-None-Match Test whether the given message object contains an "If-None-Match" header and compare it against the given stat object.
if_range (req, stat)	14.27 / If-Range The If-Range header is currently not implemented due to the lack of general byte range stuff in luci.http.protocol .
if_unmodified_since (req, stat)	14.28 / If-Unmodified-Since Test whether the given message object contains an "If-Unmodified-Since" header and compare it against the given stat object.
mk_etag (stat)	Implement 14.19 / ETag.

Functions

---

if_match (req, stat)

14.24 / If-Match Test whether the given message object contains an "If-Match" header and compare it against the given stat object.

Parameters

- req: HTTP request message object
- stat: A file.stat object

Return values:

1. Boolean indicating whether the precondition is ok
2. Alternative status code if the precondition failed

---

if_modified_since (req, stat)

14.25 / If-Modified-Since Test whether the given message object contains an "If-Modified-Since" header and compare it against the given stat object.

Parameters

- req: HTTP request message object
- stat: A file.stat object

Return values:

1. Boolean indicating whether the precondition is ok
2. Alternative status code if the precondition failed
3. Table containing extra HTTP headers if the precondition failed

---

if_none_match (req, stat)

14.26 / If-None-Match Test whether the given message object contains an "If-None-Match" header and compare it against the given stat object.

Parameters

- req: HTTP request message object
- stat: A file.stat object

Return values:

1. Boolean indicating whether the precondition is ok
2. Alternative status code if the precondition failed
3. Table containing extra HTTP headers if the precondition failed

---

if_range (req, stat)

14.27 / If-Range The If-Range header is currently not implemented due to the lack of general byte range stuff in luci.http.protocol . This function will always return false, 412 to indicate a failed precondition.

Parameters

- req: HTTP request message object
- stat: A file.stat object

Return values:

1. Boolean indicating whether the precondition is ok
2. Alternative status code if the precondition failed

---

if_unmodified_since (req, stat)

14.28 / If-Unmodified-Since Test whether the given message object contains an "If-Unmodified-Since" header and compare it against the given stat object.

Parameters

- req: HTTP request message object
- stat: A file.stat object

Return values:

1. Boolean indicating whether the precondition is ok
2. Alternative status code if the precondition failed

---

mk_etag (stat)

Implement 14.19 / ETag.

Parameters

- stat: A file.stat structure

Return value:

String containing the generated tag suitable for ETag headers

Class *luci.http.date*

Functions

compare (d1, d2)	Compare two dates which can either be unix epoch times or HTTP date strings.
to_http (time)	Convert the given unix epoch time to valid HTTP date string.
to_unix (data)	Parse given HTTP date string and convert it to unix epoch time.
tz_offset (tz)	Return the time offset in seconds between the UTC and given time zone.

Functions

---

compare (d1, d2)

Compare two dates which can either be unix epoch times or HTTP date strings.

Parameters

- d1: The first date or epoch time to compare
- d2: The first date or epoch time to compare

Return values:

1. -1 - if d1 is lower then d2
2. 0 - if both dates are equal
3. 1 - if d1 is higher then d2

---

to_http (time)

Convert the given unix epoch time to valid HTTP date string.

Parameters

- time: Unix epoch time

Return value:

String containing the formatted date

---

to_unix (data)

Parse given HTTP date string and convert it to unix epoch time.

Parameters

- data: String containing the date

Return value:

Unix epoch time

---

tz_offset (tz)

Return the time offset in seconds between the UTC and given time zone.

Parameters

- tz: Symbolic or numeric timezone specifier

Return value:

Time offset to UTC in seconds

Class *luci.http.mime*

Functions

to_ext (mimetype)	Return corresponding extension for a given mime type or nil if the given mime-type is unknown.
to_mime (filename)	Extract extension from a filename and return corresponding

	mime-type or "application/octet-stream" if the extension is unknown.

Functions

to_ext (mimetype)

Return corresponding extension for a given mime type or nil if the given mime-type is unknown.

Parameters

- mimetype: The mimetype to retrieve the extension from

Return value:

String with the extension or nil for unknown type

---

to_mime (filename)

Extract extension from a filename and return corresponding mime-type or "application/octet-stream" if the extension is unknown.

Parameters

- filename: The filename for which the mime type is guessed

Return value:

String containign the determined mime type

Class *luci.i18n*

Functions

dump ()	Return all currently loaded translation strings as a key-value table.
setlanguage (lang)	Set the context default translation language.
translate (key)	Return the translated value for a specific translation key.
translatef (key, ...)	Return the translated value for a specific translation key and use it as sprintf pattern.

Functions

dump ()

Return all currently loaded translation strings as a key-value table. The key is the hexadecimal representation of the translation key while the value is the translated text content.

Return value:

Key-value translation string table.

---

setlanguage (lang)

Set the context default translation language.

Parameters

- lang: An IETF/BCP 47 language tag or ISO3166 country code, e.g. "en-US" or "de"

Return value:

The effective loaded language, e.g. "en" for "en-US" - or nil on failure

---

translate (key)

Return the translated value for a specific translation key.

Parameters

- key: Default translation text

Return value:

Translated string

---

translatef (key, ...)

Return the translated value for a specific translation key and use it as sprintf pattern.

Parameters

- key: Default translation text
- ...: Format parameters

Return value:

Translated and formatted string

Class *luci.ip*

LuCI IP calculation and netlink access library.

Functions

new (address, netmask)	Construct a new luci.ip.cidr instance and autodetect the address family.

IPv4 (address, netmask)	Construct a new IPv4 luci.ip.cidr instance.
IPv6 (address, netmask)	Construct a new IPv6 luci.ip.cidr instance.
MAC (address, netmask)	Construct a new MAC luci.ip.cidr instance.
checkip4 (address)	Verify an IPv4 address.
checkip6 (address)	Verify an IPv6 address.
checkmac (address)	Verify an ethernet MAC address.
route (address, source)	Determine the route leading to the given destination.
routes (filter, callback)	Fetch all routes, optionally matching the given criteria.
neighbors (filter, callback)	Fetches entries from the IPv4 ARP and IPv6 neighbour kernel table
link (device)	Fetch basic device information

Functions

---

new (address, netmask)

Construct a new luci.ip.cidr instance and autodetect the address family. Throws an error if the given strings do not represent a valid address or if the given optional netmask is of a different family.

Parameters

- address: String containing a valid IPv4 or IPv6 address, optionally with prefix size (CIDR notation) or netmask separated by slash.
- netmask: String containing a valid IPv4 or IPv6 netmask or number containing a prefix size in bits (*0..32* for IPv4, *0..128* for IPv6). Overrides mask embedded in the first argument if specified. (optional)

Usage:

```
addr = luci.ip.new("10.24.0.1/24")
addr = luci.ip.new("10.24.0.1/255.255.255.0")
addr = luci.ip.new("10.24.0.1", "255.255.255.0")        -- separate netmask
addr = luci.ip.new("10.24.0.1/24", 16)                  -- override netmask

addr6 = luci.ip.new("fe80::221:63ff:fe75:aa17/64")
addr6 = luci.ip.new("fe80::221:63ff:fe75:aa17/ffff:ffff:ffff:ffff::")
addr6 = luci.ip.new("fe80::221:63ff:fe75:aa17", "ffff:ffff:ffff:ffff::")
addr6 = luci.ip.new("fe80::221:63ff:fe75:aa17/64", 128) -- override netmask
```

Return value:

A *luci.ip.cidr* object representing the given address/mask range.

See also:

- IPv4
- IPv6
- MAC

IPv4 (address, netmask)

Construct a new IPv4 luci.ip.cidr instance. Throws an error if the given string does not represent a valid IPv4 address or if the given optional netmask is of a different family.

Parameters

- address: String containing a valid IPv4, optionally with prefix size (CIDR notation) or netmask separated by slash.
- netmask: String containing a valid IPv4 netmask or number containing a prefix size between *0* and *32* bit. Overrides mask embedded in the first argument if specified. (optional)

Usage:

```
addr = luci.ip.IPv4("10.24.0.1/24")
addr = luci.ip.IPv4("10.24.0.1/255.255.255.0")
addr = luci.ip.IPv4("10.24.0.1", "255.255.255.0")        -- separate netmask
addr = luci.ip.IPv4("10.24.0.1/24", 16)                  -- override netmask
```

Return value:

A *luci.ip.cidr* object representing the given IPv4 range.

See also:

- IPv6
- MAC

IPv6 (address, netmask)

Construct a new IPv6 luci.ip.cidr instance. Throws an error if the given string does not represent a valid IPv6 address or if the given optional netmask is of a different family.

Parameters

- address: String containing a valid IPv6, optionally with prefix size (CIDR notation) or netmask separated by slash.

- netmask: String containing a valid IPv4 netmask or number containing a prefix size between 0 and 128 bit. Overrides mask embedded in the first argument if specified. (optional)

Usage:

```
addr6 = luci.ip.IPv6("fe80::221:63ff:fe75:aa17/64")
addr6 = luci.ip.IPv6("fe80::221:63ff:fe75:aa17/ffff:ffff:ffff:ffff::")
addr6 = luci.ip.IPv6("fe80::221:63ff:fe75:aa17", "ffff:ffff:ffff:ffff::")
addr6 = luci.ip.IPv6("fe80::221:63ff:fe75:aa17/64", 128) -- override netmask
```

Return value:

A *luci.ip.cidr* object representing the given IPv6 range.

See also:

- IPv4
- MAC

MAC (address, netmask)

Construct a new MAC luci.ip.cidr instance. Throws an error if the given string does not represent a valid ethernet MAC address or if the given optional mask is of a different family.

Parameters

- address: String containing a valid ethernet MAC address, optionally with prefix size (CIDR notation) or mask separated by slash.
- netmask: String containing a valid MAC address mask or number containing a prefix size between 0 and 48 bit. Overrides mask embedded in the first argument if specified. (optional)

Usage:

```
intel_macs = luci.ip.MAC("C0:B6:F9:00:00:00/24")
intel_macs = luci.ip.MAC("C0:B6:F9:00:00:00/FF:FF:FF:0:0:0")
intel_macs = luci.ip.MAC("C0:B6:F9:00:00:00", "FF:FF:FF:0:0:0")
intel_macs = luci.ip.MAC("C0:B6:F9:00:00:00/24", 48) -- override mask
```

Return value:

A *luci.ip.cidr* object representing the given MAC address range.

See also:

- IPv4
- IPv6

---

checkip4 (address)

Verify an IPv4 address. Checks whether given argument is a preexisting luci.ip.cidr IPv4 address instance or a string literal convertible to an IPv4 address and returns a plain Lua string containing the canonical representation of the address. If the argument is not a valid address, returns nothing. This function is intended to aid in safely verifying address literals without having to deal with exceptions.

Parameters

- address: String containing a valid IPv4 address or existing luci.ip.cidr IPv4 instance.

Usage:

```
ipv4 = luci.ip.checkip4(luci.ip.new("127.0.0.1"))  -- "127.0.0.1"
ipv4 = luci.ip.checkip4("127.0.0.1")               -- "127.0.0.1"
ipv4 = luci.ip.checkip4("nonesense")               -- nothing
ipv4 = luci.ip.checkip4(123)                       -- nothing
ipv4 = luci.ip.checkip4(nil)                       -- nothing
ipv4 = luci.ip.checkip4()                          -- nothing
```

Return value:

A string representing the given IPv4 address.

See also:

- checkip6
- checkmac

---

checkip6 (address)

Verify an IPv6 address. Checks whether given argument is a preexisting luci.ip.cidr IPv6 address instance or a string literal convertible to an IPv6 address and returns a plain Lua string containing the canonical representation of the address. If the argument is not a valid address, returns nothing. This function is intended to aid in safely verifying address literals without having to deal with exceptions.

Parameters

- address: String containing a valid IPv6 address or existing luci.ip.cidr IPv6 instance.

Usage:

```
ipv6 = luci.ip.checkip6(luci.ip.new("0:0:0:0:0:0:0:1"))  -- "::1"
ipv6 = luci.ip.checkip6("0:0:0:0:0:0:0:1")               -- "::1"
ipv6 = luci.ip.checkip6("nonesense")                     -- nothing
```

```
ipv6 = luci.ip.checkip6(123)                    -- nothing
ipv6 = luci.ip.checkip6(nil)                    -- nothing
ipv6 = luci.ip.checkip6()                       -- nothing
```

Return value:

A string representing the given IPv6 address.

See also:

- checkip4
- checkmac

---

checkmac (address)

Verify an ethernet MAC address. Checks whether given argument is a preexisting luci.ip.cidr MAC address instance or a string literal convertible to an ethernet MAC and returns a plain Lua string containing the canonical representation of the address. If the argument is not a valid address, returns nothing. This function is intended to aid in safely verifying address literals without having to deal with exceptions.

Parameters

- address: String containing a valid MAC address or existing luci.ip.cidr MAC address instance.

Usage:

```
mac = luci.ip.checkmac(luci.ip.new("00-11-22-cc-dd-ee"))  -- "00:11:22:CC:DD:EE"
mac = luci.ip.checkmac("00:11:22:cc:dd:ee")               -- "00:11:22:CC:DD:EE"
mac = luci.ip.checkmac("nonesense")                       -- nothing
mac = luci.ip.checkmac(123)                               -- nothing
mac = luci.ip.checkmac(nil)                               -- nothing
mac = luci.ip.checkmac()                                  -- nothing
```

Return value:

A string representing the given MAC address.

See also:

- checkip4
- checkip6

---

route (address, source)

Determine the route leading to the given destination.

Parameters

- address: A *luci.ip.cidr* instance or a string containing a valid IPv4 or IPv6 range as specified by *luci.ip.new()*.
- source: A *luci.ip.cidr* instance or a string containing the preferred source address for route selection (optional).

Usage:

- Find default gateway by getting route to Google's public NS server

```
rt = luci.ip.route("8.8.8.8")
if rt ~= nil then
  print("gateway is", rt.gw)
end
```

-
- Determine IPv6 upstream interface

```
rt = luci.ip.route("2001::/7")
if rt ~= nil then
  print("ipv6 upstream device is", rt.dev)
end
```

-

Return value:

Table containing the fields described below.

Field	Description
type	Route type with one of the following numeric values: 1 *RTN_UNICAST* - Gateway or direct route 2 *RTN_LOCAL* - Accept locally 3 *RTN_BROADCAST* - Accept locally as broadcast send as broadcast

	4 RTN_ANYCAST - Accept locally as broadcast but send as unicast 5 RTN_MULTICAST - Multicast route
family	Number containing the route family, 4 for IPv4 or 6 for IPv6
dest	Destination luci.ip.cidr instance
gw	Gateway luci.ip.cidr instance (optional)
from	Source address luci.ip.cidr instance (optional)
src	Preferred source luci.ip.cidr instance (optional)
dev	String containing the name of the outgoing interface
iif	String containing the name of the incoming interface (optional)
table	Number of the associated routing table (0..65535)
proto	Number of the associated routing protocol
scope	Number describing the scope of the route, most commonly 0 for global or 253 for on-link
metric	Number describing the route metric (optional)
expires	Number of seconds the prefix is valid (IPv6 only, optional)
error	Route destination error code (optional)

See also:

- routes

---

routes (filter, callback)

Fetch all routes, optionally matching the given criteria.

Parameters

- filter:

Table containing one or more of the possible filter criteria described below (optional)

-

Field	Description
family	Number describing the address family to return - 4 selects IPv4 routes, 6 IPv6 ones. Any other value selects both.
iif	String containing the incoming route interface to match.
oif	String containing the outgoing route interface to match.
type	Numeric type to match, e.g. 1 for unicast.
scope	Numeric scope to match, e.g. 253 for onlink.
proto	Numeric protocol to match, e.g. 2 for boot.
table	Numeric routing table to match (0..65535).
gw	String containing the gateway address to match. Can be in any notation specified by luci.ip.new(). Prefix matching is performed when comparing the routes, e.g. "192.168.1.0/24" would select routes with gateway

	addresses *192.168.1.1 .. 192.168.1.255*.
dest	String containing the destination to match. Prefix matching is performed.
from	String containing the source address to match. Prefix matching is performed.
src	String containing the preferred source address to match. Prefix matching is performed.
dest_exact	String containing the destination to match. Exact matching is performed, e.g. *dest = "0.0.0.0/0"* would match *any* IPv4 route while *dest_exact = "0.0.0.0/0"* will *only* match the default route.
from_exact	String containing the source address to match. Exact matching is performed.

- callback:

Callback function to invoke for each found route instead of returning one table of route objects (optional)

Usage:

- Find all IPv4 default routes:

```
luci.ip.routes({ dest_exact = "0.0.0.0/0" }, function(rt)
  print(rt.type, rt.gw, rt.dev)
end)
```

- Find all global IPv6 prefixes on the current system:

```
luci.ip.routes({ from = "2001::/7" }, function(rt)
  print(rt.from)
end)
```

- Fetch all IPv4 routes:

```
routes = luci.ip.routes({ family = 4 })
for _, rt in ipairs(routes) do
  print(rt.dest, rt.gw, rt.dev)
end
```

Return value:

If no callback function is provided, a table of routes as specified by *luci.ip.route()* is returned. If a callback function is given, it is invoked for each route and nothing is returned.

See also:

- route

---

neighbors (filter, callback)

Fetches entries from the IPv4 ARP and IPv6 neighbour kernel table

Parameters

- filter:

Table containing one or more of the possible filter criteria described below (optional)

Field	Description
family	Number describing the address family to return - `4` selects IPv4 ARP, `6` select IPv6 neighbour entries. Any other value selects both.
dev	String containing the associated interface to match.
dest	String containing the associated address to match. Can be in any notation specified by *luci.ip.new()*. Prefix matching is performed when comparing the addresses, e.g. "192.168.1.0/24"

	would select ARP entries for 192.168.1.1 .. 192.168.1.255.
mac	String containing MAC address to match.

- callback:

Callback function to invoke for each found neighbour entry instead of returning one table of neighbour entries (optional)

Usage:

- Find all ARP neighbours in the LAN:

```
luci.ip.neighbors({ dest = "192.168.0.0/16" }, function(n)
  print(n.dest, n.mac)
end)
```

- Find all active IPv6 addresses of host with given MAC:

```
luci.ip.neighbors({ family = 6, mac = "00:21:63:75:aa:17" },
  function(n)
          print(n.dest)
  end)
```

Return value:

If no callback function is provided, a table of neighbour entries is returned. If a callback function is given, it is invoked for each entry and nothing is returned. A neighbour entry is a table containing the following fields:

Field	Description
family	Number containing the neighbour entry family, 4 for IPv4 ARP or 6 for IPv6 NDP

dev	String containing the associated device of the neighbour entry
dest	IP address *luci.ip.cidr* instance
mac	MAC address *luci.ip.cidr* instance
router	Boolean "true" if the neighbour entry is a router (IPv6, optional)
proxy	Boolean "true" if this is a proxy entry (optional)
incomplete	Boolean "true" if the entry is in incomplete state (optional)
reachable	Boolean "true" if the entry is in reachable state (optional)
stale	Boolean "true" if the entry is stale (optional)
delay	Boolean "true" if the entry is delayed (optional)
probe	Boolean "true" if the entry is in probe state (optional)
failed	Boolean "true" if the entry is in failed state (optional)
noarp	Boolean "true" if the entry is not caused by NDP or ARP (optional)
permanent	Boolean "true" if the entry was statically configured from userspace (optional)

link (device)

Fetch basic device information

Parameters

- device: String containing the network device to query

Usage:

- Test whether device br-lan exists:

```
print(luci.ip.link("br-lan").name ~= nil)
```

-
- Query MAC address of eth0:

```
print(luci.ip.link("eth0").mac)
```

-

Return value:

If the given interface is found, a table containing the fields described below is returned, else an empty table.

Field	Description
up	Boolean indicating whether the device is in IFF_RUNNING state
type	Numeric value indicating the type of the device, e.g. *1* for ethernet.
name	String containing the name of the device
master	If queried device is a bridge port, string containing the name of parent bridge device (optional)
mtu	Number containing the current MTU of the device
qlen	Number containing the TX queue length of the device
mac	MAC address *luci.ip.cidr* instance representing the device ethernet address

Object Instance *luci.ip.cidr*

IP CIDR Object. Represents an IPv4 or IPv6 address range.

Functions

cidr:is4 ()	Checks whether the CIDR instance is an IPv4 address range
cidr:is4rfc1918 ()	Checks whether the CIDR instance is within the private RFC1918 address space
cidr:is4linklocal ()	Checks whether the CIDR instance is an IPv4 link local (Zeroconf) address
cidr:is6 ()	Checks whether the CIDR instance is an IPv6 address range
cidr:is6linklocal ()	Checks whether the CIDR instance is an IPv6 link local address
cidr:is6mapped4 ()	Checks whether the CIDR instance is an IPv6 mapped IPv4 address
cidr:ismac ()	Checks whether the CIDR instance is an ethernet MAC address range
cidr:ismaclocal ()	Checks whether the CIDR instance is a locally administered (LAA) MAC address
cidr:ismacmcast ()	Checks whether the CIDR instance is a multicast MAC address
cidr:lower (addr)	Checks whether this CIDR instance is lower than the given argument.
cidr:higher (addr)	Checks whether this CIDR instance is higher than the given argument.
cidr:equal (addr)	Checks whether this CIDR instance is equal to the given argument.
cidr:prefix (mask)	Get or set prefix size of CIDR instance.
cidr:network (mask)	Derive network address of CIDR instance.
cidr:host ()	Derive host address of CIDR instance.
cidr:mask (mask)	Derive netmask of CIDR instance.
cidr:broadcast (mask)	Derive broadcast address of CIDR instance.
cidr:mapped4 ()	Derive mapped IPv4 address of CIDR instance.
cidr:tomac ()	Derive MAC address of IPv6 link local CIDR instance.
cidr:tolinklocal ()	Derive IPv6 link local address from MAC address CIDR instance.
cidr:contains (addr)	Test whether CIDR contains given range.
cidr:add (amount, inplace)	Add given amount to CIDR instance.
cidr:sub (amount, inplace)	Subtract given amount from CIDR instance.

cidr:minhost ()	Calculate the lowest possible host address within this CIDR instance.
cidr:maxhost ()	Calculate the highest possible host address within this CIDR instance.
cidr:string ()	Convert CIDR instance into string representation.

Functions

---

cidr:is4 ()

Checks whether the CIDR instance is an IPv4 address range

Return value:

true if the CIDR is an IPv4 range, else *false*

See also:

- cidr:is6
- cidr:ismac

---

cidr:is4rfc1918 ()

Checks whether the CIDR instance is within the private RFC1918 address space

Usage:

```
local addr = luci.ip.new("192.168.45.2/24")
if addr:is4rfc1918() then
        print("Is a private address")
end
```

Return value:

true if the entire range of this CIDR lies within one of the ranges *10.0.0.0-10.255.255.255*, *172.16.0.0-172.31.0.0* or *192.168.0.0-192.168.255.255*, else *false*.

---

cidr:is4linklocal ()

Checks whether the CIDR instance is an IPv4 link local (Zeroconf) address

Usage:

```
local addr = luci.ip.new("169.254.34.125")
if addr:is4linklocal() then
        print("Is a zeroconf address")
end
```

Return value:

true if the entire range of this CIDR lies within the range the range *169.254.0.0-169.254.255.255*, else *false*.

---

cidr:is6 ()

Checks whether the CIDR instance is an IPv6 address range

Return value:

true if the CIDR is an IPv6 range, else *false*

See also:

- cidr:is4
- cidr:ismac

---

cidr:is6linklocal ()

Checks whether the CIDR instance is an IPv6 link local address

Usage:

```
local addr = luci.ip.new("fe92:53a:3216:af01:221:63ff:fe75:aa17/64")
if addr:is6linklocal() then
        print("Is a linklocal address")
end
```

Return value:

true if the entire range of this CIDR lies within the range the *fe80::/10* range, else *false*.

---

cidr:is6mapped4 ()

Checks whether the CIDR instance is an IPv6 mapped IPv4 address

Usage:

```
local addr = luci.ip.new("::ffff:192.168.1.1")
if addr:is6mapped4() then
        print("Is a mapped IPv4 address")
end
```

Return value:

true if the address is an IPv6 mapped IPv4 address in the form *::ffff:1.2.3.4*.

---

cidr:ismac ()

Checks whether the CIDR instance is an ethernet MAC address range

Return value:

true if the CIDR is a MAC address range, else *false*

See also:

- cidr:is4
- cidr:is6

---

cidr:ismaclocal ()

Checks whether the CIDR instance is a locally administered (LAA) MAC address

Usage:

```
local mac = luci.ip.new("02:C0:FF:EE:00:01")
if mac:ismaclocal() then
  print("Is an LAA MAC address")
end
```

Return value:

true if the MAC address sets the locally administered bit.

---

cidr:ismacmcast ()

Checks whether the CIDR instance is a multicast MAC address

Usage:

```
local mac = luci.ip.new("01:00:5E:7F:00:10")
if addr:ismacmcast() then
  print("Is a multicast MAC address")
end
```

Return value:

true if the MAC address sets the multicast bit.

---

cidr:lower (addr)

Checks whether this CIDR instance is lower than the given argument. The comparisation follows these rules:

- An IPv4 address is always lower than an IPv6 address and IPv6 addresses are considered lower than MAC addresses
- Prefix sizes are ignored

Parameters

- addr: A *luci.ip.cidr* instance or a string convertible by *luci.ip.new()* to compare against.

Usage:

```
local addr = luci.ip.new("192.168.1.1")
print(addr:lower(addr)) -- false
print(addr:lower("10.10.10.10/24")) -- false
print(addr:lower(luci.ip.new("::1"))) -- true
print(addr:lower(luci.ip.new("192.168.200.1"))) -- true
print(addr:lower(luci.ip.new("00:14:22:01:23:45"))) -- true
```

Return value:

true if this CIDR is lower than the given address, else *false*.

See also:

- cidr:higher
- cidr:equal

---

cidr:higher (addr)

Checks whether this CIDR instance is higher than the given argument. The comparisation follows these rules:

- An IPv4 address is always lower than an IPv6 address and IPv6 addresses are considered lower than MAC addresses
- Prefix sizes are ignored

Parameters

- addr: A *luci.ip.cidr* instance or a string convertible by *luci.ip.new()* to compare against.

Usage:

```
local addr = luci.ip.new("192.168.1.1")
print(addr:higher(addr)) -- false
print(addr:higher("10.10.10.10/24")) -- true
print(addr:higher(luci.ip.new("::1"))) -- false
print(addr:higher(luci.ip.new("192.168.200.1"))) -- false
print(addr:higher(luci.ip.new("00:14:22:01:23:45"))) -- false
```

Return value:

true if this CIDR is higher than the given address, else *false*.

See also:

- cidr:lower
- cidr:equal

---

cidr:equal (addr)

Checks whether this CIDR instance is equal to the given argument.

Parameters

- addr: A *luci.ip.cidr* instance or a string convertible by *luci.ip.new()* to compare against.

Usage:

```
local addr = luci.ip.new("192.168.1.1")
print(addr:equal(addr)) -- true
print(addr:equal("192.168.1.1")) -- true
print(addr:equal(luci.ip.new("::1"))) -- false

local addr6 = luci.ip.new("::1")
print(addr6:equal("0:0:0:0:0:0:0:1/64")) -- true
print(addr6:equal(luci.ip.new("fe80::221:63ff:fe75:aa17"))) -- false

local mac = luci.ip.new("00:14:22:01:23:45")
print(mac:equal("0:14:22:1:23:45")) -- true
print(mac:equal(luci.ip.new("01:23:45:67:89:AB")) -- false
```

Return value:

true if this CIDR is equal to the given address, else *false*.

See also:

- cidr:lower
- cidr:higher

---

cidr:prefix (mask)

Get or set prefix size of CIDR instance. If the optional mask parameter is given, the prefix size of this CIDR is altered else the current prefix size is returned.

Parameters

- mask: Either a number containing the number of bits (*0..32* for IPv4, *0..128* for IPv6 or *0..48* for MAC addresses) or a string containing a valid netmask (optional)

Usage:

```
local range = luci.ip.new("192.168.1.1/255.255.255.0")
```

```
print(range:prefix()) -- 24

range:prefix(16)
print(range:prefix()) -- 16

range:prefix("255.255.255.255")
print(range:prefix()) -- 32
```

Return value:

Bit count of the current prefix size

---

cidr:network (mask)

Derive network address of CIDR instance. Returns a new CIDR instance representing the network address of this instance with all host parts masked out. The used prefix size can be overridden by the optional mask parameter.

Parameters

- mask: Either a number containing the number of bits (*0..32* for IPv4, *0..128* for IPv6 or *0..48* for MAC addresses) or a string containing a valid netmask (optional)

Usage:

```
local range = luci.ip.new("192.168.62.243/255.255.0.0")
print(range:network())                -- "192.168.0.0"
print(range:network(24))              -- "192.168.62.0"
print(range:network("255.255.255.0")) -- "192.168.62.0"

local range6 = luci.ip.new("fd9b:62b3:9cc5:0:221:63ff:fe75:aa17/64")
print(range6:network())               -- "fd9b:62b3:9cc5::"
```

Return value:

CIDR instance representing the network address

---

cidr:host ()

Derive host address of CIDR instance. This function essentially constructs a copy of this CIDR with the prefix size set to *32* for IPv4, *128* for IPv6 or *48* for MAC addresses.

Usage:

```
local range = luci.ip.new("172.19.37.45/16")
print(range)        -- "172.19.37.45/16"
print(range:host()) -- "172.19.37.45"
```

Return value:

CIDR instance representing the host address

---

cidr:mask (mask)

Derive netmask of CIDR instance. Constructs a CIDR instance representing the netmask of this instance. The used prefix size can be overridden by the optional mask parameter.

Parameters

- mask: Either a number containing the number of bits (*0..32* for IPv4, *0..128* for IPv6 or *0..48* for MAC addresses) or a string containing a valid netmask (optional)

Usage:

```
local range = luci.ip.new("172.19.37.45/16")
print(range:mask())            -- "255.255.0.0"
print(range:mask(24))          -- "255.255.255.0"
print(range:mask("255.0.0.0")) -- "255.0.0.0"
```

Return value:

CIDR instance representing the netmask

---

cidr:broadcast (mask)

Derive broadcast address of CIDR instance. Constructs a CIDR instance representing the broadcast address of this instance. The used prefix size can be overridden by the optional mask parameter. This function has no effect on IPv6 or MAC address instances, it will return nothing in this case.

Parameters

- mask: Either a number containing the number of bits (*0..32* for IPv4) or a string containing a valid netmask (optional)

Usage:

```
local range = luci.ip.new("172.19.37.45/16")
print(range:broadcast())            -- "172.19.255.255"
print(range:broadcast(24))          -- "172.19.37.255"
print(range:broadcast("255.0.0.0")) -- "172.255.255.255"
```

Return value:

Return a new CIDR instance representing the broadcast address if this instance is an IPv4 range, else return nothing.

---

cidr:mapped4 ()

Derive mapped IPv4 address of CIDR instance. Constructs a CIDR instance representing the IPv4 address of the IPv6 mapped IPv4

address in this instance. This function has no effect on IPv4 instances, MAC address instances or IPv6 instances which are not a mapped address, it will return nothing in this case.

Usage:

```
local addr = luci.ip.new("::ffff:172.16.19.1")
print(addr:mapped4()) -- "172.16.19.1"
```

Return value:

Return a new CIDR instance representing the IPv4 address if this instance is an IPv6 mapped IPv4 address, else return nothing.

---

cidr:tomac ()

Derive MAC address of IPv6 link local CIDR instance. Constructs a CIDR instance representing the MAC address contained in the IPv6 link local address of this instance. This function has no effect on IPv4 instances, MAC address instances or IPv6 instances which are not a link local address, it will return nothing in this case.

Usage:

```
local addr = luci.ip.new("fe80::6666:b3ff:fe47:e1b9")
print(addr:tomac()) -- "64:66:B3:47:E1:B9"
```

Return value:

Return a new CIDR instance representing the MAC address if this instance is an IPv6 link local address, else return nothing.

---

cidr:tolinklocal ()

Derive IPv6 link local address from MAC address CIDR instance. Constructs a CIDR instance representing the IPv6 link local address of the MAC address represented by this instance. This function has no effect on IPv4 instances or IPv6 instances, it will return nothing in this case.

Usage:

```
local mac = luci.ip.new("64:66:B3:47:E1:B9")
print(mac:tolinklocal()) -- "fe80::6666:b3ff:fe47:e1b9"
```

Return value:

Return a new CIDR instance representing the IPv6 link local address.

---

cidr:contains (addr)

Test whether CIDR contains given range.

Parameters

- addr: A *luci.ip.cidr* instance or a string convertible by *luci.ip.new()* to test.

Usage:

```
local range = luci.ip.new("10.24.0.0/255.255.0.0")
print(range:contains("10.24.5.1"))  -- true
print(range:contains("::1"))        -- false
print(range:contains("10.0.0.0/8")) -- false

local range6 = luci.ip.new("fe80::/10")
print(range6:contains("fe80::221:63f:fe75:aa17/64"))         -- true
print(range6:contains("fd9b:6b3:c5:0:221:63f:fe75:aa17/64")) -- false

local intel_macs = luci.ip.MAC("C0:B6:F9:00:00:00/24")
print(intel_macs:contains("C0:B6:F9:A3:C:11"))  -- true
print(intel_macs:contains("64:66:B3:47:E1:B9")) -- false
```

Return value:

true if this instance fully contains the given address else *false*.

---

cidr:add (amount, inplace)

Add given amount to CIDR instance. If the result would overflow the maximum address space, the result is set to the highest possible address.

Parameters

- amount: A numeric value between 0 and 0xFFFFFFFF, a *luci.ip.cidr* instance or a string convertible by *luci.ip.new()*.
- inplace: If *true*, modify this instance instead of returning a new derived CIDR instance.

Usage:

```
local addr = luci.ip.new("192.168.1.1/24")
print(addr:add(250))           -- "192.168.1.251/24"
print(addr:add("0.0.99.0"))    -- "192.168.100.1/24"

addr:add(256, true)            -- true
print(addr)                    -- "192.168.2.1/24

addr:add("255.0.0.0", true)    -- false (overflow)
print(addr)                    -- "255.255.255.255/24

local addr6 = luci.ip.new("fe80::221:63f:fe75:aa17/64")
print(addr6:add(256))          -- "fe80::221:63f:fe75:ab17/64"
print(addr6:add("::ffff:0"))   -- "fe80::221:640:fe74:aa17/64"
```

```
addr6:add(256, true)           -- true
print(addr6)                   -- "fe80::221:63f:fe75:ab17/64

addr6:add("ffff::", true)      -- false (overflow)
print(addr6)                   -- "ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff/64"

local mac = luci.ip.new("00:14:22:01:23:45")
print(mac:add(256))            -- "00:14:22:01:24:45"
print(mac:add("0:0:0:0:FF:0")  -- "00:14:22:02:22:45"

mac:add(256, true)             -- true
print(mac)                     -- "00:14:22:01:24:45"

mac:add("FF:FF:0:0:0:0", true) -- false (overflow)
print(mac)                     -- "FF:FF:FF:FF:FF:FF"
```

Return value:

- When adding inplace: Return *true* if the addition succeeded or *false* when the addition overflowed.
- When deriving new CIDR: Return new instance representing the value of this instance plus the added amount or the highest possible address if the addition overflowed the available address space.

---

cidr:sub (amount, inplace)

Subtract given amount from CIDR instance. If the result would under, the lowest possible address is returned.

Parameters

- amount: A numeric value between 0 and 0xFFFFFFFF, a *luci.ip.cidr* instance or a string convertible by *luci.ip.new()*.
- inplace: If *true*, modify this instance instead of returning a new derived CIDR instance.

Usage:

```
local addr = luci.ip.new("192.168.1.1/24")
print(addr:sub(256))          -- "192.168.0.1/24"
print(addr:sub("0.168.0.0")) -- "192.0.1.1/24"

addr:sub(256, true)           -- true
```

```
print(addr)                    -- "192.168.0.1/24

addr:sub("255.0.0.0", true)  -- false (underflow)
print(addr)                    -- "0.0.0.0/24

local addr6 = luci.ip.new("fe80::221:63f:fe75:aa17/64")
print(addr6:sub(256))          -- "fe80::221:63f:fe75:a917/64"
print(addr6:sub("::ffff:0")) -- "fe80::221:63e:fe76:aa17/64"

addr:sub(256, true)            -- true
print(addr)                    -- "fe80::221:63f:fe75:a917/64"

addr:sub("ffff::", true)       -- false (underflow)
print(addr)                    -- "::/64"

local mac = luci.ip.new("00:14:22:01:23:45")
print(mac:sub(256))              -- "00:14:22:01:22:45"
print(mac:sub("0:0:0:0:FF:0")  -- "00:14:22:00:24:45"

mac:sub(256, true)               -- true
print(mac)                       -- "00:14:22:01:22:45"

mac:sub("FF:FF:0:0:0:0", true) -- false (overflow)
print(mac)                       -- "00:00:00:00:00:00"
```

Return value:

- When subtracting inplace: Return *true* if the subtraction succeeded or *false* when the subtraction underflowed.
- When deriving new CIDR: Return new instance representing the value of this instance minus the subtracted amount or the lowest address if the subtraction underflowed.

---

cidr:minhost ()

Calculate the lowest possible host address within this CIDR instance.

Usage:

```
local addr = luci.ip.new("192.168.123.56/24")
print(addr:minhost())  -- "192.168.123.1"

local addr6 = luci.ip.new("fd9b:62b3:9cc5:0:221:63ff:fe75:aa17/64")
print(addr6:minhost()) -- "fd9b:62b3:9cc5::1"
```

```
local mac = luci.ip.new("00:14:22:01:22:45/32")
print(mac:minhost())   -- "00:14:22:01:00:01"
```

Return value:

Returns a new CIDR instance representing the lowest host address within this range.

`cidr:maxhost ()`

Calculate the highest possible host address within this CIDR instance.

Usage:

```
local addr = luci.ip.new("192.168.123.56/24")
print(addr:maxhost())  -- "192.168.123.254" (.255 is broadcast)

local addr6 = luci.ip.new("fd9b:62b3:9cc5:0:221:63ff:fe75:aa17/64")
print(addr6:maxhost()) -- "fd9b:62b3:9cc5:0:ffff:ffff:ffff:ffff"

local mac = luci.ip.new("00:14:22:01:22:45/32")
print(mac:maxhost())   -- "00:14:22:01:FF:FF"
```

Return value:

Returns a new CIDR instance representing the highest host address within this range.

`cidr:string ()`

Convert CIDR instance into string representation. If the prefix size of instance is less than 32 for IPv4, 128 for IPv6 or 48 for MACs, the address is returned in the form "address/prefix" otherwise just "address". It is usually not required to call this function directly as CIDR objects define it as __tostring function in the associated metatable.

Return value:

Returns a string representing the range or address of this CIDR instance

Object Instance *luci.json*

Functions

ActiveDecoder (customnull)	Create a new Active JSON-Decoder.
ActiveDecoder:get ()	Fetches one JSON-object from given source
Decoder (customnull)	Create a new JSON-Decoder.
Decoder:get ()	Get the decoded data packets after the rawdata has been sent to the sink.

Decoder:sink ()	Create an LTN12 sink from the decoder object which accepts the JSON-Data.
Encoder (data, buffersize, fastescape)	Create a new JSON-Encoder.
Encoder:source ()	Create an LTN12 source providing the encoded JSON-Data.
decode (json)	Directly decode a JSON string
encode (obj)	Direcly encode a Lua object into a JSON string.
null ()	Null replacement function

Functions

ActiveDecoder (customnull)

Create a new Active JSON-Decoder.

Parameters

- customnull: Use luci.json.null instead of nil for decoding null

Return value:

Active JSON-Decoder

ActiveDecoder:get ()

Fetches one JSON-object from given source

Return value:

Decoded object

Decoder (customnull)

Create a new JSON-Decoder.

Parameters

- customnull: Use luci.json.null instead of nil for decoding null

Return value:

JSON-Decoder

Decoder:get ()

Get the decoded data packets after the rawdata has been sent to the sink.

Return value:

Decoded data

Decoder:sink ()

Create an LTN12 sink from the decoder object which accepts the JSON-Data.

Return value:

LTN12 sink

Encoder (data, buffersize, fastescape)

Create a new JSON-Encoder.

Parameters

- data: Lua-Object to be encoded.
- buffersize: Blocksize of returned data source.
- fastescape: Use non-standard escaping (don't escape control chars)

Return value:

JSON-Encoder

Encoder:source ()

Create an LTN12 source providing the encoded JSON-Data.

Return value:

LTN12 source

---

decode (json)

Directly decode a JSON string

Parameters

- json: JSON-String

Return value:

Lua object

---

encode (obj)

Direcly encode a Lua object into a JSON string.

Parameters

- obj: Lua Object

Return value:

JSON string

---

null ()

Null replacement function

Return value:

null

Class *luci.jsonc*

LuCI JSON parsing and serialization library. The luci.jsonc class is a high level Lua binding to the JSON-C library to allow reading and writing JSON data with minimal overhead.

Functions

new ()	Construct a new luci.jsonc.parser instance.
parse (json)	Parse a complete JSON string and convert it into a Lua data structure.
stringify (data, pretty)	Convert given Lua data into a JSON string.

Functions

---

new ()

Construct a new luci.jsonc.parser instance.

Usage:

```
parser = luci.jsonc.new()
```

Return value:

A *luci.jsonc.parser* object representing a JSON-C tokener.

---

parse (json)

Parse a complete JSON string and convert it into a Lua data structure.

Parameters

- json: A string containing the JSON data to parse, must be either a JSON array or a JSON object.

Usage:

```
data = luci.jsonc.parse('{ "name": "John", "age": 34 }')
print(data.name)  -- "John"
```

Return value:

On success, a table containing the parsed JSON data is returned, on failure the function returns *nil* and a string containing the reason of the parse error.

See also:

- stringify

stringify (data, pretty)

Convert given Lua data into a JSON string. This function recursively converts the given Lua data into a JSON string, ignoring any unsupported data. Lua tables are converted into JSON arrays if they only contain integer keys, mixed tables are turned into JSON objects with any existing numeric keys converted into strings. Lua functions, coroutines and userdata objects are ignored and Lua numbers are converted to integers if they do not contain fractional values.

Parameters

- data: The Lua data to convert, can be a table, string, boolean or number.
- pretty: A boolean value indicating whether the resulting JSON should be pretty printed.

Usage:

```
json = luci.jsonc.stringify({ item = true, values = { 1, 2, 3 } })
print(json)  -- '{"item":true,"values":[1,2,3]}'
```

Return value:

Returns a string containing the JSON representation of the given Lua data.

See also:

- parse

Object Instance *luci.jsonc.parser*

LuCI JSON parser instance. A JSON parser instance is useful to parse JSON data chunk by chunk, without the need to assemble all data in advance.

Functions

parser:parse (json)	Parses one chunk of JSON data.
parser:get ()	Convert parsed JSON data into Lua table.
parser:set (data)	Put Lua data into the parser.
parser:sink ()	Generate an ltn12-compatible sink.
parser:stringify (pretty)	Serialize current parser state as JSON.

Functions

---

parser:parse (json)

Parses one chunk of JSON data.

Parameters

- json: String containing the JSON fragment to parse

Usage:

```
parser = luci.jsonc.new()

while true do
	chunk = ...  -- fetch a cunk of data, e.g. from a socket
	finish, errmsg = parser.parse(chunk)

	if finish == nil then
		error("Cannot parse JSON: " .. errmsg)
	end

	if finish == true then
		break
	end
end
```

Return value:

- *true* if a complete JSON object has been parsed and no further input is expected.
- *false* if further input is required
- *nil* if an error was encountered while parsing the current chunk. In this case a string describing the parse error is returned as second value.

See also:

- parser:get

---

parser:get ()

Convert parsed JSON data into Lua table.

Usage:

```
parser = luci.jsonc.new()
parser:parse('{ "example": "test" }')

data = parser:get()
```

```
print(data.example)  -- "test"
```

Return value:

Parsed JSON object converted into a Lua table or *nil* if the parser didn't finish or encountered an error.

See also:

- parser:parse

---

parser:set (data)

Put Lua data into the parser.

Parameters

- data: Lua data to put into the parser object. The data is converted to an internal JSON representation that can be dumped with *stringify()*. The conversion follows the rules described in *luci.jsonc.stringify*.

Usage:

```
parser = luci.jsonc.new()
parser:set({ "some", "data" })
```

Return value:

Nothing is returned.

See also:

- parser:stringify

---

parser:sink ()

Generate an ltn12-compatible sink.

Usage:

```
parser = luci.jsonc.new()
ltn12.pump.all(ltn12.source.file(io.input()), parser:sink())
print(parser:get())
```

Return value:

Returns a function that can be used as an ltn12 sink.

---

parser:stringify (pretty)

Serialize current parser state as JSON.

Parameters

- pretty: A boolean value indicating whether the resulting JSON should be pretty printed.

Usage:

```
parser = luci.jsonc.new()
parser:parse('{ "example": "test" }')
print(parser:serialize())  -- '{"example":"test"}'
```

Return value:

Returns the serialized JSON data of this parser instance.

Class *luci.model.ipkg*

Functions

compare_versions (ver1, ver2, comp)	lua version of opkg compare-versions
find (pat, cb)	Find packages that match the given pattern.
info (pkg)	Return information about installed and available packages.
install (...)	Install one or more packages.
installed (pkg)	Determine whether a given package is installed.
list_all (pat, cb)	List all packages known to opkg.
list_installed (pat, cb)	List installed packages.
overlay_root ()	Determines the overlay root used by opkg.
remove (...)	Remove one or more packages.
status (pkg)	Return the package status of one or more packages.
update ()	Update package lists.
upgrade ()	Upgrades all installed packages.

Functions

---

compare_versions (ver1, ver2, comp)

lua version of opkg compare-versions

Parameters

- ver1: string version 1
- ver2: string version 2
- comp: string compare versions using "<=" or "<" lower-equal ">" or ">=" greater-equal "=" equal "<<" lower ">>" greater "~=" not equal

Return value:

Boolean indicating the status of the compare

---

find (pat, cb)

Find packages that match the given pattern.

Parameters

- pat: Find packages whose names or descriptions match this pattern, nil results in zero results
- cb: Callback function invoked for each patckage, receives name, version and description as arguments

Return value:

nothing

---

info (pkg)

Return information about installed and available packages.

Parameters

- pkg: Limit output to a (set of) packages

Return value:

Table containing package information

---

install (...)

Install one or more packages.

Parameters

- ...: List of packages to install

Return values:

1. Boolean indicating the status of the action
2. OPKG return code, STDOUT and STDERR

---

installed (pkg)

Determine whether a given package is installed.

Parameters

- pkg: Package

Return value:

Boolean

---

list_all (pat, cb)

List all packages known to opkg.

Parameters

- pat: Only find packages matching this pattern, nil lists all packages
- cb: Callback function invoked for each package, receives name, version and description as arguments

Return value:

nothing

---

list_installed (pat, cb)

List installed packages.

Parameters

- pat: Only find packages matching this pattern, nil lists all packages
- cb: Callback function invoked for each package, receives name, version and description as arguments

Return value:

nothing

---

overlay_root ()

Determines the overlay root used by opkg.

Return value:

String containing the directory path of the overlay root.

---

remove (...)

Remove one or more packages.

Parameters

- ...: List of packages to install

Return values:

1. Boolean indicating the status of the action
2. OPKG return code, STDOUT and STDERR

---

status (pkg)

Return the package status of one or more packages.

Parameters

- pkg: Limit output to a (set of) packages

Return value:

Table containing package status information

---

update ()

Update package lists.

Return values:

1. Boolean indicating the status of the action
2. OPKG return code, STDOUT and STDERR

---

upgrade ()

Upgrades all installed packages.

Return values:

1. Boolean indicating the status of the action
2. OPKG return code, STDOUT and STDERR

Object Instance *luci.model.uci*

Functions

Cursor:add (config, type)	Add an anonymous section.
Cursor:apply (rollback)	Applies UCI configuration changes.
Cursor:changes (config)	Get a table of saved but uncommitted changes.
Cursor:commit (config)	Commit saved changes.
Cursor:confirm ()	Confirms UCI apply process.
Cursor:delete (config, section, option)	Deletes a section or an option.
Cursor:delete_all (config, type, comparator)	Delete all sections of a given type that match certain criteria.
Cursor:foreach (config, type, callback)	Call a function for every section of a certain type.
Cursor:get (config, section, option)	Get a section type or an option
Cursor:get_all (config, section)	Get all sections of a config or all values of a section.
Cursor:get_bool (config, section, option)	Get a boolean option and return it's value as true or false.
Cursor:get_confdir ()	Get the configuration directory.
Cursor:get_first (config, type, option, default)	Get the given option from the first section with the given type.
Cursor:get_list (config, section, option)	Get an option or list and return values

	as table.
Cursor:get_savedir ()	Get the directory for uncomitted changes.
Cursor:get_session_id ()	Get the effective session ID.
Cursor:load (config)	Manually load a config.
Cursor:revert (config)	Revert saved but uncommitted changes.
Cursor:rollback ()	Cancels UCI apply process.
Cursor:rollback_pending ()	Checks whether a pending rollback is scheduled.
Cursor:save (config)	Saves changes made to a config to make them committable.
Cursor:section (config, type, name, values)	Create a new section and initialize it with data.
Cursor:set (config, section, option, value)	Set a value or create a named section.
Cursor:set_confdir (directory)	Set the configuration directory.
Cursor:set_list (config, section, option, value)	Set given values as list.
Cursor:set_savedir (directory)	Set the directory for uncommitted changes.
Cursor:set_session_id (id)	Set the effective session ID.
Cursor:substate ()	Create a sub-state of this cursor.
Cursor:tset (config, section, values)	Updated the data of a section using data from a table.
Cursor:unload (config)	Discard changes made to a config.
cursor ()	Create a new UCI-Cursor.
cursor_state ()	Create a new Cursor initialized to the state directory.

Functions

Cursor:add (config, type)

Add an anonymous section.

Parameters

- config: UCI config

- type: UCI section type

Return value:

Name of created section

Cursor:apply (rollback)

Applies UCI configuration changes. If the rollback parameter is set to true, the apply function will invoke the rollback mechanism which causes the configuration to be automatically reverted if no confirm() call occurs within a certain timeout. The current default timeout is 30s and can be increased using the "luci.apply.timeout" uci configuration key.

Parameters

- rollback: Enable rollback mechanism

Return value:

Boolean whether operation succeeded

Cursor:changes (config)

Get a table of saved but uncommitted changes.

Parameters

- config: UCI config

Return value:

Table of changes

See also:

- Cursor:save

Cursor:commit (config)

Commit saved changes.

Parameters

- config: UCI config

Return value:

Boolean whether operation succeeded

See also:

- Cursor:revert
- Cursor:save

Cursor:confirm ()

Confirms UCI apply process. If a previous UCI apply with rollback has been invoked using apply(true), this function confirms the process and cancels the pending rollback timer. If no apply with rollback session is active, the function has no effect and returns with a "No data" error.

Return value:

Boolean whether operation succeeded

Cursor:delete (config, section, option)

Deletes a section or an option.

Parameters

- config: UCI config
- section: UCI section name
- option: UCI option (optional)

Return value:

Boolean whether operation succeeded

Cursor:delete_all (config, type, comparator)

Delete all sections of a given type that match certain criteria.

Parameters

- config: UCI config
- type: UCI section type
- comparator: Function that will be called for each section and returns a boolean whether to delete the current section (optional)

Cursor:foreach (config, type, callback)

Call a function for every section of a certain type.

Parameters

- config: UCI config
- type: UCI section type
- callback: Function to be called

Return value:

Boolean whether operation succeeded

Cursor:get (config, section, option)

Get a section type or an option

Parameters

- config: UCI config
- section: UCI section name

- option: UCI option (optional)

Return value:

UCI value

Cursor:get_all (config, section)

Get all sections of a config or all values of a section.

Parameters

- config: UCI config
- section: UCI section name (optional)

Return value:

Table of UCI sections or table of UCI values

Cursor:get_bool (config, section, option)

Get a boolean option and return it's value as true or false.

Parameters

- config: UCI config
- section: UCI section name
- option: UCI option

Return value:

Boolean

Cursor:get_confdir ()

Get the configuration directory.

Return value:

Configuration directory

Cursor:get_first (config, type, option, default)

Get the given option from the first section with the given type.

Parameters

- config: UCI config
- type: UCI section type
- option: UCI option (optional)
- default: Default value (optional)

Return value:

UCI value

Cursor:get_list (config, section, option)

Get an option or list and return values as table.

Parameters

- config: UCI config
- section: UCI section name
- option: UCI option

Return value:

table. If the option was not found, you will simply get an empty table.

Cursor:get_savedir ()

Get the directory for uncomitted changes.

Return value:

Save directory

Cursor:get_session_id ()

Get the effective session ID.

Return value:

String containing the session ID

Cursor:load (config)

Manually load a config.

Parameters

- config: UCI config

Return value:

Boolean whether operation succeeded

See also:

- Cursor:save
- Cursor:unload

Cursor:revert (config)

Revert saved but uncommitted changes.

Parameters

- config: UCI config

Return value:

Boolean whether operation succeeded

See also:

- Cursor:commit
- Cursor:save

Cursor:rollback ()

Cancels UCI apply process. If a previous UCI apply with rollback has been invoked using apply(true), this function cancels the process and rolls back the configuration to the pre-apply state. If no apply with rollback session is active, the function has no effect and returns with a "No data" error.

Return value:

Boolean whether operation succeeded

Cursor:rollback_pending ()

Checks whether a pending rollback is scheduled. If a previous UCI apply with rollback has been invoked using apply(true), and has not been confirmed or rolled back yet, this function returns true and the remaining time until rollback in seconds. If no rollback is pending, the function returns false. On error, the function returns false and an additional string describing the error.

Return values:

1. Boolean whether rollback is pending
2. Remaining time in seconds

Cursor:save (config)

Saves changes made to a config to make them committable.

Parameters

- config: UCI config

Return value:

Boolean whether operation succeeded

See also:

- Cursor:load
- Cursor:unload

Cursor:section (config, type, name, values)

Create a new section and initialize it with data.

Parameters

- config: UCI config
- type: UCI section type
- name: UCI section name (optional)
- values: Table of key - value pairs to initialize the section with

Return value:

Name of created section

Cursor:set (config, section, option, value)

Set a value or create a named section. When invoked with three arguments *config*, *sectionname*, *sectiontype*, then a named section of the given type is created. When invoked with four arguments *config*, *sectionname*, *optionname* and *optionvalue* then the value of the specified option is set to the given value.

Parameters

- config: UCI config
- section: UCI section name
- option: UCI option or UCI section type
- value: UCI value or nothing if you want to create a section

Return value:

Boolean whether operation succeeded

Cursor:set_confdir (directory)

Set the configuration directory.

Parameters

- directory: UCI configuration directory

Return value:

Boolean whether operation succeeded

Cursor:set_list (config, section, option, value)

Set given values as list. Setting a list option to an empty list has the same effect as deleting the option.

Parameters

- config: UCI config
- section: UCI section name
- option: UCI option
- value: Value or table. Non-table values will be set as single item UCI list.

Return value:

Boolean whether operation succeeded

Cursor:set_savedir (directory)

Set the directory for uncommitted changes.

Parameters

- directory: UCI changes directory

Return value:

Boolean whether operation succeeded

Cursor:set_session_id (id)

Set the effective session ID.

Parameters

- id: String containing the session ID to set

Return value:

Boolean whether operation succeeded

Cursor:substate ()

Create a sub-state of this cursor. The sub-state is tied to the parent cursor, means it the parent unloads or loads configs, the sub state will do so as well.

Return value:

UCI state cursor tied to the parent cursor

Cursor:tset (config, section, values)

Updated the data of a section using data from a table.

Parameters

- config: UCI config
- section: UCI section name (optional)
- values: Table of key - value pairs to update the section with

---

Cursor:unload (config)

Discard changes made to a config.

Parameters

- config: UCI config

Return value:

Boolean whether operation succeeded

See also:

- Cursor:load
- Cursor:save

---

cursor ()

Create a new UCI-Cursor.

Return value:

UCI-Cursor

`cursor_state` ()

Create a new Cursor initialized to the state directory.

Return value:

UCI cursor

Object Instance *luci.rpcc*

Functions

Client:proxy (prefix)	Create a transparent RPC proxy.
Client:request (method, params, notification)	Request an RP call and get the response.

Functions

`Client:proxy` (prefix)

Create a transparent RPC proxy.

Parameters

- prefix: Method prefix

Return value:

RPC Proxy object

`Client:request` (method, params, notification)

Request an RP call and get the response.

Parameters

- method: Remote method
- params: Parameters
- notification: Notification only?

Return value:

response

Object Instance *luci.rpcc.ruci*

Functions

factory (rpccl)	Create a new UCI over RPC proxy.

Functions

`factory` (rpccl)
Create a new UCI over RPC proxy.

Parameters

- rpccl: RPC client

Return value:
Network transparent UCI module

Class *luci.sys*

Functions

call (...)	Execute a given shell command and return the error code
dmesg ()	Retrieves the output of the "dmesg" command.
exec (command)	Execute a given shell command and capture its standard output
getenv (var)	Retrieve environment variables.
hostname (String)	Get or set the current hostname.
httpget (url, stream, target)	Returns the contents of a documented referred by an URL.
mounts ()	Retrieve information about currently mounted file systems.
reboot ()	Initiate a system reboot.
syslog ()	Retrieves the output of the "logread" command.
uniqueid (bytes)	Generates a random id with specified length.
uptime ()	Returns the current system uptime stats.

Functions

`call` (...)
Execute a given shell command and return the error code

Parameters

- ...: Command to call

Return value:

Error code of the command

---

dmesg ()

Retrieves the output of the "dmesg" command.

Return value:

String containing the current log buffer

---

exec (command)

Execute a given shell command and capture its standard output

Parameters

- command: Command to call

Return value:

String containing the return the output of the command

---

getenv (var)

Retrieve environment variables. If no variable is given then a table containing the whole environment is returned otherwise this function returns the corresponding string value for the given name or nil if no such variable exists.

Parameters

- var: Name of the environment variable to retrieve (optional)

Return values:

1. String containing the value of the specified variable
2. Table containing all variables if no variable name is given

---

hostname (String)

Get or set the current hostname.

Parameters

- String: containing a new hostname to set (optional)

Return value:

String containing the system hostname

---

httpget (url, stream, target)

Returns the contents of a documented referred by an URL.

Parameters

- url: The URL to retrieve
- stream: Return a stream instead of a buffer
- target: Directly write to target file name

Return value:

String containing the contents of given the URL

---

mounts ()

Retrieve information about currently mounted file systems.

Return value:

Table containing mount information

---

reboot ()

Initiate a system reboot.

Return value:

Return value of os.execute()

---

syslog ()

Retrieves the output of the "logread" command.

Return value:

String containing the current log buffer

---

uniqueid (bytes)

Generates a random id with specified length.

Parameters

- bytes: Number of bytes for the unique id

Return value:

String containing hex encoded id

---

uptime ()

Returns the current system uptime stats.

Return value:

String containing total uptime in seconds

Class *luci.sys.init*

LuCI system utilities / init related functions.

Functions

disable (name)	Disable the given init script

enable (name)	Enable the given init script
enabled (name)	Test whether the given init script is enabled
index (name)	Get the index of he given init script
names ()	Get the names of all installed init scripts
start (name)	Start the given init script
stop (name)	Stop the given init script

Functions

---

disable (name)

Disable the given init script

Parameters

- name: Name of the init script

Return value:

Boolean indicating success

---

enable (name)

Enable the given init script

Parameters

- name: Name of the init script

Return value:

Boolean indicating success

---

enabled (name)

Test whether the given init script is enabled

Parameters

- name: Name of the init script

Return value:

Boolean indicating whether init is enabled

---

index (name)

Get the index of he given init script

Parameters

- name: Name of the init script

Return value:

Numeric index value

---

names ()

Get the names of all installed init scripts

Return value:

Table containing the names of all inistalled init scripts

---

start (name)

Start the given init script

Parameters

- name: Name of the init script

Return value:

Boolean indicating success

---

stop (name)

Stop the given init script

Parameters

- name: Name of the init script

Return value:

Boolean indicating success

Object Instance *luci.sys.iptparser*

Functions

IptParser (family)	Create a new iptables parser object.
IptParser:chain (table, chain)	Return the given firewall chain within the given table name.
IptParser:chains (table)	Find the names of all chains within the given table name.
IptParser:is_custom_target (target)	Test whether the given target points to a custom chain.
IptParser:resync ()	Rebuild the internal lookup table, for example when rules have changed through external commands.
IptParser:tables ()	Find the names of all tables.

Functions

IptParser (family)

Create a new iptables parser object.

Parameters

- family: Number specifying the address family. 4 for IPv4, 6 for IPv6

Return value:

IptParser instance

IptParser:chain (table, chain)

Return the given firewall chain within the given table name.

Parameters

- table: String containing the table name
- chain: String containing the chain name

Return value:

Table containing the fields "policy", "packets", "bytes" and "rules". The "rules" field is a table of rule tables.

IptParser:chains (table)

Find the names of all chains within the given table name.

Parameters

- table: String containing the table name

Return value:

Table of chain names in the order they occur.

IptParser:is_custom_target (target)

Test whether the given target points to a custom chain.

Parameters

- target: String containing the target action

Return value:

Boolean indicating whether target is a custom chain.

IptParser:resync ()

Rebuild the internal lookup table, for example when rules have changed through external commands.

Return value:

nothing

IptParser:tables ()

Find the names of all tables.

Return value:

Table of table names.

Class *luci.sys.net*

LuCI system utilities / network related functions.

Functions

arptable ()	Returns the current arp-table entries as two-dimensional table.
conntrack ()	Returns conntrack information
deviceinfo ()	Return information about available network interfaces.
devices ()	Determine the names of available network interfaces.
host_hints ()	Returns a two-dimensional table of host hints.
ipv4_hints ()	Returns a two-dimensional table of IPv4 address hints.
ipv6_hints ()	Returns a two-dimensional table of IPv6 address hints.
mac_hints ()	Returns a two-dimensional table of mac address hints.
pingtest (host)	Tests whether the given host responds to ping probes.
routes ()	Returns the current kernel routing table entries.
routes6 ()	Returns the current ipv6 kernel routing table entries.

Functions

---

arptable ()

Returns the current arp-table entries as two-dimensional table.

Return value:

Table of table containing the current arp entries. The following fields are defined for arp entry objects: { "IP address", "HW address", "HW type", "Flags", "Mask", "Device" }

---

conntrack ()

Returns conntrack information

Return value:

Table with the currently tracked IP connections

---

deviceinfo ()

Return information about available network interfaces.

Return value:

Table containing all current interface names and their information

devices ()

Determine the names of available network interfaces.

Return value:

Table containing all current interface names

host_hints ()

Returns a two-dimensional table of host hints.

Return value:

Table of table containing known hosts from various sources, indexed by mac address. Each subtable contains at least one of the fields "name", "ipv4" or "ipv6".

ipv4_hints ()

Returns a two-dimensional table of IPv4 address hints.

Return value:

Table of table containing known hosts from various sources. Each entry contains the values in the following order: ["ip", "name"]

ipv6_hints ()

Returns a two-dimensional table of IPv6 address hints.

Return value:

Table of table containing known hosts from various sources. Each entry contains the values in the following order: ["ip", "name"]

mac_hints ()

Returns a two-dimensional table of mac address hints.

Return value:

Table of table containing known hosts from various sources. Each entry contains the values in the following order: ["mac", "name"]

pingtest (host)

Tests whether the given host responds to ping probes.

Parameters

- host: String containing a hostname or IPv4 address

Return value:

Number containing 0 on success and >= 1 on error

routes ()

Returns the current kernel routing table entries.

Return value:

Table of tables with properties of the corresponding routes. The following fields are defined for route entry tables: { "dest", "gateway", "metric", "refcount", "usecount", "irtt", "flags", "device" }

routes6 ()

Returns the current ipv6 kernel routing table entries.

Return value:

Table of tables with properties of the corresponding routes. The following fields are defined for route entry tables: { "source", "dest", "nexthop", "metric", "refcount", "usecount", "flags", "device" }

Class *luci.sys.process*

LuCI system utilities / process related functions.

Functions

exec (commend, stdout, stderr, nowait)	Execute a process, optionally capturing stdio.
info ()	Get the current process id.
list ()	Retrieve information about currently running processes.
setgroup (gid)	Set the gid of a process identified by given pid.
setuser (uid)	Set the uid of a process identified by given pid.
signal (pid, sig)	Send a signal to a process identified by given pid.

Functions

exec (commend, stdout, stderr, nowait)

Execute a process, optionally capturing stdio. Executes the process specified by the given argv vector, e.g. {"/bin/sh","-c","echo 1"} and waits for it to terminate unless a true value has been passed for the "nowait" parameter. When a function value is passed for the stdout or stderr arguments, the passed function is repeatedly called for each chunk read from the corresponding stdio stream. The read data is passed as string containing at most 4096 bytes at a time. When a true, non-function value is passed for the stdout or stderr arguments, the data of the corresponding stdio

stream is read into an internal string buffer and returned as "stdout" or "stderr" field respectively in the result table. When a true value is passed to the nowait parameter, the function does not await process termination but returns as soon as all captured stdio streams have been closed or - if no streams are captured - immediately after launching the process.

Parameters

- commend: Table containing the argv vector to execute
- stdout: Callback function or boolean to indicate capturing (optional)
- stderr: Callback function or boolean to indicate capturing (optional)
- nowait: Don't wait for process termination when true (optional)

Return value:

Table containing at least the fields "code" which holds the exit status of the invoked process or "-1" on error and "pid", which contains the process id assigned to the spawned process. When stdout and/or stderr capturing has been requested, it additionally contains "stdout" and "stderr" fields respectively, holding the captured stdio data as string.

---

info ()

Get the current process id.

Return value:

Number containing the current pid

---

list ()

Retrieve information about currently running processes.

Return value:

Table containing process information

---

setgroup (gid)

Set the gid of a process identified by given pid.

Parameters

- gid: Number containing the Unix group id

Return values:

1. Boolean indicating successful operation
2. String containing the error message if failed
3. Number containing the error code if failed

---

setuser (uid)

Set the uid of a process identified by given pid.

Parameters

- uid: Number containing the Unix user id

Return values:

1. Boolean indicating successful operation
2. String containing the error message if failed
3. Number containing the error code if failed

---

signal (pid, sig)

Send a signal to a process identified by given pid.

Parameters

- pid: Number containing the process id
- sig: Signal to send (default: 15 [SIGTERM])

Return values:

1. Boolean indicating successful operation
2. Number containing the error code if failed

Class *luci.sys.user*

LuCI system utilities / user related functions.

Functions

getuser (uid)	Retrieve user information for given uid.
checkpasswd (username, pass)	Test whether given string matches the password of a given system user.
getpasswd (username)	Retrieve the current user password hash.
setpasswd (username, password)	Change the password of given user.

Functions

---

getuser (uid)

Retrieve user information for given uid.

Parameters

- uid: Number containing the Unix user id

Return value:

Table containing the following fields: { "uid", "gid", "name", "passwd", "dir", "shell", "gecos" }

---

checkpasswd (username, pass)

Test whether given string matches the password of a given system user.

Parameters

- username: String containing the Unix user name
- pass: String containing the password to compare

Return value:

Boolean indicating whether the passwords are equal

---

getpasswd (username)

Retrieve the current user password hash.

Parameters

- username: String containing the username to retrieve the password for

Return values:

1. String containing the hash or nil if no password is set.
2. Password database entry

---

setpasswd (username, password)

Change the password of given user.

Parameters

- username: String containing the Unix user name
- password: String containing the password to compare

Return value:

Number containing 0 on success and >= 1 on error

Class *luci.sys.wifi*

LuCI system utilities / wifi related functions.

Functions

getiwinfo (ifname)	Get wireless information for given interface.

Functions

---

getiwinfo (ifname)

Get wireless information for given interface.

Parameters

- ifname: String containing the interface name

Return value:

A wrapped iwinfo object instance

Class *luci.util*

Functions

append (src, ...)	Appends numerically indexed tables or single objects to a given table.
bigendian ()	Test whether the current system is operating in big endian mode.
class (base)	Create a Class object (Python-style object model).
clone (object, deep)	Clones the given object and return it's copy.
cmatch (str, pattern)	Count the occurrences of given substring in given string.
combine (tbl1, tbl2, ...)	Combines two or more numerically indexed tables and single objects into one table.
contains (table, value)	Checks whether the given table contains the given value.
copcall (f, ...)	This is a coroutine-safe drop-in replacement for Lua's "pcall"-function
coxpcall (f, err, ...)	This is a coroutine-safe drop-in replacement for Lua's "xpcall"-function
dumptable (t, maxdepth)	Recursively dumps a table to stdout, useful for testing and debugging.
exec (command)	Execute given commandline and gather stdout.

execi (command)	Return a line-buffered iterator over the output of given command.
get_bytecode (val)	Return the current runtime bytecode of the given data.
imatch (val)	Return a matching iterator for the given value.
instanceof (object, class)	Test whether the given object is an instance of the given class.
keys (t)	Retrieve all keys of given associative table.
kspairs (t)	Return a key, value iterator for the given table.
libpath ()	Returns the absolute path to LuCI base directory.
parse_units (ustr)	Parse certain units from the given string and return the canonical integer value or 0 if the unit is unknown.
pcdata (value)	Create valid XML PCDATA from given string.
perror (obj)	Write given object to stderr.
restore_data (str)	Restore data previously serialized with serialize_data().
serialize_data (val)	Recursively serialize given data to lua code, suitable for restoring with loadstring().
serialize_json (data, writer)	Convert data structure to JSON
shellquote (value)	Safely quote value for use in shell commands.
spairs (t, f)	Return a key, value iterator which returns the values sorted according to the provided callback function.
split (str, pat, max, regex)	Splits given string on a defined separator sequence and return a table containing the resulting substrings.
strip_bytecode (code)	Strips unnecessary lua bytecode from given string.
striptags (value)	Strip HTML tags from given string.
threadlocal ()	Create a new or get an already existing thread local store associated with the current active coroutine.
trim (str)	Remove leading and trailing whitespace from given string value.
ubus (object, method, values)	Issue an ubus call.
update (t, updates)	Update values in given table with the values from the second given table.
urldecode (str, decode_plus)	Decode an URL-encoded string - optionally decoding the "+" sign to space.
urlencode (str)	URL-encode given string.
vspairs (t)	Return a key, value iterator for the given table.

Functions

---

append (src, ...)

Appends numerically indexed tables or single objects to a given table.

Parameters

- src: Target table
- ...: Objects to insert

Return value:

Target table

---

bigendian ()

Test whether the current system is operating in big endian mode.

Return value:

Boolean value indicating whether system is big endian

---

class (base)

Create a Class object (Python-style object model). The class object can be instantiated by calling itself. Any class functions or shared parameters can be attached to this object. Attaching a table to the class object makes this table shared between all instances of this class. For object parameters use the __init__ function. Classes can inherit member functions and values from a base class. Class can be instantiated by calling them. All parameters will be passed to the __init__ function of this class - if such a function exists. The __init__ function must be used to set any object parameters that are not shared with other objects of this class. Any return values will be ignored.

Parameters

- base: The base class to inherit from (optional)

Return value:

A class object

See also:

- instanceof
- clone

---

clone (object, deep)

Clones the given object and return it's copy.

Parameters

- object: Table value to clone
- deep: Boolean indicating whether to do recursive cloning

Return value:

Cloned table value

---

cmatch (str, pattern)

Count the occurrences of given substring in given string.

Parameters

- str: String to search in
- pattern: String containing pattern to find

Return value:

Number of found occurrences

---

combine (tbl1, tbl2, ...)

Combines two or more numerically indexed tables and single objects into one table.

Parameters

- tbl1: Table value to combine
- tbl2: Table value to combine
- ...: More tables to combine

Return value:

Table value containing all values of given tables

---

contains (table, value)

Checks whether the given table contains the given value.

Parameters

- table: Table value
- value: Value to search within the given table

Return value:

Number indicating the first index at which the given value occurs within table or false.

---

copcall (f, ...)

This is a coroutine-safe drop-in replacement for Lua's "pcall"-function

Parameters

- f: Lua function to be called protected
- ...: Parameters passed to the function

Return value:

A boolean whether the function call succeeded and the returns values of the function or the error object

---

coxpcall (f, err, ...)

This is a coroutine-safe drop-in replacement for Lua's "xpcall"-function

Parameters

- f: Lua function to be called protected
- err: Custom error handler
- ...: Parameters passed to the function

Return value:

A boolean whether the function call succeeded and the return values of either the function or the error handler

---

dumptable (t, maxdepth)

Recursively dumps a table to stdout, useful for testing and debugging.

Parameters

- t: Table value to dump
- maxdepth: Maximum depth

Return value:

Always nil

---

exec (command)

Execute given commandline and gather stdout.

Parameters

- command: String containing command to execute

Return value:

String containing the command's stdout

---

execi (command)

Return a line-buffered iterator over the output of given command.

Parameters

- command: String containing the command to execute

Return value:

Iterator

---

get_bytecode (val)

Return the current runtime bytecode of the given data. The byte code will be stripped before it is returned.

Parameters

- val: Value to return as bytecode

Return value:

String value containing the bytecode of the given data

---

imatch (val)

Return a matching iterator for the given value. The iterator will return one token per invocation, the tokens are separated by whitespace. If the input value is a table, it is transformed into a string first. A nil value will result in a valid iterator which aborts with the first invocation.

Parameters

- val: The value to scan (table, string or nil)

Return value:

Iterator which returns one token per call

---

instanceof (object, class)

Test whether the given object is an instance of the given class.

Parameters

- object: Object instance
- class: Class object to test against

Return value:

Boolean indicating whether the object is an instance

See also:

- class
- clone

---

keys (t)

Retrieve all keys of given associative table.

Parameters

- t: Table to extract keys from

Return value:

Sorted table containing the keys

---

kspairs (t)

Return a key, value iterator for the given table. The table pairs are sorted by key.

Parameters

- t: The table to iterate

Return value:

Function value containing the corresponding iterator

---

libpath ()

Returns the absolute path to LuCI base directory.

Return value:

String containing the directory path

---

parse_units (ustr)

Parse certain units from the given string and return the canonical integer value or 0 if the unit is unknown. Upper- or lower case is irrelevant. Recognized units are: o "y" - one year (60*60*24*366) o "m" - one month (60*60*24*31) o "w" - one week (60*60*24*7) o "d" - one day (60*60*24) o "h" - one hour (60*60) o "min" - one minute (60) o "kb" - one kilobyte (1024) o "mb" - one megabyte (1024*1024) o "gb" - one gigabyte (1024*1024*1024) o "kib" - one si kilobyte (1000) o "mib" - one si megabyte (1000*1000) o "gib" - one si gigabyte (1000*1000*1000)

Parameters

- ustr: String containing a numerical value with trailing unit

Return value:

Number containing the canonical value

---

pcdata (value)

Create valid XML PCDATA from given string.

Parameters

- value: String value containing the data to escape

Return value:

String value containing the escaped data

---

perror (obj)

Write given object to stderr.

Parameters

- obj: Value to write to stderr

Return value:

Boolean indicating whether the write operation was successful

---

restore_data (str)

Restore data previously serialized with serialize_data().

Parameters

- str: String containing the data to restore

Return value:

Value containing the restored data structure

See also:

- serialize_data
- get_bytecode

---

serialize_data (val)

Recursively serialize given data to lua code, suitable for restoring with loadstring().

Parameters

- val: Value containing the data to serialize

Return value:

String value containing the serialized code

See also:

- restore_data
- get_bytecode

---

serialize_json (data, writer)

Convert data structure to JSON

Parameters

- data: The data to serialize
- writer: A function to write a chunk of JSON data (optional)

Return value:

String containing the JSON if called without write callback

---

shellquote (value)

Safely quote value for use in shell commands.

Parameters

- value: String containing the value to quote

Return value:

Single-quote enclosed string with embedded quotes escaped

---

spairs (t, f)

Return a key, value iterator which returns the values sorted according to the provided callback function.

Parameters

- t: The table to iterate
- f: A callback function to decide the order of elements

Return value:

Function value containing the corresponding iterator

---

split (str, pat, max, regex)

Splits given string on a defined separator sequence and return a table containing the resulting substrings. The optional max parameter specifies the number of bytes to process, regardless of the actual length of the given string. The optional last parameter, regex, specifies whether the separator sequence is nterpreted as regular expression.

Parameters

- str: String value containing the data to split up
- pat: String with separator pattern (optional, defaults to "\n")
- max: Maximum times to split (optional)
- regex: Boolean indicating whether to interpret the separator pattern as regular expression (optional, default is false)

Return value:

Table containing the resulting substrings

---

strip_bytecode (code)

Strips unnecessary lua bytecode from given string. Information like line numbers and debugging numbers will be discarded. Original version by Peter Cawley (http://lua-users.org/lists/lua-l/2008-02/msg01158.html)

Parameters

- code: String value containing the original lua byte code

Return value:

String value containing the stripped lua byte code

---

striptags (value)

Strip HTML tags from given string.

Parameters

- value: String containing the HTML text

Return value:

String with HTML tags stripped of

---

threadlocal ()

Create a new or get an already existing thread local store associated with the current active coroutine. A thread local store is private a table object whose values can't be accessed from outside of the running coroutine.

Return value:

Table value representing the corresponding thread local store

---

trim (str)

Remove leading and trailing whitespace from given string value.

Parameters

- str: String value containing whitespace padded data

Return value:

String value with leading and trailing space removed

---

ubus (object, method, values)

Issue an ubus call.

Parameters

- object: String containing the ubus object to call
- method: String containing the ubus method to call
- values: Table containing the values to pass

Return value:

Table containin the ubus result

---

update (t, updates)

Update values in given table with the values from the second given table. Both table are - in fact - merged together.

Parameters

- t: Table which should be updated
- updates: Table containing the values to update

Return value:

Always nil

---

urldecode (str, decode_plus)

Decode an URL-encoded string - optionally decoding the "+" sign to space.

Parameters

- str: Input string in x-www-urlencoded format
- decode_plus: Decode "+" signs to spaces if true (optional)

Return value:

The decoded string

See also:

- urlencode

---

urlencode (str)

URL-encode given string.

Parameters

- str: String to encode

Return value:

String containing the encoded data

See also:

- urldecode

---

vspairs (t)

Return a key, value iterator for the given table. The table pairs are sorted by value.

Parameters

- t: The table to iterate

Return value:

Function value containing the corresponding iterator

Class *nixio*

General POSIX IO library.

Functions

bind (host, port, family, socktype)	Create a new socket and bind it to a network address.
chdir (path)	Change the working directory.
closelog ()	(POSIX) Close the connection to the system logger.
connect (host, port, family, socktype)	Create a new socket and connect to a network address.
crypt (key, salt)	(POSIX) Encrypt a user password.
dup (oldfd, newfd)	Duplicate a file descriptor.
errno ()	Get the last system error code.
exec (executable, ...)	Execute a file to replace the current process.
exece (executable, arguments, environment)	Execute a file with a custom environment to replace the current process.
execp (executable, ...)	Invoke the shell and execute a file to replace the current process.
fork ()	(POSIX) Clone the current process.
getaddrinfo (host, family, service)	Look up a hostname and service via DNS.
getcwd ()	Get the current working directory.
getenv (variable)	Get the current environment table or a specific environment variable.
getgid ()	(POSIX) Get the group id of the current process.
getgr (group)	(POSIX) Get all or a specific user group.
getifaddrs ()	(Linux, BSD) Get a list of available network interfaces and their addresses.
getnameinfo (ipaddr)	Reverse look up an IP-Address via DNS.
getpid ()	Get the ID of the current process.
getppid ()	(POSIX) Get the parent process id of the current process.
getproto (proto)	Get all or a specific proto entry.
getprotobyname (name)	Get protocol entry by name.

getprotobynumber (proto)	Get protocol entry by number.
getpw (user)	(POSIX) Get all or a specific user account.
getsp (user)	(Linux, Solaris) Get all or a specific shadow password entry.
getuid ()	(POSIX) Get the user id of the current process.
kill (target, signal)	(POSIX) Send a signal to one or more processes.
nanosleep (seconds, nanoseconds)	Sleep for a specified amount of time.
nice (nice)	(POSIX) Change priority of current process.
open (path, flags, mode)	Open a file.
open_flags (flag1, ...)	Generate flags for a call to open().
openlog (ident, flag1, ...)	(POSIX) Open a connection to the system logger.
pipe ()	Create a pipe.
poll (fds, timeout)	Wait for some event on a file descriptor.
poll_flags (mode1, ...)	Generate events-bitfield or parse revents-bitfield for poll.
sendfile (socket, file, length)	(POSIX) Send data from a file to a socket in kernel-space.
setenv (variable, value)	Set or unset a environment variable.
setgid (gid)	(POSIX) Set the group id of the current process.
setlogmask (priority)	(POSIX) Set the logmask of the system logger for current process.
setsid ()	(POSIX) Create a new session and set the process group ID.
setuid (gid)	(POSIX) Set the user id of the current process.
signal (signal, handler)	Ignore or use set the default handler for a signal.
socket (domain, type)	Create a new socket.
splice (fdin, fdout, length, flags)	(Linux) Send data from / to a pipe in kernel-space.

splice_flags (flag1, ...)	(Linux) Generate a flag bitfield for a call to splice.
strerror (errno)	Get the error message for the corresponding error code.
sysinfo ()	(Linux) Get overall system statistics.
syslog (priority)	(POSIX) Write a message to the system logger.
times ()	(POSIX) Get process times.
tls (mode)	Create a new TLS context.
umask (mask)	Sets the file mode creation mask.
uname ()	(POSIX) Get information about current system and kernel.
waitpid (pid, flag1, ...)	(POSIX) Wait for a process to change state.

Functions

---

bind (host, port, family, socktype)

Create a new socket and bind it to a network address. This function is a shortcut for calling nixio.socket and then bind() on the socket object.

Parameters

- host: Hostname or IP-Address (optional, default: all addresses)
- port: Port or service description
- family: Address family ["**any**", "inet", "inet6"]
- socktype: Socket Type ["**stream**", "dgram"]

Usage

- This functions calls getaddrinfo(), socket(), setsockopt() and bind() but NOT listen().
- The reuseaddr-option is automatically set before binding.

Return value:

Socket Object

---

chdir (path)

Change the working directory.

Parameters

- path: New working directory

Return value:

true

---

closelog ()

(POSIX) Close the connection to the system logger.

---

connect (host, port, family, socktype)

Create a new socket and connect to a network address. This function is a shortcut for calling nixio.socket and then connect() on the socket object.

Parameters

- host: Hostname or IP-Address (optional, default: localhost)
- port: Port or service description
- family: Address family ["**any**", "inet", "inet6"]
- socktype: Socket Type ["**stream**", "dgram"]

Usage:

This functions calls getaddrinfo(), socket() and connect().

Return value:

Socket Object

---

crypt (key, salt)

(POSIX) Encrypt a user password.

Parameters

- key: Key
- salt: Salt

Return value:

password hash

---

dup (oldfd, newfd)

Duplicate a file descriptor.

Parameters

- oldfd: Old descriptor [File Object, Socket Object (POSIX only)]
- newfd: New descriptor to serve as copy (optional)

Usage:

This funcation calls dup2() if newfd is set, otherwise dup().

Return value:

File Object of new descriptor

---

errno ()

Get the last system error code.

Return value:

Error code

---

exec (executable, ...)

Execute a file to replace the current process.

Parameters

- executable: Executable
- ...: Parameters

Usage

- The name of the executable is automatically passed as argv[0]
- This function does not return on success.

---

exece (executable, arguments, environment)

Execute a file with a custom environment to replace the current process.

Parameters

- executable: Executable
- arguments: Argument Table
- environment: Environment Table (optional)

Usage

- The name of the executable is automatically passed as argv[0]
- This function does not return on success.

---

execp (executable, ...)

Invoke the shell and execute a file to replace the current process.

Parameters

- executable: Executable
- ...: Parameters

Usage

- The name of the executable is automatically passed as argv[0]
- This function does not return on success.

---

fork ()

(POSIX) Clone the current process.

Return value:

the child process id for the parent process, 0 for the child process

---

getaddrinfo (host, family, service)

Look up a hostname and service via DNS.

Parameters

- host: hostname to lookup (optional)
- family: address family ["**any**", "inet", "inet6"]
- service: service name or port (optional)

Return value:

Table containing one or more tables containing:

- family = ["inet", "inet6"]
- socktype = ["stream", "dgram", "raw"]
- address = Resolved IP-Address
- port = Resolved Port (if service was given)

---

getcwd ()

Get the current working directory.

Return value:

workign directory

---

getenv (variable)

Get the current environment table or a specific environment variable.

Parameters

- variable: Variable (optional)

Return value:

environment table or single environment variable

---

getgid ()

(POSIX) Get the group id of the current process.

Return value:

process group id

---

getgr (group)

(POSIX) Get all or a specific user group.

Parameters

- group: Group ID or groupname (optional)

Return value:

Table containing:

- name = Group Name
- gid = Group ID
- passwd = Password
- mem = {Member #1, Member #2, ...}

---

getifaddrs ()

(Linux, BSD) Get a list of available network interfaces and their addresses.

Return value:

Table containing one or more tables containing:

- name = Interface Name
- family = ["inet", "inet6", "packet"]
- addr = Interface Address (IPv4, IPv6, MAC, ...)
- broadaddr = Broadcast Address
- dstaddr = Destination Address (Point-to-Point)
- netmask = Netmask (if available)
- prefix = Prefix (if available)
- flags = Table of interface flags (up, multicast, loopback, ...)
- data = Statistics (Linux, "packet"-family)
- hatype = Hardware Type Identifier (Linix, "packet"-family)
- ifindex = Interface Index (Linux, "packet"-family)

---

getnameinfo (ipaddr)

Reverse look up an IP-Address via DNS.

Parameters

- ipaddr: IPv4 or IPv6-Address

Return value:

FQDN

---

getpid ()

Get the ID of the current process.

Return value:

process id

---

getppid ()

(POSIX) Get the parent process id of the current process.

Return value:

parent process id

---

getproto (proto)

Get all or a specific proto entry.

Parameters

- proto: protocol number or name to lookup (optional)

Return value:

Table (or if no parameter is given, a table of tables) containing the following fields:

- name = Protocol Name
- proto = Protocol Number
- aliases = Table of alias names

---

getprotobyname (name)

Get protocol entry by name.

Parameters

- name: protocol name to lookup

Usage:

This function returns nil if the given protocol is unknown.

Return value:

Table containing the following fields:

- name = Protocol Name
- proto = Protocol Number
- aliases = Table of alias names

---

getprotobynumber (proto)

Get protocol entry by number.

Parameters

- proto: protocol number to lookup

Usage:

This function returns nil if the given protocol is unknown.

Return value:

Table containing the following fields:

- name = Protocol Name
- proto = Protocol Number
- aliases = Table of alias names

---

getpw (user)

(POSIX) Get all or a specific user account.

Parameters

- user: User ID or username (optional)

Return value:

Table containing:

- name = Name
- uid = ID
- gid = Group ID
- passwd = Password
- dir = Home directory
- gecos = Information
- shell = Shell

---

getsp (user)

(Linux, Solaris) Get all or a specific shadow password entry.

Parameters

- user: username (optional)

Return value:

Table containing:

- namp = Name
- expire = Expiration Date
- flag = Flags
- inact = Inactivity Date
- lstchg = Last change
- max = Maximum
- min = Minimum
- warn = Warning
- pwdp = Password Hash

---

getuid ()

(POSIX) Get the user id of the current process.

Return value:

process user id

---

kill (target, signal)

(POSIX) Send a signal to one or more processes.

Parameters

- target: Target process of process group.
- signal: Signal to send

Return value:

true

---

nanosleep (seconds, nanoseconds)

Sleep for a specified amount of time.

Parameters

- seconds: Seconds to wait (optional)
- nanoseconds: Nanoseconds to wait (optional)

Usage

- Not all systems support nanosecond precision but you can expect to have at least maillisecond precision.
- This function is not signal-protected and may fail with EINTR.

Return value:

true

---

nice (nice)

(POSIX) Change priority of current process.

Parameters

- nice: Nice Value

Return value:

true

---

open (path, flags, mode)

Open a file.

Parameters

- path: Filesystem path to open
- flags: Flag string or number (see open_flags). ["**r**", "r+", "w", "w+", "a", "a+"]
- mode: File mode for newly created files (see chmod, umask).

Usage:

Although this function also supports the traditional fopen() file flags it does not create a file stream but uses the open() syscall.

Return value:

File Object

See also:

- umask
- open_flags

---

open_flags (flag1, ...)

Generate flags for a call to open().

Parameters

- flag1: First Flag ["append", "creat", "excl", "nonblock", "ndelay", "sync", "trunc", "rdonly", "wronly", "rdwr"]
- ...: More Flags [-"-]

Usage

- This function cannot fail and will never return nil.
- The "nonblock" and "ndelay" flags are aliases.
- The "nonblock", "ndelay" and "sync" flags are no-ops on Windows.

Return value:

flag to be used as second parameter to open

---

openlog (ident, flag1, ...)

(POSIX) Open a connection to the system logger.

Parameters

- ident: Identifier
- flag1: Flag 1 ["cons", "nowait", "pid", "perror", "ndelay", "odelay"]
- ...: More flags [-"-]

---

pipe ()

Create a pipe.

Return values:

1. File Object of the read end
2. File Object of the write end

---

poll (fds, timeout)

Wait for some event on a file descriptor. poll() sets the revents-field of the tables provided by fds to a bitfield indicating the events that occurred.

Parameters

- fds: Table containing one or more tables containing
    - fd = I/O Descriptor [Socket Object, File Object (POSIX)]
    - events = events to wait for (bitfield generated with poll_flags)
- timeout: Timeout in milliseconds

Usage

- This function works in-place on the provided table and only writes the revents field, you can use other fields on your demand.
- All metamethods on the tables provided as fds are ignored.
- The revents-fields are not reset when the call times out. You have to check the first return value to be 0 to handle this case.
- If you want to wait on a TLS-Socket you have to use the underlying socket instead.
- On Windows poll is emulated through select(), can only be used on socket descriptors and cannot take more than 64 descriptors per call.
- This function is not signal-protected and may fail with EINTR.

Return values:

1. number of ready IO descriptors
2. the fds-table with revents-fields set

See also:

- poll_flags

---

poll_flags (mode1, ...)

Generate events-bitfield or parse revents-bitfield for poll.

Parameters

- mode1: revents-Flag bitfield returned from poll to parse OR ["in", "out", "err", "pri" (POSIX), "hup" (POSIX), "nval" (POSIX)]
- ...: More mode strings for generating the flag [-"-]

Return value:

table with boolean fields reflecting the mode parameter OR bitfield to use for the events-Flag field

See also:

- poll

sendfile (socket, file, length)

(POSIX) Send data from a file to a socket in kernel-space.

Parameters

- socket: Socket Object
- file: File Object
- length: Amount of data to send (in Bytes).

Return value:

bytes sent

---

setenv (variable, value)

Set or unset a environment variable.

Parameters

- variable: Variable
- value: Value (optional)

Usage:

The environment variable will be unset if value is omitted.

Return value:

true

---

setgid (gid)

(POSIX) Set the group id of the current process.

Parameters

- gid: New Group ID

Return value:

true

---

setlogmask (priority)

(POSIX) Set the logmask of the system logger for current process.

Parameters

- priority: Priority ["emerg", "alert", "crit", "err", "warning", "notice", "info", "debug"]

---

setsid ()

(POSIX) Create a new session and set the process group ID.

Return value:

session id

---

setuid (gid)

(POSIX) Set the user id of the current process.

Parameters

- gid: New User ID

Return value:

true

---

signal (signal, handler)

Ignore or use set the default handler for a signal.

Parameters

- signal: Signal
- handler: ["ign", "dfl"]

Return value:

true

---

socket (domain, type)

Create a new socket.

Parameters

- domain: Domain ["inet", "inet6", "unix"]
- type: Type ["stream", "dgram", "raw"]

Return value:

Socket Object

---

splice (fdin, fdout, length, flags)

(Linux) Send data from / to a pipe in kernel-space.

Parameters

- fdin: Input I/O descriptor
- fdout: Output I/O descriptor
- length: Amount of data to send (in Bytes).
- flags: (optional, bitfield generated by splice_flags)

Return value:

bytes sent

See also:

- splice_flags

---

splice_flags (flag1, ...)

(Linux) Generate a flag bitfield for a call to splice.

Parameters

- flag1: First Flag ["move", "nonblock", "more"]
- ...: More flags [-"-]

Return value:

Flag bitfield

See also:

- splice

---

strerror (errno)

Get the error message for the corresponding error code.

Parameters

- errno: System error code

Return value:

Error message

---

sysinfo ()

(Linux) Get overall system statistics.

Return value:

Table containing:

- uptime = system uptime in seconds
- loads = {loadavg1, loadavg5, loadavg15}
- totalram = total RAM
- freeram = free RAM
- sharedram = shared RAM
- bufferram = buffered RAM
- totalswap = total SWAP
- freeswap = free SWAP
- procs = number of running processes

---

syslog (priority)

(POSIX) Write a message to the system logger.

Parameters

- priority: Priority ["emerg", "alert", "crit", "err", "warning", "notice", "info", "debug"]

---

times ()

(POSIX) Get process times.

Return value:

Table containing:

- utime = user time
- utime = system time
- cutime = children user time
- cstime = children system time

---

tls (mode)

Create a new TLS context.

Parameters

- mode: TLS-Mode ["client", "server"]

Return value:

TLSContext Object

---

umask (mask)

Sets the file mode creation mask.

Parameters

- mask: New creation mask (see chmod for format specifications)

Return values:

1. the old umask as decimal mode number
2. the old umask as mode string

---

uname ()

(POSIX) Get information about current system and kernel.

Return value:

Table containing:

- sysname = operating system
- nodename = network name (usually hostname)
- release = OS release
- version = OS version
- machine = hardware identifier

---

waitpid (pid, flag1, ...)

(POSIX) Wait for a process to change state.

Parameters

- pid: Process ID (optional, default: any childprocess)
- flag1: Flag (optional) ["nohang", "untraced", "continued"]

- ...: More Flags [-"-]

Usage:

If the "nohang" is given this function becomes non-blocking.

Return values:

1. process id of child or 0 if no child has changed state
2. ["exited", "signaled", "stopped"]
3. [exit code, terminate signal, stop signal]

Class nixio.CHANGELOG

Changes and improvements.

Tables

0.2	Initial Release.
0.3	Service Release.

Tables

0.2

Initial Release.

- Initial Release

0.3

Service Release.

- Added getifaddrs() function.
- Added getsockopt(), setsockopt(), getsockname() and getpeername() directly to TLS-socket objects unifying the socket interface.
- Added support for CyaSSL as cryptographical backend.
- Added support for x509 certificates in DER format.
- Added support for splice() in UnifiedIO.copyz().
- Added interface to inject chunks into UnifiedIO.linesource() buffer.
- Changed TLS behaviour to explicitly separate servers and clients.
- Fixed usage of signed datatype breaking Base64 decoding.
- Fixed namespace clashes for nixio.fs.
- Fixed splice() support for some exotic C libraries.

- Reconfigure axTLS cryptographical provider and mark it as obsolete.

Object Instance *nixio.CryptoHash*

Cryptographical Hash and HMAC object.

Functions

CryptoHash:final ()	Finalize the hash and return the digest.
CryptoHash:update (chunk)	Add another chunk of data to be hashed.

Functions

CryptoHash:final ()

Finalize the hash and return the digest.

Usage:

You cannot call update on a hash object that was already finalized you can however call final multiple times to get the digest.

Return values:

1. hexdigest
2. buffer containing binary digest

CryptoHash:update (chunk)

Add another chunk of data to be hashed.

Parameters

- chunk: Chunk of data

Return value:

CryptoHash object (self)

Object Instance *nixio.File*

Large File Object. Large file operations are supported up to 52 bits if the Lua number type is double (default).

Functions

File:close ()	Close the file descriptor.

File:fileno ()	Get the number of the filedescriptor.
File:lock (command, length)	Apply or test a lock on the file.
File:read (length)	Read from a file descriptor.
File:seek (offset, whence)	Reposition read / write offset of the file descriptor.
File:setblocking (blocking)	(POSIX) Set the blocking mode of the file descriptor.
File:stat (field)	Get file status and attributes.
File:sync (data_only)	Synchronizes the file with the storage device.
File:tell ()	Return the current read / write offset of the file descriptor.
File:write (buffer, offset, length)	Write to the file descriptor.

Functions

File:close ()

Close the file descriptor.

Return value:

true

File:fileno ()

Get the number of the filedescriptor.

Return value:

file descriptor number

File:lock (command, length)

Apply or test a lock on the file.

Parameters

- command: Locking Command ["lock", "tlock", "ulock", "test"]
- length: Amount of Bytes to lock from current offset (optional)

Usage

- This function calls lockf() on POSIX and _locking() on Windows.
- The "lock" command is blocking, "tlock" is non-blocking, "ulock" unlocks and "test" only tests for the lock.
- The "test" command is not available on Windows.
- Locks are by default advisory on POSIX, but mandatory on Windows.

Return value:

true

File:read (length)

Read from a file descriptor.

Parameters

- length: Amount of data to read (in Bytes).

Usage

- **Warning:** It is not guaranteed that all requested data is read at once especially when dealing with pipes. You have to check the return value - the length of the buffer actually read - or use the safe IO functions in the high-level IO utility module.
- The length of the return buffer is limited by the (compile time) nixio buffersize which is nixio.const.buffersize (8192 by default). Any read request greater than that will be safely truncated to this value.

Return value:

buffer containing data successfully read

File:seek (offset, whence)

Reposition read / write offset of the file descriptor. The seek will be done either from the beginning of the file or relative to the current position or relative to the end.

Parameters

- offset: File Offset
- whence: Starting point ["set", "cur", "end"]

Usage:

This function calls lseek().

Return value:

new (absolute) offset position

File:setblocking (blocking)

(POSIX) Set the blocking mode of the file descriptor.

Parameters

- blocking: (boolean)

Return value:

true

File:stat (field)

Get file status and attributes.

Parameters

- field: Only return a specific field, not the whole table (optional)

Usage:

This function calls fstat().

Return value:

Table containing:

- atime = Last access timestamp
- blksize = Blocksize (POSIX only)
- blocks = Blocks used (POSIX only)
- ctime = Creation timestamp
- dev = Device ID
- gid = Group ID
- ino = Inode
- modedec = Mode converted into a decimal number
- modestr = Mode as string as returned by `ls -l`
- mtime = Last modification timestamp
- nlink = Number of links
- rdev = Device ID (if special file)
- size = Size in bytes
- type = ["reg", "dir", "chr", "blk", "fifo", "lnk", "sock"]
- uid = User ID

File:sync (data_only)

Synchronizes the file with the storage device. Returns when the file is successfully written to the disk.

Parameters

- data_only: Do not synchronize the metadata. (optional, boolean)

Usage

- This function calls fsync() when data_only equals false otherwise fdatasync(), on Windows _commit() is used instead.
- fdatasync() is only supported by Linux and Solaris. For other systems the data_only parameter is ignored and fsync() is always called.

Return value:

true

File:tell ()

Return the current read / write offset of the file descriptor.

Usage:

This function calls lseek() with offset 0 from the current position.

Return value:

offset position

File:write (buffer, offset, length)

Write to the file descriptor.

Parameters

- buffer: Buffer holding the data to be written.
- offset: Offset to start reading the buffer from. (optional)
- length: Length of chunk to read from the buffer. (optional)

Usage

- Warning: It is not guaranteed that all data in the buffer is written at once especially when dealing with pipes. You have to check the return value - the number of bytes actually written - or use the safe IO functions in the high-level IO utility module.
- Unlike standard Lua indexing the lowest offset and default is 0.

Return value:

number of bytes written

Class nixio.README

General Information.

Tables

Errorhandling	General error handling information.
Functions	Function conventions.
Platforms	Platform information.
TLS-Crypto	Cryptography and TLS libraries.

Tables

Errorhandling

General error handling information.

- Most of the functions available in this library may fail. If any error occurs the function returns **nil or false**, an error code (usually errno) and an additional error message text (if avaialable).
- At the moment false is only returned when a non-blocking I/O function fails with EAGAIN, EWOULDBLOCK or WSAEWOULDBLOCK for any others nil is returned as first parameter. Therefore you can use false to write portable non-blocking I/O applications.
- Note that the function documentation does only mention the return values in case of a successful operation.
- You can find a table of common error numbers and other useful constants like signal numbers in **nixio.const** e.g. nixio.const.EINVAL, nixio.const.SIGTERM, etc. For portability there is a second error constant table **nixio.const_sock** for socket error codes. This might be important if you are dealing with Windows applications, on POSIX however const_sock is just an alias for const.
- With some exceptions - which are explicitly stated in the function documentation - all blocking functions are signal-protected and will not fail with EINTR.
- On POSIX the SIGPIPE signal will be set to ignore upon initialization. You should restore the default behaviour or set a custom signal handler in your program after loading nixio if you need this behaviour.

Functions

Function conventions.

In general all functions are namend and behave like their POSIX API counterparts - where applicable - applying the following rules:

- Functions should be named like the underlying POSIX API function omitting prefixes or suffixes - especially when placed in an object-context (lockf -> File:lock, fsync -> File:sync, dup2 -> dup, ...)
- If you are unclear about the behaviour of a function you should consult your OS API documentation (e.g. the manpages).
- If the name is significantly different from the POSIX-function, the underlying function(s) are stated in the documentation.
- Parameters should reflect those of the C-API, buffer length arguments and by-reference parameters should be omitted for practical purposes.
- If a C function accepts a bitfield as parameter, it should be translated into lower case string flags representing the flags if the bitfield is the last parameter and also omitting prefixes or

suffixes. (e.g. waitpid (pid, &s, WNOHANG | WUNTRACED) -> waitpid(pid, "nohang", "untraced"), getsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt)) -> Socket:getopt("socket", "reuseaddr"), etc.)

- If it is not applicable to provide a string representation of the bitfield a bitfield generator helper is provided. It is named FUNCTION_flags. (open("/tmp/test", O_RDONLY | O_NONBLOCK) -> open("/tmp/test", open_flags("rdonly", "nonblock")))

Platforms

Platform information.

- The minimum platform requirements are a decent POSIX 2001 support. Builds are more or less tested on Linux, Solaris and FreeBSD. Builds for Windows XP SP1 and later can be compiled with MinGW either from Windows itself or using the MinGW cross-compiler. Earlier versions of Windows are not supported.
- In general all functions which don't have any remarks in their documentation are available on all platforms.
- Functions with a (POSIX), (Linux) or similar prefix are only available on these specific platforms. Same appplies to parameters of functions with a similar suffix.
- Some functions might have limitations on some platforms. This should be stated in the documentation. Please also consult your OS API documentation.

TLS-Crypto

Cryptography and TLS libraries.

- Currently 3 underlying cryptography libraries are supported: openssl, cyassl and axTLS. The name of the library in use is written to **nixio.tls_provider**
- You should whenever possible use openssl or cyassl as axTLS has only limited support. It does not provide support for non-blocking sockets and is probably less audited than the other ones.
- As the supported Windows versions are not suitable for embedded devices axTLS is at the moment not supported on Windows.

Object Instance *nixio.Socket*

Socket Object. Supports IPv4, IPv6 and UNIX (POSIX only) families.

Functions

Socket:accept ()	Accept a connection on the socket.
Socket:bind (host, port)	Bind the socket to a network address.
Socket:close ()	Close the socket.
Socket:connect (host, port)	Connect the socket to a network address.
Socket:fileno ()	Get the number of the filedescriptor.
Socket:getopt (level, option)	Get a socket option.
Socket:getpeername ()	Get the peer address of a socket.
Socket:getsockname ()	Get the local address of a socket.
Socket:listen (backlog)	Listen for connections on the socket.
Socket:read (length)	Receive a message on the socket (This is an alias for recv).
Socket:recv (length)	Receive a message on the socket.
Socket:recvfrom (length)	Receive a message on the socket including the senders source address.
Socket:send (buffer, offset, length)	Send a message on the socket.
Socket:sendto (buffer, host, port, offset, length)	Send a message on the socket specifying the destination.
Socket:setblocking (blocking)	Set the blocking mode of the socket.
Socket:setopt (level, option, value)	Set a socket option.
Socket:shutdown (how)	Shut down part of a full-duplex connection.
Socket:write (buffer, offset, length)	Send a message on the socket (This is an alias for send).

Functions

Socket:accept ()

Accept a connection on the socket.

Return values:

1. Socket Object
2. Peer IP-Address

3. Peer Port

Socket:bind (host, port)

Bind the socket to a network address.

Parameters

- host: Host (optional, default: all addresses)
- port: Port or service description

Usage

- This function calls getaddrinfo() and bind() but NOT listen().
- If host is a domain name it will be looked up and bind() tries the IP-Addresses in the order returned by the DNS resolver until the bind succeeds.
- UNIX sockets ignore the port, and interpret host as a socket path.

Return value:

true

Socket:close ()

Close the socket.

Return value:

true

Socket:connect (host, port)

Connect the socket to a network address.

Parameters

- host: Hostname or IP-Address (optional, default: localhost)
- port: Port or service description

Usage

- This function calls getaddrinfo() and connect().
- If host is a domain name it will be looked up and connect() tries the IP-Addresses in the order returned by the DNS resolver until the connect succeeds.
- UNIX sockets ignore the port, and interpret host as a socket path.

Return value:

true

Socket:fileno ()

Get the number of the filedescriptor.

Return value:

file descriptor number

Socket:getopt (level, option)

Get a socket option.

Parameters

- level: Level ["socket", "tcp", "ip", "ipv6"]
- option: Option ["keepalive", "reuseaddr", "sndbuf", "rcvbuf", "priority", "broadcast", "linger", "sndtimeo", "rcvtimeo", "dontroute", "bindtodevice", "error", "oobinline", "cork" (TCP), "nodelay" (TCP), "mtu" (IP, IPv6), "hdrincl" (IP), "multicast_ttl" (IP), "multicast_loop" (IP, IPv6), "multicast_if" (IP, IPv6), "v6only" (IPv6), "multicast_hops" (IPv6), "add_membership" (IP, IPv6), "drop_membership" (IP, IPv6)]

Return value:

Value

Socket:getpeername ()

Get the peer address of a socket.

Return values:

1. IP-Address
2. Port

Socket:getsockname ()

Get the local address of a socket.

Return values:

1. IP-Address
2. Port

Socket:listen (backlog)

Listen for connections on the socket.

Parameters

- backlog: Length of queue for pending connections

Return value:

true

Socket:read (length)

Receive a message on the socket (This is an alias for recv). See the recvfrom description for more details.

Parameters

- length: Amount of data to read (in Bytes).

Return value:

buffer containing data successfully read

See also:

- Socket:recvfrom

Socket:recv (length)

Receive a message on the socket. This function is identical to recvfrom except that it does not return the sender's source address. See the recvfrom description for more details.

Parameters

- length: Amount of data to read (in Bytes).

Return value:

buffer containing data successfully read

See also:

- Socket:recvfrom

Socket:recvfrom (length)

Receive a message on the socket including the senders source address.

Parameters

- length: Amount of data to read (in Bytes).

Usage

- **Warning:** It is not guaranteed that all requested data is read at once. You have to check the return value - the length of the buffer actually read - or use the safe IO functions in the high-level IO utility module.
- The length of the return buffer is limited by the (compile time) nixio buffersize which is nixio.const.buffersize (8192 by

default). Any read request greater than that will be safely truncated to this value.

Return values:

1. buffer containing data successfully read
2. host IP-Address of the sender
3. port Port of the sender

Socket:send (buffer, offset, length)

Send a message on the socket. This function is identical to sendto except for the missing destination parameters. See the sendto description for a detailed description.

Parameters

- buffer: Buffer holding the data to be written.
- offset: Offset to start reading the buffer from. (optional)
- length: Length of chunk to read from the buffer. (optional)

Return value:

number of bytes written

See also:

- Socket:sendto

Socket:sendto (buffer, host, port, offset, length)

Send a message on the socket specifying the destination.

Parameters

- buffer: Buffer holding the data to be written.
- host: Target IP-Address
- port: Target Port
- offset: Offset to start reading the buffer from. (optional)
- length: Length of chunk to read from the buffer. (optional)

Usage

- **Warning:** It is not guaranteed that all data in the buffer is written at once. You have to check the return value - the number of bytes actually written - or use the safe IO functions in the high-level IO utility module.
- Unlike standard Lua indexing the lowest offset and default is 0.

Return value:

number of bytes written

Socket:setblocking (blocking)

Set the blocking mode of the socket.

Parameters

- blocking: (boolean)

Return value:

true

Socket:setopt (level, option, value)

Set a socket option.

Parameters

- level: Level ["socket", "tcp", "ip", "ipv6"]
- option: Option ["keepalive", "reuseaddr", "sndbuf", "rcvbuf", "priority", "broadcast", "linger", "sndtimeo", "rcvtimeo", "dontroute", "bindtodevice", "error", "oobinline", "cork" (TCP), "nodelay" (TCP), "mtu" (IP, IPv6), "hdrincl" (IP), "multicast_ttl" (IP), "multicast_loop" (IP, IPv6), "multicast_if" (IP, IPv6), "v6only" (IPv6), "multicast_hops" (IPv6), "add_membership" (IP, IPv6), "drop_membership" (IP, IPv6)]
- value: Value

Return value:

true

Socket:shutdown (how)

Shut down part of a full-duplex connection.

Parameters

- how: (optional, default: rdwr) ["rdwr", "rd", "wr"]

Return value:

true

Socket:write (buffer, offset, length)

Send a message on the socket (This is an alias for send). See the sendto description for a detailed description.

Parameters

- buffer: Buffer holding the data to be written.
- offset: Offset to start reading the buffer from. (optional)
- length: Length of chunk to read from the buffer. (optional)

Return value:

number of bytes written

See also:

- Socket:sendto

Object Instance *nixio.TLSContext*

Transport Layer Security Context Object.

Functions

TLSContext:create (socket)	Create a TLS Socket from a socket descriptor.
TLSContext:set_cert (path)	Assign a PEM certificate to this context.
TLSContext:set_ciphers (cipherlist)	Set the available ciphers for this context.
TLSContext:set_key (path)	Assign a PEM private key to this context.
TLSContext:set_verify (flag1, ...)	Set the verification flags of this context.
TLSContext:set_verify_depth (depth)	Set the verification depth of this context.

Functions

TLSContext:create (socket)

Create a TLS Socket from a socket descriptor.

Parameters

- socket: Socket Object

Return value:

TLSSocket Object

TLSContext:set_cert (path)

Assign a PEM certificate to this context.

Parameters

- path: Certificate File path

Usage:

This function calls SSL_CTX_use_certificate_chain_file().

Return value:

true

TLSContext:set_ciphers (cipherlist)

Set the available ciphers for this context.

Parameters

- cipherlist: String containing a list of ciphers

Usage:

This function calls SSL_CTX_set_cipher_list().

Return value:

true

TLSContext:set_key (path)

Assign a PEM private key to this context.

Parameters

- path: Private Key File path

Usage:

This function calls SSL_CTX_use_PrivateKey_file().

Return value:

true

TLSContext:set_verify (flag1, ...)

Set the verification flags of this context.

Parameters

- flag1: First Flag ["none", "peer", "verify_fail_if_no_peer_cert", "client_once"]
- ...: More Flags [-"-]

Usage:

This function calls SSL_CTX_set_verify().

Return value:

true

TLSContext:set_verify_depth (depth)

Set the verification depth of this context.

Parameters

- depth: Depth

Usage:

This function calls SSL_CTX_set_verify_depth().

Return value:

true

Object Instance *nixio.TLSSocket*

TLS Socket Object. TLS Sockets contain the underlying socket and context in the fields "socket" and "context".

Functions

TLSSocket:accept ()	Wait for a TLS handshake from a client.
TLSSocket:connect ()	Initiate the TLS handshake as client with the server.
TLSSocket:read (length)	Receive a message on the socket (This is an alias for recv).
TLSSocket:recv (length)	Receive a message on the socket.
TLSSocket:send (buffer, offset, length)	Send a message to the socket.
TLSSocket:shutdown ()	Shut down the TLS connection.
TLSSocket:write (buffer, offset, length)	Send a message on the socket (This is an alias for send).

Functions

TLSSocket:accept ()

Wait for a TLS handshake from a client.

Usage

- This function calls SSL_accept().
- You have to call either connect or accept before transmitting data.

Return value:

true

See also:

- TLSSocket:connect

TLSSocket:connect ()

Initiate the TLS handshake as client with the server.

Usage

- This function calls SSL_connect().
- You have to call either connect or accept before transmitting data.

Return value:

true

See also:

- TLSSocket:accept

TLSSocket:read (length)

Receive a message on the socket (This is an alias for recv). See the recv description for more details.

Parameters

- length: Amount of data to read (in Bytes).

Return value:

buffer containing data successfully read

See also:

- TLSSocket:recv

TLSSocket:recv (length)

Receive a message on the socket.

Parameters

- length: Amount of data to read (in Bytes).

Usage

- This function calls SSL_read().
- **Warning:** It is not guaranteed that all requested data is read at once. You have to check the return value - the length of the buffer actually read - or use the safe IO functions in the high-level IO utility module.
- The length of the return buffer is limited by the (compile time) nixio buffersize which is nixio.const.buffersize (8192 by default). Any read request greater than that will be safely truncated to this value.

Return value:

buffer containing data successfully read

TLSSocket:send (buffer, offset, length)

Send a message to the socket.

Parameters

- buffer: Buffer holding the data to be written.
- offset: Offset to start reading the buffer from. (optional)
- length: Length of chunk to read from the buffer. (optional)

Usage

- This function calls SSL_write().
- **Warning:** It is not guaranteed that all data in the buffer is written at once. You have to check the return value - the number of bytes actually written - or use the safe IO functions in the high-level IO utility module.
- Unlike standard Lua indexing the lowest offset and default is 0.

Return value:

number of bytes written

TLSSocket:shutdown ()

Shut down the TLS connection.

Usage:

This function calls SSL_shutdown().

Return value:

true

TLSSocket:write (buffer, offset, length)

Send a message on the socket (This is an alias for send). See the send description for a detailed description.

Parameters

- buffer: Buffer holding the data to be written.
- offset: Offset to start reading the buffer from. (optional)
- length: Length of chunk to read from the buffer. (optional)

Return value:

number of bytes written

See also:

- TLSSocket:send

Object Instance *nixio.UnifiedIO*

Unified high-level I/O utility API for Files, Sockets and TLS-Sockets. These functions are added to the object function tables by doing **require "nixio.util"**, can be used on all nixio IO Descriptors and are based on the shared low-level read() and write() functions.

Functions

UnifiedIO:blocksource (blocksize, limit)	Create a block-based iterator.
UnifiedIO:close ()	Close the descriptor.
UnifiedIO:copy (fdout, size)	Copy data from the current descriptor to another one.
UnifiedIO:copyz (fdout, size)	Copy data from the current descriptor to another one using kernel-space copying if possible.
UnifiedIO:is_file ()	Test whether the I/O-Descriptor is a file.
UnifiedIO:is_socket ()	Test whether the I/O-Descriptor is a socket.
UnifiedIO:is_tls_socket ()	Test whether the I/O-Descriptor is a TLS socket.
UnifiedIO:linesource (limit)	Create a line-based iterator.
UnifiedIO:readall (length)	Read a block of data and wait until all data is available.
UnifiedIO:sink (close_when_done)	Create a sink.
UnifiedIO:writeall (block)	Write a block of data and wait until all data is written.

Functions

UnifiedIO:blocksource (blocksize, limit)

Create a block-based iterator.

Parameters

- blocksize: Advisory blocksize (optional)
- limit: Amount of data to consume (optional)

Usage

- This function uses the low-level read function of the descriptor.
- The blocksize given is only advisory and to be seen as an upper limit, if an underlying read returns less bytes the chunk is nevertheless returned.
- If the limit parameter is omitted, the iterator returns data until an end-of-file, end-of-stream, connection shutdown or similar happens.
- The iterator will not buffer so it is safe to mix with calls to read.

- If the descriptor is non-blocking the iterator may fail with EAGAIN.
- The iterator can be used as an LTN12 source.

Return value:

Block-based Iterator

UnifiedIO:close ()

Close the descriptor.

Usage:

If the descriptor is a TLS-socket the underlying descriptor is closed without touching the TLS connection.

Return value:

true

UnifiedIO:copy (fdout, size)

Copy data from the current descriptor to another one.

Parameters

- fdout: Target Descriptor
- size: Bytes to copy (optional)

Usage

- This function uses the blocksource function of the source descriptor and the sink function of the target descriptor.
- If the limit parameter is omitted, data is copied until an end-of-file, end-of-stream, connection shutdown or similar happens.
- If the descriptor is non-blocking the function may fail with EAGAIN.

Return values:

1. bytes that were successfully written if no error occurred
2. - reserved for error code -
3. - reserved for error message -
4. bytes that were successfully written even if an error occurred

---

UnifiedIO:copyz (fdout, size)

Copy data from the current descriptor to another one using kernel-space copying if possible.

Parameters

- fdout: Target Descriptor
- size: Bytes to copy (optional)

Usage

- This function uses the sendfile() syscall to copy the data or the blocksource function of the source descriptor and the sink function of the target descriptor as a fallback mechanism.
- If the limit parameter is omitted, data is copied until an end-of-file, end-of-stream, connection shutdown or similar happens.
- If the descriptor is non-blocking the function may fail with EAGAIN.

Return values:

1. bytes that were successfully written if no error occurred
2. - reserved for error code -
3. - reserved for error message -
4. bytes that were successfully written even if an error occurred

---

UnifiedIO:is_file ()

Test whether the I/O-Descriptor is a file.

Return value:

boolean

UnifiedIO:is_socket ()

Test whether the I/O-Descriptor is a socket.

Return value:

boolean

UnifiedIO:is_tls_socket ()

Test whether the I/O-Descriptor is a TLS socket.

Return value:

boolean

UnifiedIO:linesource (limit)

Create a line-based iterator. Lines may end with either \n or \r\n, these control chars are not included in the return value.

Parameters

- limit: Line limit

Usage

- This function uses the low-level read function of the descriptor.
- **Note:** This function uses an internal buffer to read ahead. Do NOT mix calls to read(all) and the returned iterator. If you want to stop reading line-based and want to use the read(all) functions

instead you can pass "true" to the iterator which will flush the buffer and return the bufferd data.
- If the limit parameter is omitted, this function uses the nixio buffersize (8192B by default).
- If the descriptor is non-blocking the iterator may fail with EAGAIN.
- The iterator can be used as an LTN12 source.

Return value:

Line-based Iterator

UnifiedIO:readall (length)

Read a block of data and wait until all data is available.

Parameters

- length: Bytes to read (optional)

Usage

- This function uses the low-level read function of the descriptor.
- If the length parameter is omitted, this function returns all data that can be read before an end-of-file, end-of-stream, connection shutdown or similar happens.
- If the descriptor is non-blocking this function may fail with EAGAIN.

Return values:

1. data that was successfully read if no error occurred
2. - reserved for error code -
3. - reserved for error message -
4. data that was successfully read even if an error occurred

UnifiedIO:sink (close_when_done)

Create a sink. This sink will simply write all data that it receives and optionally close the descriptor afterwards.

Parameters

- close_when_done: (optional, boolean)

Usage

- This function uses the writeall function of the descriptor.
- If the descriptor is non-blocking the sink may fail with EAGAIN.
- The iterator can be used as an LTN12 sink.

Return value:

Sink

UnifiedIO:writeall (block)

Write a block of data and wait until all data is written.

Parameters

- block: Bytes to write

Usage

- This function uses the low-level write function of the descriptor.
- If the descriptor is non-blocking this function may fail with EAGAIN.

Return values:

1. bytes that were successfully written if no error occurred
2. - reserved for error code -
3. - reserved for error message -
4. bytes that were successfully written even if an error occurred

Class *nixio.bin*

Binary operations and conversion.

Functions

b64decode (buffer)	Base64 decode a given buffer.
b64encode (buffer)	Base64 encode a given buffer.
crc32 (buffer, initial)	Calculate the CRC32 value of a buffer.
hexlify (buffer)	Return a hexadecimal ASCII represantation of the content of a buffer.
unhexlify (hexvalue)	Return a binary buffer from a hexadecimal ASCII representation.

Functions

---

b64decode (buffer)

Base64 decode a given buffer.

Parameters

- buffer: Base64 Encoded data

Return value:

binary data

---

b64encode (buffer)

Base64 encode a given buffer.

Parameters

- buffer: Buffer

Return value:

base64 encoded buffer

---

crc32 (buffer, initial)

Calculate the CRC32 value of a buffer.

Parameters

- buffer: Buffer
- initial: Initial CRC32 value (optional)

Return value:

crc32 value

---

hexlify (buffer)

Return a hexadecimal ASCII represantation of the content of a buffer.

Parameters

- buffer: Buffer

Return value:

representation using characters [0-9a-f]

---

unhexlify (hexvalue)

Return a binary buffer from a hexadecimal ASCII representation.

Parameters

- hexvalue: representation using characters [0-9a-f]

Return value:

binary data

Class *nixio.bit*

Bitfield operators and mainpulation functions. Can be used as a drop-in replacement for bitlib.

Functions

arshift (oper, shift)	Arithmetically right shift a number.
band (oper1, oper2, ...)	Bitwise AND several numbers.
bnot (oper)	Invert given number.
bor (oper1, oper2, ...)	Bitwise OR several numbers.
bxor (oper1, oper2, ...)	Bitwise XOR several numbers.
cast (oper)	Cast a number to the bit-operating range.
check (bitfield, flag1, ...)	Checks whether given flags are set in a bitfield.
div (oper1, oper2, ...)	Integer division of 2 or more numbers.
lshift (oper, shift)	Left shift a number.
rshift (oper, shift)	Right shift a number.
set (bitfield, flag1, ...)	Sets one or more flags of a bitfield.
unset (bitfield, flag1, ...)	Unsets one or more flags of a bitfield.

Functions

`arshift (oper, shift)`

Arithmetically right shift a number.

Parameters

- `oper`: number
- `shift`: bits to shift

Return value:

number

`band (oper1, oper2, ...)`

Bitwise AND several numbers.

Parameters

- `oper1`: First Operand
- `oper2`: Second Operand
- `...`: More Operands

Return value:

number

bnot (oper)

Invert given number.

Parameters

- oper: Operand

Return value:

number

---

bor (oper1, oper2, ...)

Bitwise OR several numbers.

Parameters

- oper1: First Operand
- oper2: Second Operand
- ...: More Operands

Return value:

number

---

bxor (oper1, oper2, ...)

Bitwise XOR several numbers.

Parameters

- oper1: First Operand
- oper2: Second Operand
- ...: More Operands

Return value:

number

---

cast (oper)

Cast a number to the bit-operating range.

Parameters

- oper: number

Return value:

number

---

check (bitfield, flag1, ...)

Checks whether given flags are set in a bitfield.

Parameters

- bitfield: Bitfield
- flag1: First Flag

- ...: More Flags

Return value:

true when all flags are set, otherwise false

---

div (oper1, oper2, ...)

Integer division of 2 or more numbers.

Parameters

- oper1: Operand 1
- oper2: Operand 2
- ...: More Operands

Return value:

number

---

lshift (oper, shift)

Left shift a number.

Parameters

- oper: number
- shift: bits to shift

Return value:

number

---

rshift (oper, shift)

Right shift a number.

Parameters

- oper: number
- shift: bits to shift

Return value:

number

---

set (bitfield, flag1, ...)

Sets one or more flags of a bitfield.

Parameters

- bitfield: Bitfield
- flag1: First Flag
- ...: More Flags

Return value:

altered bitfield

unset (bitfield, flag1, ...)

Unsets one or more flags of a bitfield.

Parameters

- bitfield: Bitfield
- flag1: First Flag
- ...: More Flags

Return value:

altered bitfield

Class *nixio.crypto*

Cryptographical library.

Functions

hash (algo)	Create a hash object.
hmac (algo, key)	Create a HMAC object.

Functions

hash (algo)

Create a hash object.

Parameters

- algo: Algorithm ["sha1", "md5"]

Return value:

CryptoHash Object

hmac (algo, key)

Create a HMAC object.

Parameters

- algo: Algorithm ["sha1", "md5"]
- key: HMAC-Key

Return value:

CryptoHash Object

Class *nixio.fs*

Low-level and high-level filesystem manipulation library.

Functions

access (path, mode1, ...)	Check user's permission on a file.
basename (path)	Strip the directory part from a path.
chmod (path, mode)	Change the file mode.
chown (path, user, group)	(POSIX) Change owner and group of a file.
copy (src, dest)	Copy a file, directory or symlink non-recursively preserving file mode, timestamps, owner and group.
copyr (src, dest)	Copy a file, directory or symlink recursively preserving file mode, timestamps, owner and group.
datacopy (src, dest, limit)	Copy data between files.
dir (path)	Iterate over the entries of a directory.
dirname (path)	Strip the base from a path.
glob (pattern)	(POSIX) Find pathnames matching a pattern.
lchown (path, user, group)	(POSIX) Change owner and group of a file and do not resolve if target is a symlink.
link (oldpath, newpath)	Create a hard link.
lstat (path, field)	Get file status and attributes and do not resolve if target is a symlink.
mkdir (path, mode)	Create a new directory.
mkdirr (dest, mode)	Create a directory and all needed parent directories recursively.
mkfifo (path, mode)	(POSIX) Create a FIFO (named pipe).
move (src, dest)	Rename a file, directory or symlink non-recursively across filesystems.
mover (src, dest)	Rename a file, directory or symlink recursively across filesystems.
readfile (path, limit)	Read the contents of a file into a buffer.
readlink (path)	(POSIX) Read the target of a symbolic link.
realpath (path)	Return the cannonicalized absolute pathname.
remove (path)	Remove a file or directory.
rename (src, dest)	Renames a file or directory.

rmdir (path)	Remove an empty directory.
stat (path, field)	Get file status and attributes.
statvfs (path)	(POSIX) Get filesystem statistics.
symlink (oldpath, newpath)	(POSIX) Create a symbolic link.
unlink (path)	Delete a name and - if no links are left - the associated file.
utimes (path, actime, mtime)	Change file last access and last modification time.
writefile (path, data)	Write a buffer into a file truncating the file first.

Functions

---

access (path, mode1, ...)

Check user's permission on a file.

Parameters

- path: Path
- mode1: First Mode to check ["f", "r", "w", "x"]
- ...: More Modes to check [-"-]

Return value:

true

---

basename (path)

Strip the directory part from a path.

Parameters

- path: Path

Usage:

This function cannot fail and will never return nil.

Return value:

basename

---

chmod (path, mode)

Change the file mode.

Parameters

- path: Path
- mode: File mode [decimal mode number, "[-r][-w][-xsS][-r][-w][-xsS][-r][-w][-xtT]"]

Usage

- Windows only supports setting the write-protection through the "Writable to others" bit.
- **Notice:** The mode-flag for the functions open, mkdir, mkfifo are affected by the umask.

Return value:

true

See also:

- umask

---

chown (path, user, group)

(POSIX) Change owner and group of a file.

Parameters

- path: Path
- user: User ID or Username (optional)
- group: Group ID or Groupname (optional)

Return value:

true

---

copy (src, dest)

Copy a file, directory or symlink non-recursively preserving file mode, timestamps, owner and group.

Parameters

- src: Source path
- dest: Destination path

Usage:

The destination must always be a full destination path e.g. do not omit the basename even if source and destination basename are equal.

Return value:

true

---

copyr (src, dest)

Copy a file, directory or symlink recursively preserving file mode, timestamps, owner and group.

Parameters

- src: Source path
- dest: Destination path

Usage:

The destination must always be a full destination path e.g. do not omit the basename even if source and destination basename are equal.

Return value:

true

---

datacopy (src, dest, limit)

Copy data between files.

Parameters

- src: Source file path
- dest: Destination file path
- limit: Maximum bytes to copy (optional)

Return value:

true

---

dir (path)

Iterate over the entries of a directory.

Parameters

- path: Path

Usage:

The special entries "." and ".." are omitted.

Return value:

directory iterator returning one entry per call

---

dirname (path)

Strip the base from a path.

Parameters

- path: Path

Usage:

This function cannot fail and will never return nil.

Return value:

dirname

---

glob (pattern)

(POSIX) Find pathnames matching a pattern.

Parameters

- pattern: Pattern

Return values:

1. path iterator
2. number of matches

---

lchown (path, user, group)

(POSIX) Change owner and group of a file and do not resolve if target is a symlink.

Parameters

- path: Path
- user: User ID or Username (optional)
- group: Group ID or Groupname (optional)

Return value:

true

---

link (oldpath, newpath)

Create a hard link.

Parameters

- oldpath: Path
- newpath: Path

Usage:

This function calls link() on POSIX and CreateHardLink() on Windows.

Return value:

true

---

lstat (path, field)

Get file status and attributes and do not resolve if target is a symlink.

Parameters

- path: Path
- field: Only return a specific field, not the whole table (optional)

Return value:

Table containing attributes (see stat for a detailed description)

See also:

- stat

---

mkdir (path, mode)

Create a new directory.

Parameters

- path: Path
- mode: File mode (optional, see chmod and umask)

Return value:

true

See also:

- chmod
- umask

---

mkdirr (dest, mode)

Create a directory and all needed parent directories recursively.

Parameters

- dest: Destination path
- mode: File mode (optional, see chmod and umask)

Return value:

true

See also:

- chmod
- umask

---

mkfifo (path, mode)

(POSIX) Create a FIFO (named pipe).

Parameters

- path: Path
- mode: File mode (optional, see chmod and umask)

Return value:

true

See also:

- chmod
- umask

---

move (src, dest)

Rename a file, directory or symlink non-recursively across filesystems.

Parameters

- src: Source path
- dest: Destination path

Usage:

The destination must always be a full destination path e.g. do not omit the basename even if source and destination basename are equal.

Return value:

true

---

mover (src, dest)

Rename a file, directory or symlink recursively across filesystems.

Parameters

- src: Source path
- dest: Destination path

Usage:

The destination must always be a full destination path e.g. do not omit the basename even if source and destination basename are equal.

Return value:

true

---

readfile (path, limit)

Read the contents of a file into a buffer.

Parameters

- path: Path
- limit: Maximum bytes to read (optional)

Return value:

file contents

---

readlink (path)

(POSIX) Read the target of a symbolic link.

Parameters

- path: Path

Return value:

target path

---

realpath (path)

Return the cannonicalized absolute pathname.

Parameters

- path: Path

Return value:

absolute path

---

remove (path)

Remove a file or directory.

Parameters

- path: Path

Return value:

true

---

rename (src, dest)

Renames a file or directory.

Parameters

- src: Source path
- dest: Destination path

Usage:

It is normally not possible to rename files across filesystems.

Return value:

true

---

rmdir (path)

Remove an empty directory.

Parameters

- path: Path

Return value:

true

---

stat (path, field)

Get file status and attributes.

Parameters

- path: Path
- field: Only return a specific field, not the whole table (optional)

Return value:

Table containing:

- atime = Last access timestamp
- blksize = Blocksize (POSIX only)

- blocks = Blocks used (POSIX only)
- ctime = Creation timestamp
- dev = Device ID
- gid = Group ID
- ino = Inode
- modedec = Mode converted into a decimal number
- modestr = Mode as string as returned by `ls -l`
- mtime = Last modification timestamp
- nlink = Number of links
- rdev = Device ID (if special file)
- size = Size in bytes
- type = ["reg", "dir", "chr", "blk", "fifo", "lnk", "sock"]
- uid = User ID

---

statvfs (path)

(POSIX) Get filesystem statistics.

Parameters

- path: Path to any file within the filesystem.

Return value:

Table containing:

- bavail = available blocks
- bfree = free blocks
- blocks = number of fragments
- frsize = fragment size
- favail = available inodes
- ffree = free inodes
- files = inodes
- flag = flags
- fsid = filesystem ID
- namemax = maximum filename length

---

symlink (oldpath, newpath)

(POSIX) Create a symbolic link.

Parameters

- oldpath: Path
- newpath: Path

Return value:

true

---

unlink (path)

Delete a name and - if no links are left - the associated file.

Parameters

- path: Path

Return value:

true

---

utimes (path, actime, mtime)

Change file last access and last modification time.

Parameters

- path: Path
- actime: Last access timestamp (optional, default: current time)
- mtime: Last modification timestamp (optional, default: actime)

Return value:

true

---

writefile (path, data)

Write a buffer into a file truncating the file first.

Parameters

- path: Path
- data: Data to write

Return value:

true