

Package: multichainr (via r-universe)

September 15, 2026

Title R Interface to the 'MultiChain' Blockchain RPC API

Version 0.1.0

Description Provides a comprehensive R interface to the 'MultiChain' blockchain JSON-RPC API

<<https://www.multichain.com/developers/json-rpc-api/>>. Allows users to manage blockchain nodes, create and subscribe to data streams, issue assets, and manage network permissions directly from the R console. Supports both local node management and remote server interaction.

License MIT + file LICENSE

URL <https://github.com/datascienceadvice/multichainr>,
<https://datascienceadvice.github.io/multichainr/>

BugReports <https://github.com/datascienceadvice/multichainr/issues>

Depends R (>= 3.6.0)

Imports htr2, jsonlite, magrittr

Suggests knitr, rmarkdown, testthat (>= 3.0.0), withr

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

SystemRequirements MultiChain (>= 2.0)
<<https://www.multichain.com/download-install/>>

Config/pak/sysreqs libssl-dev

Repository <https://datascienceadvice.r-universe.dev>

Date/Publication 2026-07-30 17:47:43 UTC

RemoteUrl <https://github.com/datascienceadvice/multichainr>

RemoteRef HEAD

RemoteSha b1ff93be1991b81c37b915806c1ce16283a89d3b

Contents

hex_to_char	5
mc_add_library_update	6
mc_add_library_update_from	7
mc_add_multisig_address	8
mc_add_node	9
mc_append_binary_cache	10
mc_append_raw_change	10
mc_append_raw_data	11
mc_append_raw_exchange	12
mc_append_raw_transaction	13
mc_approve_from	14
mc_backup_wallet	15
mc_change_wallet_passphrase	16
mc_clear_mempool	16
mc_combine_unspent	17
mc_complete_raw_exchange	18
mc_connect	19
mc_create_binary_cache	20
mc_create_keypairs	21
mc_create_library	22
mc_create_multisig	23
mc_create_raw_exchange	24
mc_create_raw_send_from	25
mc_create_raw_transaction	26
mc_create_stream	27
mc_create_stream_filter	28
mc_create_stream_from	29
mc_create_tx_filter	30
mc_create_upgrade	31
mc_create_variable	32
mc_create_variable_from	33
mc_decode_raw_exchange	34
mc_decode_raw_transaction	35
mc_delete_binary_cache	36
mc_disable_raw_transaction	37
mc_dump_privkey	38
mc_dump_wallet	38
mc_encrypt_wallet	39
mc_get_added_node_info	40
mc_get_address_balances	41
mc_get_address_transaction	41
mc_get_addresses	42
mc_get_asset_info	43
mc_get_asset_transaction	44
mc_get_block	45
mc_get_block_hash	46

mc_get_blockchain_info	46
mc_get_blockchain_params	47
mc_get_chain_totals	48
mc_get_chunk_queue_info	49
mc_get_chunk_queue_totals	50
mc_get_config	51
mc_get_filter_code	52
mc_get_info	52
mc_get_init_status	53
mc_get_last_block_info	54
mc_get_library_code	55
mc_get_mempool_info	56
mc_get_multi_balances	57
mc_get_network_info	58
mc_get_new_address	59
mc_get_peer_info	59
mc_get_raw_mempool	60
mc_get_raw_transaction	61
mc_get_runtime_params	62
mc_get_stream_info	63
mc_get_stream_item	64
mc_get_stream_key_summary	65
mc_get_stream_publisher_summary	66
mc_get_token_balances	67
mc_get_token_info	68
mc_get_total_balances	69
mc_get_tx_out	69
mc_get_tx_out_data	70
mc_get_variable_history	71
mc_get_variable_info	72
mc_get_variable_value	73
mc_get_wallet_info	73
mc_get_wallet_transaction	74
mc_grant	75
mc_grant_from	76
mc_grant_with_data	77
mc_grant_with_data_from	78
mc_help	79
mc_import_address	80
mc_import_privkey	81
mc_import_wallet	81
mc_issue	82
mc_issue_from	83
mc_issue_more	84
mc_issue_more_from	85
mc_issue_token	86
mc_issue_token_from	87
mc_list_address_transactions	89

mc_list_addresses	90
mc_list_asset_issues	91
mc_list_asset_transactions	92
mc_list_assets	93
mc_list_blocks	94
mc_list_libraries	95
mc_list_lock_unspent	96
mc_list_miners	96
mc_list_permissions	97
mc_list_stored_nodes	98
mc_list_stream_block_items	99
mc_list_stream_filters	100
mc_list_stream_items	101
mc_list_stream_key_items	102
mc_list_stream_keys	103
mc_list_stream_publisher_items	104
mc_list_stream_publishers	105
mc_list_stream_query_items	106
mc_list_stream_tx_items	107
mc_list_streams	108
mc_list_tx_filters	109
mc_list_unspent	110
mc_list_upgrades	110
mc_list_variables	111
mc_list_wallet_transactions	112
mc_lock_unspent	113
mc_lock_wallet	114
mc_node_init	114
mc_node_start	115
mc_node_stop	116
mc_pause	117
mc_ping	118
mc_prepare_lock_unspent	119
mc_prepare_lock_unspent_from	120
mc_publish	121
mc_publish_from	122
mc_publish_multi	123
mc_publish_multi_from	124
mc_resume	125
mc_revoke	126
mc_revoke_from	127
mc_run_stream_filter	128
mc_run_tx_filter	129
mc_send	130
mc_send_asset	131
mc_send_asset_from	132
mc_send_from	133
mc_send_raw_transaction	134

mc_send_with_data	135
mc_send_with_data_from	136
mc_set_last_block	137
mc_set_path	138
mc_set_runtime_param	139
mc_set_variable_value	140
mc_set_variable_value_from	141
mc_sign_message	142
mc_sign_raw_transaction	143
mc_store_node	144
mc_subscribe	145
mc_test_library	146
mc_test_stream_filter	147
mc_test_tx_filter	148
mc_txout_to_binary_cache	149
mc_unlock_wallet	150
mc_unsubscribe	151
mc_update	151
mc_update_from	152
mc_validate_address	153
mc_verify_message	154
mc_verify_permission	155
mc_wait_for_confirmation	156
null_default	156
print.mc_help	157
print.multichain_conn	157

Index 159

hex_to_char	<i>Decode hex string to character</i>
-------------	---------------------------------------

Description

Converts a hexadecimal encoded string back into its original character representation. This is primarily used for reading human-readable data published to MultiChain streams.

Usage

```
hex_to_char(hex_str)
```

Arguments

hex_str	A character string in hexadecimal format.
---------	---

Details

The function performs a basic validation to ensure the string is a valid hexadecimal representation (even length and containing only hex characters) before attempting to convert.

Value

A decoded character string. If the input is not a valid hex string or an error occurs during decoding, the original hex_str is returned.

Examples

```
hex_to_char("48656c6c66") # Returns "Hello"
```

mc_add_library_update *Add an update to an existing library*

Description

Adds a new version (update) to an existing library. The update mechanism depends on the library's updatemode.

Usage

```
mc_add_library_update(conn, library, updatename, js_code)
```

Arguments

conn	A connection object created by mc_connect .
library	Character string. Library name or transaction ID.
updatename	Character string. Name of this update (must be unique).
js_code	Character string. The new JavaScript code (or patch).

Value

A list containing the result of the RPC call (usually transaction ID).

See Also

[mc_create_library](#), [mc_add_library_update_from](#)

Other libraries: [mc_add_library_update_from\(\)](#), [mc_create_library\(\)](#), [mc_get_library_code\(\)](#), [mc_list_libraries\(\)](#), [mc_test_library\(\)](#)

Examples

```
## Not run:
mc_add_library_update(conn, "math", "v2", "function add(a, b) { return a + b + 1; }")

## End(Not run)
```

`mc_add_library_update_from`*Add an update to a library from a specific address*

Description

Similar to [mc_add_library_update](#), but allows specifying the sending address.

Usage

```
mc_add_library_update_from(conn, from_address, library, updatename, js_code)
```

Arguments

<code>conn</code>	A connection object created by mc_connect .
<code>from_address</code>	Character string. Address that pays for and issues the update.
<code>library</code>	Character string. Library name or transaction ID.
<code>updatename</code>	Character string. Name of this update.
<code>js_code</code>	Character string. The new JavaScript code (or patch).

Value

A list containing the result of the RPC call (usually transaction ID).

See Also

Other libraries: [mc_add_library_update\(\)](#), [mc_create_library\(\)](#), [mc_get_library_code\(\)](#), [mc_list_libraries\(\)](#), [mc_test_library\(\)](#)

Examples

```
## Not run:  
mc_add_library_update_from(conn, "1A...", "math", "v2", "function add(a, b) { return a + b; }")  
  
## End(Not run)
```

mc_add_multisig_address

Add a multi-signature address

Description

Creates a multi-signature address and adds it to the node's wallet. The address is a Pay-to-Script-Hash (P2SH) address that requires a specified number of signatures from the provided keys.

Usage

```
mc_add_multisig_address(conn, n_required, keys)
```

Arguments

conn	A connection object created by mc_connect .
n_required	Integer. Number of signatures required to spend funds.
keys	Character vector. Public keys or addresses that will be part of the multi-signature set.

Value

A character string containing the multi-signature address.

See Also

[mc_create_multisig](#) to create a multisig address without adding it to the wallet.

Other addresses: [mc_create_keypairs\(\)](#), [mc_create_multisig\(\)](#), [mc_get_addresses\(\)](#), [mc_get_new_address\(\)](#), [mc_import_address\(\)](#), [mc_list_addresses\(\)](#), [mc_validate_address\(\)](#)

Examples

```
## Not run:
# Assume connection 'conn' is already established
addr <- mc_add_multisig_address(conn, n_required = 2,
                               keys = c("1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa",
                                         "1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2"))

## End(Not run)
```

mc_add_node

Add or remove a peer-to-peer connection

Description

Manages the node's peer connections. Can add a node to the connection queue, remove an existing connection, or attempt a one-time connection.

Usage

```
mc_add_node(conn, node, command = c("add", "remove", "onetry"))
```

Arguments

conn	A connection object created by mc_connect .
node	Character string. The IP address and port of the peer node, e.g., "127.0.0.1:8571".
command	Character string. The action to perform: "add" – add the node to the connection queue (will attempt to connect and stay connected). "remove" – disconnect from the node if currently connected. "onetry" – attempt a single connection; do not keep retrying.

Value

Invisibly returns the RPC result (typically NULL) on success; throws an error if the command fails.

See Also

[mc_get_added_node_info](#) to list added nodes, [mc_get_peer_info](#) for connected peers.

Other networking: [mc_get_added_node_info\(\)](#), [mc_get_network_info\(\)](#), [mc_get_peer_info\(\)](#), [mc_list_stored_nodes\(\)](#), [mc_ping\(\)](#), [mc_store_node\(\)](#)

Examples

```
## Not run:
# Add a peer
mc_add_node(conn, "192.168.1.10:8571", command = "add")

# Remove a peer
mc_add_node(conn, "192.168.1.10:8571", command = "remove")

## End(Not run)
```

mc_append_binary_cache

Append data to a binary cache item

Description

Appends data to an existing binary cache item. If data = "" (the default), the RPC call returns the current size without adding new data.

Usage

```
mc_append_binary_cache(conn, identifier, data = "")
```

Arguments

conn	A connection object created by mc_connect .
identifier	Character string. The cache item identifier returned by mc_create_binary_cache .
data	Data to append. Can be: • a character string of hex data, • a list with element text (will be converted to hex), • a list with element json (will be converted to JSON and then to hex), • "" (default) returns the current size without appending.

Value

Integer. The resulting size of the cache item in bytes after appending (or the current size if data = "").

See Also

Other binary cache: [mc_create_binary_cache\(\)](#), [mc_delete_binary_cache\(\)](#), [mc_txout_to_binary_cache\(\)](#)

mc_append_raw_change *Add a change output to a raw transaction*

Description

Appends a change output to a raw transaction. This is useful when the transaction inputs exceed the required output amounts; the change is sent back to the specified address.

Usage

```
mc_append_raw_change(conn, tx_hex, address, native_fee = NULL)
```

Arguments

conn	A connection object created by mc_connect .
tx_hex	Character string. The raw transaction hex to which change is added.
address	Character string. The address that will receive the change.
native_fee	Optional numeric. Native currency fee to be deducted from the change. If provided, the change output is reduced accordingly.

Value

A character string containing the updated raw transaction hex.

See Also

[mc_append_raw_data](#), [mc_append_raw_transaction](#)

Other raw transactions: [mc_append_raw_data\(\)](#), [mc_append_raw_transaction\(\)](#), [mc_create_raw_send_from\(\)](#), [mc_create_raw_transaction\(\)](#), [mc_decode_raw_transaction\(\)](#), [mc_send_raw_transaction\(\)](#), [mc_sign_raw_transaction\(\)](#)

Examples

```
## Not run:
# Add change to a raw transaction
updated_hex <- mc_append_raw_change(conn, tx_hex, "1A...")

## End(Not run)
```

mc_append_raw_data	<i>Add metadata to a raw transaction</i>
--------------------	--

Description

Appends arbitrary data (metadata) to a raw transaction. The data is embedded in an output with zero native value.

Usage

```
mc_append_raw_data(conn, tx_hex, data)
```

Arguments

conn	A connection object created by mc_connect .
tx_hex	Character string. The raw transaction hex.
data	Data to embed. Can be a string or a list (converted to JSON then hex).

Value

A character string containing the updated raw transaction hex.

See Also

[mc_append_raw_change](#), [mc_append_raw_transaction](#)

Other raw transactions: [mc_append_raw_change\(\)](#), [mc_append_raw_transaction\(\)](#), [mc_create_raw_send_from\(\)](#), [mc_create_raw_transaction\(\)](#), [mc_decode_raw_transaction\(\)](#), [mc_send_raw_transaction\(\)](#), [mc_sign_raw_transaction\(\)](#)

Examples

```
## Not run:
# Add a text note
updated_hex <- mc_append_raw_data(conn, tx_hex, "This is a note.")

# Add JSON metadata
updated_hex <- mc_append_raw_data(conn, tx_hex, list(tag = "invoice", id = 123))

## End(Not run)
```

mc_append_raw_exchange

Add to a raw atomic exchange transaction

Description

Appends a new input–output pair to a partially constructed atomic exchange transaction. This function is used when multiple parties are contributing to the exchange, each adding their own locked output and specifying what they want in return.

Usage

```
mc_append_raw_exchange(conn, tx_hex, txid, vout, amounts)
```

Arguments

conn	A connection object created by mc_connect .
tx_hex	Character string. Hexadecimal representation of the partial exchange transaction.
txid	Character string. Transaction ID of the output being added to the offer.
vout	Integer. Output index (vout) of the transaction being added.
amounts	A list specifying the assets or native currency asked for in exchange for this addition. Format: <code>list(asset_name = quantity, ...)</code> or a numeric value for native currency.

Value

A list with two elements:

hex	The new partial transaction hex string.
complete	Logical; TRUE if the exchange is now complete.

See Also

[mc_create_raw_exchange](#), [mc_complete_raw_exchange](#)

Other atomic exchange: [mc_complete_raw_exchange\(\)](#), [mc_create_raw_exchange\(\)](#), [mc_decode_raw_exchange\(\)](#), [mc_disable_raw_transaction\(\)](#), [mc_prepare_lock_unspent\(\)](#), [mc_prepare_lock_unspent_from\(\)](#)

Examples

```
## Not run:
# Assume 'partial_hex' is a partial exchange from a previous step
new <- mc_append_raw_exchange(conn, partial_hex,
                             txid = "abc...", vout = 0,
                             amounts = list(myasset = 10))

## End(Not run)
```

mc_append_raw_transaction

Add inputs and outputs to a raw transaction

Description

Appends additional inputs and outputs to an existing raw transaction. This is useful for building multi-party transactions or adding extra components after creation.

Usage

```
mc_append_raw_transaction(conn, tx_hex, inputs = list(), outputs = list())
```

Arguments

conn	A connection object created by mc_connect .
tx_hex	Character string. The raw transaction hex to which inputs/outputs are added.
inputs	A list of input objects (each with txid and vout).
outputs	A named list of outputs (mapping addresses to amounts).

Value

A character string containing the updated raw transaction hex.

See Also

`mc_create_raw_transaction`, `mc_append_raw_change`

Other raw transactions: `mc_append_raw_change()`, `mc_append_raw_data()`, `mc_create_raw_send_from()`, `mc_create_raw_transaction()`, `mc_decode_raw_transaction()`, `mc_send_raw_transaction()`, `mc_sign_raw_transaction()`

Examples

```
## Not run:
# Add another input and output
new_input <- list(list(txid = "def...", vout = 1))
new_output <- list("1C..." = 0.2)
updated_hex <- mc_append_raw_transaction(conn, tx_hex,
                                         inputs = new_input,
                                         outputs = new_output)

## End(Not run)
```

mc_approve_from	<i>Approve or disapprove an upgrade or filter</i>
-----------------	---

Description

Sends an approval or disapproval transaction from a specific address (must have admin permissions). This is used to vote on upgrades or to approve/disapprove filters.

Usage

```
mc_approve_from(conn, from_address, entity, approve)
```

Arguments

conn	A connection object created by <code>mc_connect</code> .
from_address	Character string. Admin address that issues the approval.
entity	Character string. Name or transaction ID of the upgrade/filter.
approve	Either a logical (for global upgrades/filters) or a list of the form <code>list("for" = "stream", approve = TRUE)</code> for stream-specific filter approval.

Value

A list containing the result of the RPC call (usually transaction ID).

See Also

`mc_create_upgrade`, `mc_create_stream_filter`

Other filters: `mc_create_stream_filter()`, `mc_create_tx_filter()`, `mc_create_upgrade()`, `mc_get_filter_code()`, `mc_list_stream_filters()`, `mc_list_tx_filters()`, `mc_list_upgrades()`, `mc_run_stream_filter()`, `mc_run_tx_filter()`, `mc_test_stream_filter()`, `mc_test_tx_filter()`

Examples

```
## Not run:
# Approve a global upgrade
mc_approve_from(conn, "admin_address", "speedup", approve = TRUE)

# Approve a stream filter for a specific stream
mc_approve_from(conn, "admin_address", "myfilter",
               approve = list("for" = "mystream", approve = TRUE))

## End(Not run)
```

mc_backup_wallet	<i>Backup the wallet file</i>
------------------	-------------------------------

Description

Safely copies the `wallet.dat` file to a specified destination.

Usage

```
mc_backup_wallet(conn, filename)
```

Arguments

conn	A connection object to the MultiChain node.
filename	Character. The full path and filename for the backup. Note: This path is relative to the machine where the MultiChain node is running.

Value

Invisibly returns the RPC result (typically NULL) on success.

See Also

Other wallet: `mc_change_wallet_passphrase()`, `mc_combine_unspent()`, `mc_dump_privkey()`, `mc_dump_wallet()`, `mc_encrypt_wallet()`, `mc_get_wallet_info()`, `mc_import_privkey()`, `mc_import_wallet()`, `mc_list_lock_unspent()`, `mc_list_unspent()`, `mc_lock_unspent()`, `mc_lock_wallet()`, `mc_unlock_wallet()`

mc_change_wallet_passphrase
Change the wallet passphrase

Description

Changes the encryption password of the wallet.

Usage

```
mc_change_wallet_passphrase(conn, old_passphrase, new_passphrase)
```

Arguments

conn A connection object to the MultiChain node.
old_passphrase Character. The current password.
new_passphrase Character. The new password.

Value

Invisibly returns the RPC result (typically NULL) on success.

See Also

Other wallet: [mc_backup_wallet\(\)](#), [mc_combine_unspent\(\)](#), [mc_dump_privkey\(\)](#), [mc_dump_wallet\(\)](#), [mc_encrypt_wallet\(\)](#), [mc_get_wallet_info\(\)](#), [mc_import_privkey\(\)](#), [mc_import_wallet\(\)](#), [mc_list_lock_unspent\(\)](#), [mc_list_unspent\(\)](#), [mc_lock_unspent\(\)](#), [mc_lock_wallet\(\)](#), [mc_unlock_wallet\(\)](#)

mc_clear_mempool *Clear the node's memory pool*

Description

Removes all unconfirmed transactions from the node's memory pool (mempool). This function is typically used after pausing incoming and mining tasks to reset the mempool state. It requires the node to be paused first with [mc_pause\(conn, "incoming,mining"\)](#).

Usage

```
mc_clear_mempool(conn)
```

Arguments

conn A connection object created by [mc_connect](#).

Value

Invisibly returns the RPC result (typically NULL) on success.

See Also

[mc_pause](#), [mc_resume](#), [mc_get_mempool_info](#) to inspect mempool state.

Examples

```
## Not run:
# Pause the node before clearing the mempool
mc_pause(conn, "incoming,mining")
mc_clear_mempool(conn)
mc_resume(conn, "incoming,mining")

## End(Not run)
```

mc_combine_unspent	<i>Combine unspent outputs (UTXOs)</i>
--------------------	--

Description

Sends transactions to combine many small unspent transaction outputs (UTXOs) into a single output. This is used to improve wallet performance and reduce the size of the wallet's UTXO set.

Usage

```
mc_combine_unspent(
  conn,
  addresses = "*",
  minconf = 1,
  maxcombines = 100,
  mininputs = 2,
  maxinputs = 100,
  maxtime = 15
)
```

Arguments

conn	A connection object to the MultiChain node.
addresses	A vector of addresses, or "*" for all addresses (default "*").
minconf	Integer. Minimum confirmations (default 1).
maxcombines	Integer. Maximum number of transactions to create (default 100).
mininputs	Integer. Minimum number of inputs per transaction (default 2).
maxinputs	Integer. Maximum number of inputs per transaction (default 100).
maxtime	Integer. Maximum seconds to spend combining (default 15).

Value

A character vector of the transaction IDs (txids) created.

See Also

Other wallet: [mc_backup_wallet\(\)](#), [mc_change_wallet_passphrase\(\)](#), [mc_dump_privkey\(\)](#), [mc_dump_wallet\(\)](#), [mc_encrypt_wallet\(\)](#), [mc_get_wallet_info\(\)](#), [mc_import_privkey\(\)](#), [mc_import_wallet\(\)](#), [mc_list_lock_unspent\(\)](#), [mc_list_unspent\(\)](#), [mc_lock_unspent\(\)](#), [mc_lock_wallet\(\)](#), [mc_unlock_wallet\(\)](#)

mc_complete_raw_exchange

Finalize an atomic exchange transaction

Description

Completes a multi-party atomic exchange by adding the final input–output pair. After this step, the transaction is fully built and ready to be broadcast.

Usage

```
mc_complete_raw_exchange(conn, tx_hex, txid, vout, amounts, data = NULL)
```

Arguments

conn	A connection object created by mc_connect .
tx_hex	Character string. Hexadecimal representation of the partial exchange transaction (should already contain all other parties' contributions).
txid	Character string. Transaction ID of the completing output.
vout	Integer. Output index (vout) of the completing transaction.
amounts	A list specifying the assets or native currency for the final part of the exchange.
data	Optional metadata. Can be a character string or a list (which will be converted to JSON and then to hex). This data is embedded in the transaction.

Value

Character string. Raw transaction hex ready for sending via [mc_send_raw_transaction](#).

See Also

[mc_create_raw_exchange](#), [mc_append_raw_exchange](#)

Other atomic exchange: [mc_append_raw_exchange\(\)](#), [mc_create_raw_exchange\(\)](#), [mc_decode_raw_exchange\(\)](#), [mc_disable_raw_transaction\(\)](#), [mc_prepare_lock_unspent\(\)](#), [mc_prepare_lock_unspent_from\(\)](#)

Examples

```
## Not run:
final <- mc_complete_raw_exchange(conn, partial_hex,
                                txid = "def...", vout = 1,
                                amounts = list(ETH = 5),
                                data = "exchange complete")

## End(Not run)
```

mc_connect	<i>Create a MultiChain connection object</i>
------------	--

Description

Establishes a connection to a MultiChain node by constructing an RPC endpoint. The function accepts either explicit parameters (host, port, user, password) or a configuration list (typically obtained from [mc_get_config](#)).

Usage

```
mc_connect(host = "127.0.0.1", port = NULL, user = NULL, password = NULL)
```

Arguments

host	Either a character string with the IP address or hostname of the MultiChain node, or a configuration list (as returned by mc_get_config) containing host, port, user, and password. When a list is provided, the other arguments are ignored.
port	Integer. RPC port number. Required unless host is a list.
user	Character string. RPC username. Required unless host is a list.
password	Character string. RPC password. Required unless host is a list.

Value

An object of class "multichain_conn" containing the RPC URL, username, and password (the password is stored but hidden in printing).

See Also

[mc_get_config](#) to obtain a configuration list, [print.multichain_conn](#) for printing connections.

Examples

```
## Not run:
# Using explicit parameters
conn <- mc_connect(host = "127.0.0.1", port = 8570,
                  user = "multichainrpc", password = "mysecret")

# Using a configuration object from mc_get_config
config <- mc_get_config("my_chain")
conn <- mc_connect(config)

## End(Not run)
```

mc_create_binary_cache

Create a new binary cache item

Description

Creates an empty item (file) in the node's binary cache and returns its unique identifier. Binary cache items are temporary storage for binary data that can be used in transactions or passed between nodes.

Usage

```
mc_create_binary_cache(conn)
```

Arguments

conn A connection object created by [mc_connect](#).

Value

A character string identifier (filename) for the newly created cache item.

See Also

[mc_append_binary_cache](#) to add data, [mc_delete_binary_cache](#) to remove.

Other binary cache: [mc_append_binary_cache\(\)](#), [mc_delete_binary_cache\(\)](#), [mc_txout_to_binary_cache\(\)](#)

Examples

```
## Not run:
id <- mc_create_binary_cache(conn)

## End(Not run)
```

mc_create_keypairs	Create new key pairs
--------------------	----------------------

Description

Generates one or more public/private key pairs. These keys are *not* stored in the node's wallet, so they must be kept secure by the user.

Usage

```
mc_create_keypairs(conn, count = 1)
```

Arguments

conn	A connection object created by mc_connect .
count	Integer. Number of key pairs to generate. Default is 1.

Value

A data frame with three columns:

address	The public address derived from the key pair.
pubkey	The public key.
privkey	The private key (keep this secret!).

See Also

[mc_get_new_address](#) to create an address stored in the wallet.

Other addresses: [mc_add_multisig_address\(\)](#), [mc_create_multisig\(\)](#), [mc_get_addresses\(\)](#), [mc_get_new_address\(\)](#), [mc_import_address\(\)](#), [mc_list_addresses\(\)](#), [mc_validate_address\(\)](#)

Examples

```
## Not run:
# Generate a single key pair
keys <- mc_create_keypairs(conn)
print(keys)

# Generate 5 key pairs
keys5 <- mc_create_keypairs(conn, count = 5)

## End(Not run)
```

mc_create_library	Create a new library
-------------------	----------------------

Description

Creates a JavaScript library on the blockchain. Libraries contain reusable code that can be imported by filters.

Usage

```
mc_create_library(
  conn,
  name,
  updatemode = c("none", "instant", "approve"),
  js_code
)
```

Arguments

conn	A connection object created by mc_connect .
name	Character string. Library name (must be unique).
updatemode	Character string. How library updates are handled: "none" – no updates allowed; "instant" – updates take effect immediately; "approve" – updates require admin approval.
js_code	Character string. The JavaScript code for the library.

Value

A list containing the result of the RPC call (usually transaction ID).

See Also

[mc_add_library_update](#), [mc_list_libraries](#)

Other libraries: [mc_add_library_update\(\)](#), [mc_add_library_update_from\(\)](#), [mc_get_library_code\(\)](#), [mc_list_libraries\(\)](#), [mc_test_library\(\)](#)

Examples

```
## Not run:
js_code <- "function add(a, b) { return a + b; }"
mc_create_library(conn, "math", updatemode = "none", js_code)

## End(Not run)
```

mc_create_multisig	Create multi-signature address (external)
--------------------	---

Description

Creates a Pay-to-Script-Hash (P2SH) multi-signature address without adding it to the wallet. The address can be used in transactions that require multiple signatures, but the node cannot spend funds from it unless the private keys are also imported.

Usage

```
mc_create_multisig(conn, n_required, keys)
```

Arguments

conn	A connection object created by mc_connect .
n_required	Integer. Number of required signatures.
keys	Character vector. Public keys or addresses.

Value

A list with two components:

address	The multi-signature address.
redeemScript	The redeem script (needed for spending).

See Also

[mc_add_multisig_address](#) to create and add the address to the wallet.

Other addresses: [mc_add_multisig_address\(\)](#), [mc_create_keypairs\(\)](#), [mc_get_addresses\(\)](#), [mc_get_new_address\(\)](#), [mc_import_address\(\)](#), [mc_list_addresses\(\)](#), [mc_validate_address\(\)](#)

Examples

```
## Not run:
multisig <- mc_create_multisig(conn, n_required = 2,
                             keys = c("pubkey1", "pubkey2", "pubkey3"))
cat("Multisig address:", multisig$address)

## End(Not run)
```

mc_create_raw_exchange

Create a new atomic exchange transaction

Description

Initialises a partial atomic exchange transaction by specifying the first locked output and the desired assets/currency in return. This is the first step in constructing a multi-party atomic exchange.

Usage

```
mc_create_raw_exchange(conn, txid, vout, amounts)
```

Arguments

conn	A connection object created by mc_connect .
txid	Character string. Transaction ID of the locked output (obtained via mc_prepare_lock_unspent).
vout	Integer. Output index (vout) of the locked output.
amounts	A list specifying the assets or native currency asked for in exchange. Format: <code>list(asset_name = quantity, ...)</code> or a numeric value for native currency.

Value

Character string. Raw partial transaction in hexadecimal.

See Also

[mc_prepare_lock_unspent](#), [mc_append_raw_exchange](#)

Other atomic exchange: [mc_append_raw_exchange\(\)](#), [mc_complete_raw_exchange\(\)](#), [mc_decode_raw_exchange\(\)](#), [mc_disable_raw_transaction\(\)](#), [mc_prepare_lock_unspent\(\)](#), [mc_prepare_lock_unspent_from\(\)](#)

Examples

```
## Not run:
# First, lock some output
locked <- mc_prepare_lock_unspent(conn, amounts = list(myasset = 10))
# Create the exchange offer
offer <- mc_create_raw_exchange(conn, locked$txid, locked$vout,
                               amounts = list(otherasset = 5))

## End(Not run)
```

mc_create_raw_send_from

Create and fund a raw transaction from a specific address

Description

Creates a raw transaction that is automatically funded from a specified address. This is a convenience function that selects UTXOs from the given address and builds the transaction.

Usage

```
mc_create_raw_send_from(
  conn,
  from_address,
  to_amounts,
  data = list(),
  action = ""
)
```

Arguments

conn	A connection object created by mc_connect .
from_address	Character string. The address that will fund the transaction.
to_amounts	A named list mapping recipient addresses to amounts (e.g., <code>list("1A..." = 0.5, "1B..." = list(asset = 10))</code>).
data	Optional array of metadata (strings or lists; will be hex-encoded).
action	Optional action: "send", "sign", etc.

Value

A character string (raw transaction hex) or a list with hex and complete if signing is requested.

See Also

[mc_create_raw_transaction](#)

Other raw transactions: [mc_append_raw_change\(\)](#), [mc_append_raw_data\(\)](#), [mc_append_raw_transaction\(\)](#), [mc_create_raw_transaction\(\)](#), [mc_decode_raw_transaction\(\)](#), [mc_send_raw_transaction\(\)](#), [mc_sign_raw_transaction\(\)](#)

Examples

```
## Not run:
# Send 1.0 native coin to address
tx_hex <- mc_create_raw_send_from(conn, "1A...", list("1B..." = 1.0))

# Send asset and metadata
```

```
tx_hex <- mc_create_raw_send_from(conn, "1A...",
                                list("1B..." = list(myasset = 50)),
                                data = list("payment", list(ref = 123)))

## End(Not run)
```

```
mc_create_raw_transaction
      Create a raw transaction
```

Description

Creates a raw (unsigned) transaction from a list of inputs and outputs. This is the first step in building a custom transaction before signing and broadcasting.

Usage

```
mc_create_raw_transaction(conn, inputs, outputs, data = list(), action = "")
```

Arguments

conn	A connection object created by mc_connect .
inputs	A list of input objects, each containing: • txid – Transaction ID of the UTXO. • vout – Output index (vout) of the UTXO.
outputs	A named list (or list of named lists) mapping addresses to amounts. Example: <code>list("address1" = 0.5, "address2" = list(asset = 10))</code> .
data	Optional array of metadata. Each element can be a string or a list (which will be converted to JSON then hex). The data is embedded in the transaction outputs.
action	Optional action string: "lock" (lock inputs), "sign" (sign the transaction), "lock,sign" (both), or "send" (sign and send). Default is "" (just create).

Value

A character string containing the raw transaction hex (if no action) or a list with hex and complete if action includes signing.

See Also

[mc_sign_raw_transaction](#), [mc_send_raw_transaction](#)

Other raw transactions: [mc_append_raw_change\(\)](#), [mc_append_raw_data\(\)](#), [mc_append_raw_transaction\(\)](#), [mc_create_raw_send_from\(\)](#), [mc_decode_raw_transaction\(\)](#), [mc_send_raw_transaction\(\)](#), [mc_sign_raw_transaction\(\)](#)

Examples

```
## Not run:
# Build a simple transaction
inputs <- list(list(txid = "abc...", vout = 0))
outputs <- list("1A..." = 1.0)
tx_hex <- mc_create_raw_transaction(conn, inputs, outputs)

# With metadata
tx_hex <- mc_create_raw_transaction(conn, inputs, outputs,
                                   data = list("Hello", list(key = "value")))

## End(Not run)
```

mc_create_stream	Create a new stream
------------------	---------------------

Description

Creates a new stream on the blockchain. Streams are ordered collections of key-value items that can be used for data storage, messaging, or other applications. The stream can be open (anyone can write) or restricted.

Usage

```
mc_create_stream(conn, name, open = TRUE, custom_fields = NULL)
```

Arguments

conn	A connection object created by mc_connect .
name	Character string. Name of the stream (must be unique).
open	Either a logical (TRUE for open stream, FALSE for restricted) or a list of parameters, e.g., <code>list(restrict = "write")</code> . For open streams, any address with send permission can publish. For restricted, only addresses with write permission on the stream can publish.
custom_fields	Optional list of custom fields (e.g., <code>list(field1 = "value")</code>).

Value

A character string containing the transaction ID (txid) of the stream creation.

See Also

[mc_create_stream_from](#) to specify the creator address, [mc_get_stream_info](#) to query stream details.

Other streams: [mc_create_stream_from\(\)](#), [mc_get_stream_info\(\)](#), [mc_get_stream_item\(\)](#), [mc_get_stream_key_summary\(\)](#), [mc_get_stream_publisher_summary\(\)](#), [mc_list_stream_block_items\(\)](#),

```
mc_list_stream_items(), mc_list_stream_key_items(), mc_list_stream_keys(), mc_list_stream_publisher_items(),
mc_list_stream_publishers(), mc_list_stream_query_items(), mc_list_stream_tx_items(),
mc_list_streams(), mc_publish(), mc_publish_from(), mc_publish_multi(), mc_publish_multi_from()
```

Examples

```
## Not run:
# Create an open stream
txid <- mc_create_stream(conn, "mystream", open = TRUE)

# Create a restricted stream with custom fields
txid <- mc_create_stream(conn, "private", open = list(restrict = "write"),
                      custom_fields = list(owner = "admin"))

## End(Not run)
```

mc_create_stream_filter

Create a stream filter

Description

Creates a new stream filter on the blockchain. Stream filters are JavaScript programs that can be attached to streams to validate or transform items.

Usage

```
mc_create_stream_filter(conn, name, options, js_code)
```

Arguments

conn	A connection object created by mc_connect .
name	Character string. Name of the filter (must be unique).
options	List of filter options. Typically includes <code>libraries</code> (list of library names) that the filter depends on. Example: <code>list(libraries = list("mylib"))</code> .
js_code	Character string. The JavaScript code for the filter.

Value

A list containing the result of the RPC call (usually transaction ID).

See Also

[mc_create_tx_filter](#), [mc_list_stream_filters](#)

Other filters: [mc_approve_from\(\)](#), [mc_create_tx_filter\(\)](#), [mc_create_upgrade\(\)](#), [mc_get_filter_code\(\)](#), [mc_list_stream_filters\(\)](#), [mc_list_tx_filters\(\)](#), [mc_list_upgrades\(\)](#), [mc_run_stream_filter\(\)](#), [mc_run_tx_filter\(\)](#), [mc_test_stream_filter\(\)](#), [mc_test_tx_filter\(\)](#)

Examples

```
## Not run:
js_code <- "function filter(stream, item) { return true; }"
mc_create_stream_filter(conn, "myfilter", list(libraries = list()), js_code)

## End(Not run)
```

mc_create_stream_from *Create a new stream from a specific address*

Description

Creates a stream from a specified address (the address pays for the transaction). This is useful when the node has multiple addresses and you want to control which address appears as the creator.

Usage

```
mc_create_stream_from(
  conn,
  from_address,
  name,
  open = TRUE,
  custom_fields = NULL
)
```

Arguments

conn	A connection object created by mc_connect .
from_address	Character string. The address that will create the stream.
name	Character string. Stream name.
open	Either logical or a list of parameters (see mc_create_stream).
custom_fields	Optional list of custom fields.

Value

A character string containing the transaction ID.

See Also

[mc_create_stream](#)

Other streams: [mc_create_stream\(\)](#), [mc_get_stream_info\(\)](#), [mc_get_stream_item\(\)](#), [mc_get_stream_key_summary\(\)](#), [mc_get_stream_publisher_summary\(\)](#), [mc_list_stream_block_items\(\)](#), [mc_list_stream_items\(\)](#), [mc_list_stream_key_items\(\)](#), [mc_list_stream_keys\(\)](#), [mc_list_stream_publisher_items\(\)](#), [mc_list_stream_publishers\(\)](#), [mc_list_stream_query_items\(\)](#), [mc_list_stream_tx_items\(\)](#), [mc_list_streams\(\)](#), [mc_publish\(\)](#), [mc_publish_from\(\)](#), [mc_publish_multi\(\)](#), [mc_publish_multi_from\(\)](#)

Examples

```
## Not run:
txid <- mc_create_stream_from(conn, "1A...", "mystream", open = TRUE)

## End(Not run)
```

mc_create_tx_filter *Create a transaction filter*

Description

Creates a new transaction filter on the blockchain. Transaction filters are JavaScript programs that validate or transform transactions.

Usage

```
mc_create_tx_filter(conn, name, options, js_code)
```

Arguments

conn	A connection object created by mc_connect .
name	Character string. Name of the filter (must be unique).
options	List of filter options. Usually includes for (target asset or stream) and libraries (list of library names). Example: <code>list("for" = "asset1", libraries = list("lib1"))</code> .
js_code	Character string. The JavaScript code for the filter.

Value

A list containing the result of the RPC call (usually transaction ID).

See Also

[mc_create_stream_filter](#), [mc_list_tx_filters](#)

Other filters: [mc_approve_from\(\)](#), [mc_create_stream_filter\(\)](#), [mc_create_upgrade\(\)](#), [mc_get_filter_code\(\)](#), [mc_list_stream_filters\(\)](#), [mc_list_tx_filters\(\)](#), [mc_list_upgrades\(\)](#), [mc_run_stream_filter\(\)](#), [mc_run_tx_filter\(\)](#), [mc_test_stream_filter\(\)](#), [mc_test_tx_filter\(\)](#)

Examples

```
## Not run:
js_code <- "function filter(tx) { return tx.vin.length > 0; }"
mc_create_tx_filter(conn, "myfilter", list("for" = "asset1"), js_code)

## End(Not run)
```

mc_create_upgrade	Create a blockchain upgrade
-------------------	-----------------------------

Description

Creates a new upgrade proposal to change blockchain parameters (e.g., target block time, maximum block size). Upgrades require admin approval.

Usage

```
mc_create_upgrade(conn, name, params)
```

Arguments

conn	A connection object created by mc_connect .
name	Character string. Name of the upgrade (must be unique).
params	List of parameters to upgrade, e.g., <code>list("target-block-time" = 20, "max-block-size" = 10000000)</code> .

Value

A list containing the result of the RPC call (usually transaction ID).

See Also

[mc_approve_from](#) to approve an upgrade.

Other filters: [mc_approve_from\(\)](#), [mc_create_stream_filter\(\)](#), [mc_create_tx_filter\(\)](#), [mc_get_filter_code\(\)](#), [mc_list_stream_filters\(\)](#), [mc_list_tx_filters\(\)](#), [mc_list_upgrades\(\)](#), [mc_run_stream_filter\(\)](#), [mc_run_tx_filter\(\)](#), [mc_test_stream_filter\(\)](#), [mc_test_tx_filter\(\)](#)

Examples

```
## Not run:
mc_create_upgrade(conn, "speedup", list("target-block-time" = 20))

## End(Not run)
```

mc_create_variable	Create a new variable
--------------------	-----------------------

Description

Creates a global variable on the blockchain. Variables are key-value stores that can be read by filters and transactions.

Usage

```
mc_create_variable(conn, name, open = TRUE, value = NULL)
```

Arguments

conn	A connection object created by mc_connect .
name	Character string. Variable name (must be unique).
open	Logical. If TRUE, anyone with create permissions can edit the variable. If FALSE, only admins can edit.
value	Optional. Initial JSON value (list, number, string, etc.).

Value

A list containing the result of the RPC call (usually transaction ID).

See Also

[mc_set_variable_value](#), [mc_get_variable_value](#)

Other variables: [mc_create_variable_from\(\)](#), [mc_get_variable_history\(\)](#), [mc_get_variable_info\(\)](#), [mc_get_variable_value\(\)](#), [mc_list_variables\(\)](#), [mc_set_variable_value\(\)](#), [mc_set_variable_value_from\(\)](#)

Examples

```
## Not run:
mc_create_variable(conn, "myvar", open = TRUE, value = list(key = "value"))

## End(Not run)
```

`mc_create_variable_from`*Create a variable from a specific address*

Description

Creates a global variable, specifying the address that issues the transaction.

Usage

```
mc_create_variable_from(conn, from_address, name, open = TRUE, value = NULL)
```

Arguments

<code>conn</code>	A connection object created by mc_connect .
<code>from_address</code>	Character string. Address that pays for and creates the variable.
<code>name</code>	Character string. Variable name.
<code>open</code>	Logical. If TRUE, anyone with create permissions can edit the variable.
<code>value</code>	Optional. Initial JSON value.

Value

A list containing the result of the RPC call (usually transaction ID).

See Also

Other variables: [mc_create_variable\(\)](#), [mc_get_variable_history\(\)](#), [mc_get_variable_info\(\)](#), [mc_get_variable_value\(\)](#), [mc_list_variables\(\)](#), [mc_set_variable_value\(\)](#), [mc_set_variable_value_from\(\)](#)

Examples

```
## Not run:  
mc_create_variable_from(conn, "1A...", "myvar", open = TRUE, value = 42)  
  
## End(Not run)
```

`mc_decode_raw_exchange`*Decode a raw exchange transaction*

Description

Parses a raw atomic exchange transaction (partial or complete) and returns a human-readable representation of its structure, including the involved inputs, outputs, and the assets being exchanged.

Usage

```
mc_decode_raw_exchange(conn, tx_hex, verbose = FALSE)
```

Arguments

<code>conn</code>	A connection object created by mc_connect .
<code>tx_hex</code>	Character string. Hexadecimal representation of the exchange transaction.
<code>verbose</code>	Logical. If TRUE, lists all individual stages (contributions) of the exchange. Default is FALSE.

Value

A list (or a data frame if verbose) containing the decoded exchange details.

See Also

[mc_create_raw_exchange](#)

Other atomic exchange: [mc_append_raw_exchange\(\)](#), [mc_complete_raw_exchange\(\)](#), [mc_create_raw_exchange\(\)](#), [mc_disable_raw_transaction\(\)](#), [mc_prepare_lock_unspent\(\)](#), [mc_prepare_lock_unspent_from\(\)](#)

Examples

```
## Not run:
decoded <- mc_decode_raw_exchange(conn, my_tx_hex)
print(decoded)

## End(Not run)
```

`mc_decode_raw_transaction`*Decode a raw transaction hex*

Description

Parses a raw transaction hex string into a human-readable structure, showing inputs, outputs, amounts, metadata, and other transaction details.

Usage

```
mc_decode_raw_transaction(conn, tx_hex)
```

Arguments

<code>conn</code>	A connection object created by mc_connect .
<code>tx_hex</code>	Character string. The raw transaction hex to decode.

Value

A list with decoded transaction information (inputs, outputs, etc.).

See Also

[mc_get_raw_transaction](#)

Other raw transactions: [mc_append_raw_change\(\)](#), [mc_append_raw_data\(\)](#), [mc_append_raw_transaction\(\)](#), [mc_create_raw_send_from\(\)](#), [mc_create_raw_transaction\(\)](#), [mc_send_raw_transaction\(\)](#), [mc_sign_raw_transaction\(\)](#)

Examples

```
## Not run:
decoded <- mc_decode_raw_transaction(conn, tx_hex)
print(decoded$vin)

## End(Not run)
```

`mc_delete_binary_cache`*Delete an item from the binary cache*

Description

Removes a previously created binary cache item. Once deleted, the identifier becomes invalid and cannot be used further.

Usage

```
mc_delete_binary_cache(conn, identifier)
```

Arguments

<code>conn</code>	A connection object created by mc_connect .
<code>identifier</code>	Character string. The cache item identifier to remove.

Value

Invisibly returns NULL on success; throws an error if the item does not exist or cannot be deleted.

See Also

[mc_create_binary_cache](#), [mc_append_binary_cache](#)

Other binary cache: [mc_append_binary_cache\(\)](#), [mc_create_binary_cache\(\)](#), [mc_txout_to_binary_cache\(\)](#)

Examples

```
## Not run:  
id <- mc_create_binary_cache(conn)  
# ... use the cache item ...  
mc_delete_binary_cache(conn, id)  
  
## End(Not run)
```

`mc_disable_raw_transaction`*Disable an offer of exchange*

Description

Invalidates a previously created partial exchange transaction, preventing it from being completed. The transaction is replaced with a disabling transaction that spends the locked output(s) back to the original owner(s).

Usage

```
mc_disable_raw_transaction(conn, tx_hex)
```

Arguments

<code>conn</code>	A connection object created by mc_connect .
<code>tx_hex</code>	Character string. Hexadecimal representation of the exchange transaction to disable.

Value

Character string. Transaction ID of the disabling transaction.

See Also

[mc_prepare_lock_unspent](#)

Other atomic exchange: [mc_append_raw_exchange\(\)](#), [mc_complete_raw_exchange\(\)](#), [mc_create_raw_exchange\(\)](#), [mc_decode_raw_exchange\(\)](#), [mc_prepare_lock_unspent\(\)](#), [mc_prepare_lock_unspent_from\(\)](#)

Examples

```
## Not run:  
# After creating an offer, but before completion, you may decide to cancel  
disable_txid <- mc_disable_raw_transaction(conn, offer_hex)  
  
## End(Not run)
```

mc_dump_privkey	<i>Dump a private key for an address</i>
-----------------	--

Description

Dump a private key for an address

Usage

mc_dump_privkey(conn, address)

Arguments

- conn A connection object to the MultiChain node.
- address Character. The wallet address for which to retrieve the private key.

Value

Character. The private key in Wallet Import Format (WIF).

See Also

Other wallet: [mc_backup_wallet\(\)](#), [mc_change_wallet_passphrase\(\)](#), [mc_combine_unspent\(\)](#), [mc_dump_wallet\(\)](#), [mc_encrypt_wallet\(\)](#), [mc_get_wallet_info\(\)](#), [mc_import_privkey\(\)](#), [mc_import_wallet\(\)](#), [mc_list_lock_unspent\(\)](#), [mc_list_unspent\(\)](#), [mc_lock_unspent\(\)](#), [mc_lock_wallet\(\)](#), [mc_unlock_wallet\(\)](#)

mc_dump_wallet	<i>Dump all private keys to a file</i>
----------------	--

Description

Exports all wallet private keys into a human-readable text file.

Usage

mc_dump_wallet(conn, filename)

Arguments

- conn A connection object to the MultiChain node.
- filename Character. Full path for the text file on the node’s machine.

Value

Invisibly returns the RPC result (typically NULL) on success.

See Also

Other wallet: [mc_backup_wallet\(\)](#), [mc_change_wallet_passphrase\(\)](#), [mc_combine_unspent\(\)](#), [mc_dump_privkey\(\)](#), [mc_encrypt_wallet\(\)](#), [mc_get_wallet_info\(\)](#), [mc_import_privkey\(\)](#), [mc_import_wallet\(\)](#), [mc_list_lock_unspent\(\)](#), [mc_list_unspent\(\)](#), [mc_lock_unspent\(\)](#), [mc_lock_wallet\(\)](#), [mc_unlock_wallet\(\)](#)

mc_encrypt_wallet	<i>Encrypt the wallet</i>
-------------------	---------------------------

Description

Encrypts the wallet with a passphrase for the first time.

Usage

```
mc_encrypt_wallet(conn, passphrase)
```

Arguments

conn	A connection object to the MultiChain node.
passphrase	Character. The new wallet password.

Value

Invisibly returns the RPC result (a message indicating the node is shutting down).

Warning

MultiChain will shut down after this command is successful. You must manually restart the node to continue operations.

See Also

Other wallet: [mc_backup_wallet\(\)](#), [mc_change_wallet_passphrase\(\)](#), [mc_combine_unspent\(\)](#), [mc_dump_privkey\(\)](#), [mc_dump_wallet\(\)](#), [mc_get_wallet_info\(\)](#), [mc_import_privkey\(\)](#), [mc_import_wallet\(\)](#), [mc_list_lock_unspent\(\)](#), [mc_list_unspent\(\)](#), [mc_lock_unspent\(\)](#), [mc_lock_wallet\(\)](#), [mc_unlock_wallet\(\)](#)

mc_get_added_node_info

Get information about manually added nodes

Description

Returns details about nodes that were added via [mc_add_node](#). Can return either a list of node addresses or detailed information.

Usage

```
mc_get_added_node_info(conn, verbose = FALSE, node = NULL)
```

Arguments

conn	A connection object created by mc_connect .
verbose	Logical. If TRUE, returns a data frame with detailed information about each added node (e.g., connected status, last connection time). If FALSE (default), returns a character vector of node addresses.
node	Optional character string. If provided, returns information only for that specific node.

Value

If verbose = FALSE: a character vector of node addresses. If verbose = TRUE: a data frame (via `rpc_res_to_df`) with node details.

See Also

[mc_add_node](#), [mc_get_peer_info](#)

Other networking: [mc_add_node\(\)](#), [mc_get_network_info\(\)](#), [mc_get_peer_info\(\)](#), [mc_list_stored_nodes\(\)](#), [mc_ping\(\)](#), [mc_store_node\(\)](#)

Examples

```
## Not run:
# List all added nodes (addresses only)
nodes <- mc_get_added_node_info(conn)

# Get detailed information for a specific node
details <- mc_get_added_node_info(conn, verbose = TRUE, node = "192.168.1.10:8571")

## End(Not run)
```

mc_get_address_balances

Get asset balances for a specific address

Description

Returns a list of all asset balances for a given address in the node's wallet.

Usage

```
mc_get_address_balances(conn, address, minconf = 1, include_locked = FALSE)
```

Arguments

conn	A connection object to the MultiChain node.
address	Character. The MultiChain address to query.
minconf	Integer. The minimum number of confirmations (default 1).
include_locked	Logical. If TRUE, includes unspent outputs that have been locked (default FALSE).

Value

A data frame containing asset names, amounts, and other balance details.

See Also

Other transactions: [mc_get_address_transaction\(\)](#), [mc_get_multi_balances\(\)](#), [mc_get_token_balances\(\)](#), [mc_get_total_balances\(\)](#), [mc_get_tx_out_data\(\)](#), [mc_get_wallet_transaction\(\)](#), [mc_list_address_transactions\(\)](#), [mc_list_wallet_transactions\(\)](#), [mc_send\(\)](#), [mc_send_asset\(\)](#), [mc_send_asset_from\(\)](#), [mc_send_from\(\)](#), [mc_send_with_data\(\)](#), [mc_send_with_data_from\(\)](#)

mc_get_address_transaction

Get details of a transaction for a specific address

Description

Provides information about a specific transaction, but only if it involves the specified address.

Usage

```
mc_get_address_transaction(conn, address, txid, verbose = FALSE)
```

Arguments

conn	A connection object to the MultiChain node.
address	Character. The MultiChain address to query.
txid	Character. The transaction ID.
verbose	Logical. If TRUE, provides a detailed breakdown of inputs and outputs (default FALSE).

Value

A list containing transaction details.

See Also

Other transactions: [mc_get_address_balances\(\)](#), [mc_get_multi_balances\(\)](#), [mc_get_token_balances\(\)](#), [mc_get_total_balances\(\)](#), [mc_get_tx_out_data\(\)](#), [mc_get_wallet_transaction\(\)](#), [mc_list_address_transactions\(\)](#), [mc_list_wallet_transactions\(\)](#), [mc_send\(\)](#), [mc_send_asset\(\)](#), [mc_send_asset_from\(\)](#), [mc_send_from\(\)](#), [mc_send_with_data\(\)](#), [mc_send_with_data_from\(\)](#)

mc_get_addresses	<i>Get node wallet addresses</i>
------------------	----------------------------------

Description

Returns all addresses owned by the current node. If verbose = TRUE, detailed information (including balances and transactions) is returned.

Usage

mc_get_addresses(conn, verbose = FALSE)

Arguments

conn	A connection object created by mc_connect .
verbose	Logical. If TRUE, returns a list with detailed information for each address. If FALSE (default), returns a character vector of addresses.

Value

If verbose = FALSE: a character vector of addresses. If verbose = TRUE: a list (or data frame) with details.

See Also

[mc_list_addresses](#) for more flexible listing options.
Other addresses: [mc_add_multisig_address\(\)](#), [mc_create_keypairs\(\)](#), [mc_create_multisig\(\)](#), [mc_get_new_address\(\)](#), [mc_import_address\(\)](#), [mc_list_addresses\(\)](#), [mc_validate_address\(\)](#)

Examples

```
## Not run:
# Get all addresses (simple list)
addresses <- mc_get_addresses(conn)

# Get detailed information
details <- mc_get_addresses(conn, verbose = TRUE)

## End(Not run)
```

mc_get_asset_info	<i>Get information about a specific asset</i>
-------------------	---

Description

Retrieves details about an asset on the MultiChain blockchain. The asset can be identified by its name, reference, or issuance transaction ID.

Usage

```
mc_get_asset_info(conn, asset, verbose = FALSE)
```

Arguments

conn	A connection object created by mc_connect .
asset	Character string. Asset name, reference, or issuance transaction ID.
verbose	Logical. If TRUE, returns details about individual issuances (for fungible assets with multiple issuances). Default is FALSE.

Value

A list (or data frame, depending on verbosity) containing asset information such as name, type, total quantity, units, etc.

See Also

[mc_list_assets](#) to list all assets, [mc_list_asset_issues](#) to list issuances.

Other assets: [mc_get_token_info\(\)](#), [mc_issue\(\)](#), [mc_issue_from\(\)](#), [mc_issue_more\(\)](#), [mc_issue_more_from\(\)](#), [mc_issue_token\(\)](#), [mc_issue_token_from\(\)](#), [mc_list_asset_issues\(\)](#), [mc_list_assets\(\)](#), [mc_update\(\)](#), [mc_update_from\(\)](#)

Examples

```
## Not run:
# Get basic info about an asset named "mycoin"
info <- mc_get_asset_info(conn, "mycoin")
print(info$name)

# Get verbose info with issuance details
info_verbose <- mc_get_asset_info(conn, "mycoin", verbose = TRUE)

## End(Not run)
```

mc_get_asset_transaction

Get a specific transaction involving a subscribed asset

Description

Returns details of a single transaction that affects a subscribed asset.

Usage

```
mc_get_asset_transaction(conn, asset, txid, verbose = FALSE)
```

Arguments

conn	A connection object created by mc_connect .
asset	Character string. Subscribed asset name, reference, or issuance ID.
txid	Character string. Transaction ID.
verbose	Logical. If TRUE, includes additional details.

Value

A list (or data frame) with transaction details, including inputs, outputs, and asset movements.

See Also

[mc_list_asset_transactions](#) to list multiple transactions.

Other asset transactions: [mc_list_asset_transactions\(\)](#)

Examples

```
## Not run:
# Get details of a specific transaction
tx <- mc_get_asset_transaction(conn, "mycoin", "txid...")

## End(Not run)
```

mc_get_block	<i>Get block information</i>
--------------	------------------------------

Description

Retrieves detailed information about a specific block. The block can be identified either by its hash (string) or height (integer). The verbosity level controls how much detail is returned.

Usage

```
mc_get_block(conn, hash_or_height, verbose = 1)
```

Arguments

conn	A connection object created by mc_connect .
hash_or_height	Either a character string (block hash) or an integer (block height).
verbose	Integer. Verbosity level from 0 to 4. Default is 1. • 0: returns only the block hash as a string. • 1: returns a list with basic block information. • 2-4: include additional details (transactions, etc.).

Value

Depends on verbose:

- If verbose = 0: a character string with the block hash.
- If verbose >= 1: a list containing block details (height, time, transactions, etc.).

See Also

[mc_get_block_hash](#) to obtain a block hash from height, [mc_list_blocks](#) to list multiple blocks.

Other blockchain information: [mc_get_block_hash\(\)](#), [mc_get_blockchain_info\(\)](#), [mc_get_chain_totals\(\)](#), [mc_get_last_block_info\(\)](#), [mc_list_blocks\(\)](#), [mc_list_miners\(\)](#)

Examples

```
## Not run:
# Get block by height
block <- mc_get_block(conn, 123456)

# Get block by hash with full transaction details
block <- mc_get_block(conn, "0000...", verbose = 2)

## End(Not run)
```

mc_get_block_hash	<i>Get block hash by height</i>
-------------------	---------------------------------

Description

Returns the hash of a block at a given height.

Usage

```
mc_get_block_hash(conn, height)
```

Arguments

conn	A connection object created by mc_connect .
height	Integer. The block height (0 for genesis block).

Value

A character string containing the block hash.

See Also

[mc_get_block](#) to retrieve block details.

Other blockchain information: [mc_get_block\(\)](#), [mc_get_blockchain_info\(\)](#), [mc_get_chain_totals\(\)](#), [mc_get_last_block_info\(\)](#), [mc_list_blocks\(\)](#), [mc_list_miners\(\)](#)

Examples

```
## Not run:
hash <- mc_get_block_hash(conn, 0) # genesis block hash

## End(Not run)
```

mc_get_blockchain_info	<i>Get general blockchain information</i>
------------------------	---

Description

Returns global information about the blockchain, such as the current block height, chain name, protocol version, difficulty, and consensus status.

Usage

```
mc_get_blockchain_info(conn)
```

Arguments

conn A connection object created by [mc_connect](#).

Value

A list with blockchain metadata. Typical fields:

chain	Name of the chain.
blocks	Current block height.
headers	Number of block headers.
bestblockhash	Hash of the most recent block.
difficulty	Current mining difficulty.
chainwork	Total work in the chain.

See Also

[mc_get_chain_totals](#) for counts of entities, [mc_get_last_block_info](#) for the most recent block.

Other blockchain information: [mc_get_block\(\)](#), [mc_get_block_hash\(\)](#), [mc_get_chain_totals\(\)](#), [mc_get_last_block_info\(\)](#), [mc_list_blocks\(\)](#), [mc_list_miners\(\)](#)

Examples

```
## Not run:
info <- mc_get_blockchain_info(conn)
print(info$blocks)

## End(Not run)
```

mc_get_blockchain_params

Get blockchain parameters

Description

Returns the parameters that were used to initialize this blockchain. These are fixed at chain creation and cannot be changed later.

Usage

```
mc_get_blockchain_params(conn)
```

Arguments

conn A connection object created by [mc_connect](#).

Value

A list containing blockchain configuration parameters, such as:

protocolversion	Protocol version.
targetblocktime	Target time between blocks (seconds).
maxblocksize	Maximum block size (bytes).
...	Other chain-specific parameters.

See Also

[mc_get_runtime_params](#) for modifiable parameters.
Other node configuration: [mc_get_runtime_params\(\)](#), [mc_set_runtime_param\(\)](#)

Examples

```
## Not run:
params <- mc_get_blockchain_params(conn)
print(params$targetblocktime)

## End(Not run)
```

mc_get_chain_totals	<i>Get counts of blockchain entities</i>
---------------------	--

Description

Returns the total number of various objects in the blockchain, such as addresses, transactions, assets, streams, and permissions.

Usage

```
mc_get_chain_totals(conn)
```

Arguments

conn	A connection object created by mc_connect .
------	---

Value

A list with counts, typically containing:

addresses	Number of addresses.
transactions	Number of transactions.
assets	Number of assets.
streams	Number of streams.
permissions	Number of permission entries.

See Also

[mc_get_blockchain_info](#) for global blockchain stats.

Other blockchain information: [mc_get_block\(\)](#), [mc_get_block_hash\(\)](#), [mc_get_blockchain_info\(\)](#), [mc_get_last_block_info\(\)](#), [mc_list_blocks\(\)](#), [mc_list_miners\(\)](#)

Examples

```
## Not run:
totals <- mc_get_chain_totals(conn)
print(totals$transactions)

## End(Not run)
```

mc_get_chunk_queue_info

Get information about off-chain chunk queue

Description

Returns details about the node's off-chain chunk queue, which handles the transmission of large data items that are split into chunks.

Usage

```
mc_get_chunk_queue_info(conn)
```

Arguments

conn A connection object created by [mc_connect](#).

Value

A list containing:

chunk_count	Number of chunks currently queued.
byte_count	Total size in bytes of queued chunks.
...	Other queue statistics.

See Also

[mc_get_chunk_queue_totals](#) for cumulative statistics.

Other off-chain data: [mc_get_chunk_queue_totals\(\)](#)

Examples

```
## Not run:
queue_info <- mc_get_chunk_queue_info(conn)
cat("Chunks pending:", queue_info$chunk_count)

## End(Not run)
```

mc_get_chunk_queue_totals

Get cumulative statistics on off-chain chunk requests

Description

Returns total counts of chunk deliveries, failures, and timeouts since the node started.

Usage

```
mc_get_chunk_queue_totals(conn)
```

Arguments

conn A connection object created by [mc_connect](#).

Value

A list with cumulative statistics:

delivered	Number of chunks successfully delivered.
undelivered	Number of chunks not yet delivered.
timeouts	Number of delivery timeouts.
...	Other totals.

See Also

[mc_get_chunk_queue_info](#) for current queue state.

Other off-chain data: [mc_get_chunk_queue_info\(\)](#)

Examples

```
## Not run:
totals <- mc_get_chunk_queue_totals(conn)
print(totals)

## End(Not run)
```

mc_get_config	<i>Get MultiChain configuration</i>
---------------	-------------------------------------

Description

Reads the configuration parameters (RPC user, password, port) for a given blockchain from the MultiChain data directory. The function automatically determines the platform-specific base directory, but a custom base can be supplied for testing.

Usage

```
mc_get_config(chain_name, base_dir = NULL)
```

Arguments

chain_name	Character string. Name of the MultiChain blockchain.
base_dir	Optional character string. Base directory where MultiChain stores blockchain data. If NULL (default), the platform-specific default is used: • Windows: %APPDATA%/MultiChain • macOS: ~/Library/Application Support/MultiChain • Linux/other: ~/.multichain

Value

A list with four components:

user	RPC username (from multichain.conf).
password	RPC password (from multichain.conf).
port	RPC port number (integer).
host	Always "127.0.0.1" (hard-coded).

See Also

[mc_connect](#) to create a connection object using the returned configuration.

Examples

```
## Not run:  
# Get configuration for a chain called "my_chain"  
config <- mc_get_config("my_chain")  
print(config)  
  
## End(Not run)
```

mc_get_filter_code	<i>Get JavaScript code of a filter</i>
--------------------	--

Description

Retrieves the JavaScript code (and associated metadata) of an existing stream or transaction filter.

Usage

```
mc_get_filter_code(conn, filter)
```

Arguments

conn	A connection object created by mc_connect .
filter	Character string. Filter name or transaction ID.

Value

A list containing the filter's code and details.

See Also

[mc_get_library_code](#) for libraries.

Other filters: [mc_approve_from\(\)](#), [mc_create_stream_filter\(\)](#), [mc_create_tx_filter\(\)](#), [mc_create_upgrade\(\)](#), [mc_list_stream_filters\(\)](#), [mc_list_tx_filters\(\)](#), [mc_list_upgrades\(\)](#), [mc_run_stream_filter\(\)](#), [mc_run_tx_filter\(\)](#), [mc_test_stream_filter\(\)](#), [mc_test_tx_filter\(\)](#)

Examples

```
## Not run:
code_info <- mc_get_filter_code(conn, "myfilter")
print(code_info$code)

## End(Not run)
```

mc_get_info	<i>Get general node information</i>
-------------	-------------------------------------

Description

Returns comprehensive information about the node's status, including version, protocol, network connections, balance, and mining status.

Usage

```
mc_get_info(conn)
```

Arguments

conn A connection object created by [mc_connect](#).

Value

A list with node information, typically including:

version	Node software version.
protocolversion	Protocol version.
walletversion	Wallet version.
balance	Node's wallet balance.
blocks	Current block height.
timeoffset	Time offset from network.
connections	Number of active connections.
...	Other status details.

See Also

[mc_get_blockchain_info](#) for blockchain-level info, [mc_get_runtime_params](#) for runtime settings.

Other node information: [mc_get_init_status\(\)](#)

Examples

```
## Not run:
info <- mc_get_info(conn)
cat("Balance:", info$balance, "\n")
cat("Blocks:", info$blocks, "\n")

## End(Not run)
```

mc_get_init_status	<i>Get node initialization status</i>
--------------------	---------------------------------------

Description

Returns information about the node's initialization progress, especially useful during startup when the node is still syncing or loading the wallet.

Usage

```
mc_get_init_status(conn)
```

Arguments

conn A connection object created by [mc_connect](#).

Value

A list with:

status Character string describing the current state (e.g., "Loading blockchain", "Synchronizing", "Ready").

progress Numeric value between 0 and 1 indicating initialization progress (1 = fully initialized).

See Also

[mc_get_info](#) for general node status.

Other node information: [mc_get_info\(\)](#)

Examples

```
## Not run:
init <- mc_get_init_status(conn)
while (init$progress < 1) {
  cat("Init progress:", init$progress, "\n")
  Sys.sleep(5)
  init <- mc_get_init_status(conn)
}

## End(Not run)
```

mc_get_last_block_info

Get information about the last block

Description

Retrieves details of the most recent block, optionally skipping back by a number of blocks.

Usage

```
mc_get_last_block_info(conn, skip = 0)
```

Arguments

conn A connection object created by [mc_connect](#).

skip Integer. Number of blocks to skip back from the tip. skip = 0 returns the latest block, skip = 1 returns the previous block, etc. Default is 0.

Value

A list with block information (similar to [mc_get_block](#)).

See Also

[mc_get_block](#) for general block retrieval.

Other blockchain information: [mc_get_block\(\)](#), [mc_get_block_hash\(\)](#), [mc_get_blockchain_info\(\)](#), [mc_get_chain_totals\(\)](#), [mc_list_blocks\(\)](#), [mc_list_miners\(\)](#)

Examples

```
## Not run:
latest <- mc_get_last_block_info(conn)
previous <- mc_get_last_block_info(conn, skip = 1)

## End(Not run)
```

mc_get_library_code	<i>Get JavaScript code for a library</i>
---------------------	--

Description

Retrieves the JavaScript code of a library. By default, returns the active code. If updatename is provided, returns the code of that specific update (or the initial code if updatename = "").

Usage

```
mc_get_library_code(conn, library, updatename = NULL)
```

Arguments

conn	A connection object created by mc_connect .
library	Character string. Library name or transaction ID.
updatename	Optional character string. Update name. If omitted, returns the active code. Use "" to retrieve the initial code.

Value

A list containing the library code and metadata.

See Also

[mc_get_filter_code](#) for filters.

Other libraries: [mc_add_library_update\(\)](#), [mc_add_library_update_from\(\)](#), [mc_create_library\(\)](#), [mc_list_libraries\(\)](#), [mc_test_library\(\)](#)

Examples

```
## Not run:
active_code <- mc_get_library_code(conn, "math")
initial_code <- mc_get_library_code(conn, "math", updatename = "")

## End(Not run)
```

mc_get_mempool_info	<i>Get memory pool information</i>
---------------------	------------------------------------

Description

Returns information about the node's memory pool (mempool), which holds unconfirmed transactions awaiting inclusion in a block.

Usage

```
mc_get_mempool_info(conn)
```

Arguments

conn A connection object created by [mc_connect](#).

Value

A list with mempool statistics:

size	Number of transactions in the mempool.
bytes	Total size in bytes.
usage	Memory usage.

See Also

[mc_get_raw_mempool](#) for the list of transaction IDs.

Other mempool & transactions: [mc_get_raw_mempool\(\)](#), [mc_get_raw_transaction\(\)](#), [mc_get_tx_out\(\)](#)

Examples

```
## Not run:
mempool <- mc_get_mempool_info(conn)
print(paste("Pending transactions:", mempool$size))

## End(Not run)
```

mc_get_multi_balances *Get balances for multiple addresses and assets*

Description

Returns a breakdown of balances across a set of addresses and/or assets.

Usage

```
mc_get_multi_balances(
  conn,
  addresses = "*",
  assets = "*",
  minconf = 1,
  include_watch_only = FALSE,
  include_locked = FALSE
)
```

Arguments

conn	A connection object to the MultiChain node.
addresses	A vector of addresses, or "*" for all addresses in the wallet (default "*").
assets	A vector of asset names/refs, or "*" for all assets (default "*").
minconf	Integer. Minimum confirmations (default 1).
include_watch_only	Logical. Include watch-only addresses (default FALSE).
include_locked	Logical. Include locked unspent outputs (default FALSE).

Value

A list or data frame of balances, indexed by address and asset.

See Also

Other transactions: [mc_get_address_balances\(\)](#), [mc_get_address_transaction\(\)](#), [mc_get_token_balances\(\)](#), [mc_get_total_balances\(\)](#), [mc_get_tx_out_data\(\)](#), [mc_get_wallet_transaction\(\)](#), [mc_list_address_transaction](#), [mc_list_wallet_transactions\(\)](#), [mc_send\(\)](#), [mc_send_asset\(\)](#), [mc_send_asset_from\(\)](#), [mc_send_from\(\)](#), [mc_send_with_data\(\)](#), [mc_send_with_data_from\(\)](#)

mc_get_network_info	<i>Get information about node's network status</i>
---------------------	--

Description

Returns details about the node's network configuration, including listening port, local addresses, and network-related flags.

Usage

```
mc_get_network_info(conn)
```

Arguments

conn	A connection object created by mc_connect .
------	---

Value

A list containing network information, typically:

version	Node version.
subversion	Node subversion string.
protocolversion	Protocol version.
localservices	Services offered by the node.
localaddresses	List of local IP addresses.
timeoffset	Time offset from network.
connections	Number of active connections.
relayfee	Minimum relay fee.
...	Other network parameters.

See Also

[mc_get_peer_info](#) for peer details.

Other networking: [mc_add_node\(\)](#), [mc_get_added_node_info\(\)](#), [mc_get_peer_info\(\)](#), [mc_list_stored_nodes\(\)](#), [mc_ping\(\)](#), [mc_store_node\(\)](#)

Examples

```
## Not run:
net_info <- mc_get_network_info(conn)
cat("Listening port:", net_info$localaddresses[[1]]$port)

## End(Not run)
```

mc_get_new_address	<i>Create new wallet address</i>
--------------------	----------------------------------

Description

Generates a new address (with a private key) and adds it to the node's wallet.

Usage

```
mc_get_new_address(conn)
```

Arguments

conn	A connection object created by mc_connect .
------	---

Value

A character string with the newly created address.

See Also

[mc_create_keypairs](#) to generate key pairs not stored in the wallet.

Other addresses: [mc_add_multisig_address\(\)](#), [mc_create_keypairs\(\)](#), [mc_create_multisig\(\)](#), [mc_get_addresses\(\)](#), [mc_import_address\(\)](#), [mc_list_addresses\(\)](#), [mc_validate_address\(\)](#)

Examples

```
## Not run:
new_addr <- mc_get_new_address(conn)
print(new_addr)

## End(Not run)
```

mc_get_peer_info	<i>Get information about connected peers</i>
------------------	--

Description

Returns detailed information about each peer currently connected to the node.

Usage

```
mc_get_peer_info(conn)
```

Arguments

conn A connection object created by [mc_connect](#).

Value

A data frame (via `rpc_res_to_df`) with one row per peer. Common columns include:

addr	Peer address and port.
addrlocal	Local address used for the connection.
services	Services offered.
lastsend	Last time a message was sent.
lastrecv	Last time a message was received.
bytessent	Total bytes sent.
bytesrecv	Total bytes received.
conntime	Connection start time.
pingtime	Ping time (seconds).
version	Peer's version.
subver	Peer's subversion string.
inbound	Whether the peer connected inbound.

See Also

[mc_ping](#) to measure latency, [mc_get_network_info](#) for node network status.

Other networking: [mc_add_node\(\)](#), [mc_get_added_node_info\(\)](#), [mc_get_network_info\(\)](#), [mc_list_stored_nodes\(\)](#), [mc_ping\(\)](#), [mc_store_node\(\)](#)

Examples

```
## Not run:
peers <- mc_get_peer_info(conn)
print(head(peers))

## End(Not run)
```

mc_get_raw_mempool	<i>Get list of transaction IDs in mempool</i>
--------------------	---

Description

Returns a character vector of transaction IDs currently in the node's memory pool.

Usage

```
mc_get_raw_mempool(conn)
```

Arguments

conn A connection object created by [mc_connect](#).

Value

A character vector of transaction IDs (txids).

See Also

[mc_get_mempool_info](#) for mempool statistics.

Other mempool & transactions: [mc_get_mempool_info\(\)](#), [mc_get_raw_transaction\(\)](#), [mc_get_tx_out\(\)](#)

Examples

```
## Not run:
pending <- mc_get_raw_mempool(conn)
length(pending) # number of pending transactions

## End(Not run)
```

mc_get_raw_transaction

Get a raw transaction from the blockchain

Description

Retrieves a transaction from the blockchain by its transaction ID. If verbose = TRUE, returns a detailed decoded transaction; if FALSE (default), returns the raw transaction in hexadecimal.

Usage

```
mc_get_raw_transaction(conn, txid, verbose = FALSE)
```

Arguments

conn A connection object created by [mc_connect](#).

txid Character string. Transaction ID.

verbose Logical. If TRUE, returns a parsed transaction object; if FALSE, returns the raw transaction hex. Default is FALSE.

Value

If verbose = FALSE, a character string (hex). If verbose = TRUE, a list with transaction details.

See Also

[mc_get_tx_out](#) to inspect a specific output.

Other mempool & transactions: [mc_get_mempool_info\(\)](#), [mc_get_raw_mempool\(\)](#), [mc_get_tx_out\(\)](#)

Examples

```
## Not run:
# Get raw hex of a transaction
raw <- mc_get_raw_transaction(conn, "abc...")

# Get decoded transaction
tx <- mc_get_raw_transaction(conn, "abc...", verbose = TRUE)

## End(Not run)
```

mc_get_runtime_params *Get node runtime parameters*

Description

Returns the current runtime parameters of the node. These can be changed while the node is running (see [mc_set_runtime_param](#)).

Usage

```
mc_get_runtime_params(conn)
```

Arguments

conn A connection object created by [mc_connect](#).

Value

A list of runtime parameters, e.g.:

mining	Logical; whether mining is enabled.
maxconnections	Maximum number of inbound connections.
...	Other runtime settings.

See Also

[mc_set_runtime_param](#) to modify parameters, [mc_get_blockchain_params](#) for fixed chain parameters.

Other node configuration: [mc_get_blockchain_params\(\)](#), [mc_set_runtime_param\(\)](#)

Examples

```
## Not run:
runtime <- mc_get_runtime_params(conn)
print(runtime$mining)

## End(Not run)
```

mc_get_stream_info	<i>Get information about a specific stream</i>
--------------------	--

Description

Returns metadata about a stream, including its creation transaction, open/restricted status, and optionally the creator address.

Usage

```
mc_get_stream_info(conn, stream, verbose = FALSE)
```

Arguments

conn	A connection object created by mc_connect .
stream	Character string. Stream name, reference, or creation transaction ID.
verbose	Logical. If TRUE, includes the creator address.

Value

A list with stream details (name, ref, open, txid, etc.).

See Also

[mc_list_streams](#) to list all streams.

Other streams: [mc_create_stream\(\)](#), [mc_create_stream_from\(\)](#), [mc_get_stream_item\(\)](#), [mc_get_stream_key_summary\(\)](#), [mc_get_stream_publisher_summary\(\)](#), [mc_list_stream_block_items\(\)](#), [mc_list_stream_items\(\)](#), [mc_list_stream_key_items\(\)](#), [mc_list_stream_keys\(\)](#), [mc_list_stream_publisher_items\(\)](#), [mc_list_stream_publishers\(\)](#), [mc_list_stream_query_items\(\)](#), [mc_list_stream_tx_items\(\)](#), [mc_list_streams\(\)](#), [mc_publish\(\)](#), [mc_publish_from\(\)](#), [mc_publish_multi\(\)](#), [mc_publish_multi_from\(\)](#)

Examples

```
## Not run:
info <- mc_get_stream_info(conn, "mystream")
print(info$open)

## End(Not run)
```

mc_get_stream_item	<i>Get a specific item from a stream</i>
--------------------	--

Description

Retrieves a single stream item by its transaction ID.

Usage

```
mc_get_stream_item(conn, stream, txid, verbose = FALSE)
```

Arguments

conn	A connection object created by mc_connect .
stream	Character string. Stream name, reference, or txid.
txid	Character string. Transaction ID of the item.
verbose	Logical. If TRUE, includes additional details.

Value

A list with item details (key, data, publisher, etc.).

See Also

[mc_list_stream_items](#) to list items.

Other streams: [mc_create_stream\(\)](#), [mc_create_stream_from\(\)](#), [mc_get_stream_info\(\)](#), [mc_get_stream_key_summary\(\)](#), [mc_get_stream_publisher_summary\(\)](#), [mc_list_stream_block_items\(\)](#), [mc_list_stream_items\(\)](#), [mc_list_stream_key_items\(\)](#), [mc_list_stream_keys\(\)](#), [mc_list_stream_publisher_items\(\)](#), [mc_list_stream_publishers\(\)](#), [mc_list_stream_query_items\(\)](#), [mc_list_stream_tx_items\(\)](#), [mc_list_streams\(\)](#), [mc_publish\(\)](#), [mc_publish_from\(\)](#), [mc_publish_multi\(\)](#), [mc_publish_multi_from\(\)](#)

Examples

```
## Not run:
item <- mc_get_stream_item(conn, "mystream", "txid...")
print(item$data)

## End(Not run)
```

mc_get_stream_key_summary

Get summary of JSON objects in a stream for a specific key

Description

Aggregates JSON objects published under a specific key, merging them according to the specified mode.

Usage

```
mc_get_stream_key_summary(conn, stream, key, mode = "jsonobjectmerge")
```

Arguments

conn	A connection object created by mc_connect .
stream	Character string. Stream name, reference, or txid.
key	Character string. The key to summarize.
mode	Character string. Merge mode (default "jsonobjectmerge"). Other options include "recursive", "noupdate", etc.

Value

A JSON object (list) representing the merged summary.

See Also

[mc_get_stream_publisher_summary](#)

Other streams: [mc_create_stream\(\)](#), [mc_create_stream_from\(\)](#), [mc_get_stream_info\(\)](#), [mc_get_stream_item\(\)](#), [mc_get_stream_publisher_summary\(\)](#), [mc_list_stream_block_items\(\)](#), [mc_list_stream_items\(\)](#), [mc_list_stream_key_items\(\)](#), [mc_list_stream_keys\(\)](#), [mc_list_stream_publisher_items\(\)](#), [mc_list_stream_publishers\(\)](#), [mc_list_stream_query_items\(\)](#), [mc_list_stream_tx_items\(\)](#), [mc_list_streams\(\)](#), [mc_publish\(\)](#), [mc_publish_from\(\)](#), [mc_publish_multi\(\)](#), [mc_publish_multi_from\(\)](#)

Examples

```
## Not run:
summary <- mc_get_stream_key_summary(conn, "mystream", "key", mode = "jsonobjectmerge")

## End(Not run)
```

mc_get_stream_publisher_summary

Get summary of JSON objects published by a specific address

Description

Aggregates JSON objects published by a given address in a stream.

Usage

```
mc_get_stream_publisher_summary(
  conn,
  stream,
  address,
  mode = "jsonobjectmerge"
)
```

Arguments

conn	A connection object created by mc_connect .
stream	Character string. Stream name, reference, or txid.
address	Character string. Publisher address.
mode	Character string. Merge mode (default "jsonobjectmerge").

Value

A JSON object (list) representing the merged summary.

See Also

Other streams: [mc_create_stream\(\)](#), [mc_create_stream_from\(\)](#), [mc_get_stream_info\(\)](#), [mc_get_stream_item\(\)](#), [mc_get_stream_key_summary\(\)](#), [mc_list_stream_block_items\(\)](#), [mc_list_stream_items\(\)](#), [mc_list_stream_key_items\(\)](#), [mc_list_stream_keys\(\)](#), [mc_list_stream_publisher_items\(\)](#), [mc_list_stream_publishers\(\)](#), [mc_list_stream_query_items\(\)](#), [mc_list_stream_tx_items\(\)](#), [mc_list_streams\(\)](#), [mc_publish\(\)](#), [mc_publish_from\(\)](#), [mc_publish_multi\(\)](#), [mc_publish_multi_from\(\)](#)

Examples

```
## Not run:
summary <- mc_get_stream_publisher_summary(conn, "mystream", "1A...")

## End(Not run)
```

mc_get_token_balances *Get balances for non-fungible tokens (NFTs)*

Description

Specifically retrieves balances for tokens (sub-assets or individual units) associated with a parent asset.

Usage

```
mc_get_token_balances(
  conn,
  addresses = "*",
  assets = "*",
  minconf = 1,
  include_watch_only = FALSE,
  include_locked = FALSE
)
```

Arguments

conn	A connection object to the MultiChain node.
addresses	A vector of addresses, or "*" (default "*").
assets	A vector of parent asset names/refs, or "*" (default "*").
minconf	Integer. Minimum confirmations (default 1).
include_watch_only	Logical. Include watch-only addresses (default FALSE).
include_locked	Logical. Include locked outputs (default FALSE).

Value

A data frame containing token-level balance details.

See Also

Other transactions: [mc_get_address_balances\(\)](#), [mc_get_address_transaction\(\)](#), [mc_get_multi_balances\(\)](#), [mc_get_total_balances\(\)](#), [mc_get_tx_out_data\(\)](#), [mc_get_wallet_transaction\(\)](#), [mc_list_address_transaction\(\)](#), [mc_list_wallet_transactions\(\)](#), [mc_send\(\)](#), [mc_send_asset\(\)](#), [mc_send_asset_from\(\)](#), [mc_send_from\(\)](#), [mc_send_with_data\(\)](#), [mc_send_with_data_from\(\)](#)

mc_get_token_info	<i>Get information about a specific token (MultiChain 2.2.1+)</i>
-------------------	---

Description

Retrieves details about a token (non-fungible) within a parent asset. This function requires MultiChain version 2.2.1 or later.

Usage

```
mc_get_token_info(conn, asset, token, verbose = FALSE)
```

Arguments

conn	A connection object created by mc_connect .
asset	Character string. Parent asset name, reference, or issuance transaction ID.
token	Character string. Token name or index (e.g., "token0").
verbose	Logical. If TRUE, returns detailed information, including token details. Default is FALSE.

Value

A list with token information, such as name, quantity, and (if verbose) custom fields.

See Also

[mc_issue_token](#) to issue tokens, [mc_list_assets](#) to list parent assets.

Other assets: [mc_get_asset_info\(\)](#), [mc_issue\(\)](#), [mc_issue_from\(\)](#), [mc_issue_more\(\)](#), [mc_issue_more_from\(\)](#), [mc_issue_token\(\)](#), [mc_issue_token_from\(\)](#), [mc_list_asset_issues\(\)](#), [mc_list_assets\(\)](#), [mc_update\(\)](#), [mc_update_from\(\)](#)

Examples

```
## Not run:
# Get basic info about token "nft1" in asset "art"
token_info <- mc_get_token_info(conn, "art", "nft1")

## End(Not run)
```

mc_get_total_balances	<i>Get total wallet balances</i>
-----------------------	----------------------------------

Description

Returns the total balance of all assets across all addresses in the wallet.

Usage

```
mc_get_total_balances(  
    conn,  
    minconf = 1,  
    include_watch_only = FALSE,  
    include_locked = FALSE  
)
```

Arguments

- conn A connection object to the MultiChain node.
- minconf Integer. Minimum confirmations (default 1).
- include_watch_only Logical. Include watch-only addresses (default FALSE).
- include_locked Logical. Include locked outputs (default FALSE).

Value

A data frame summarizing total balances for each asset.

See Also

Other transactions: [mc_get_address_balances\(\)](#), [mc_get_address_transaction\(\)](#), [mc_get_multi_balances\(\)](#), [mc_get_token_balances\(\)](#), [mc_get_tx_out_data\(\)](#), [mc_get_wallet_transaction\(\)](#), [mc_list_address_transactions\(\)](#), [mc_list_wallet_transactions\(\)](#), [mc_send\(\)](#), [mc_send_asset\(\)](#), [mc_send_asset_from\(\)](#), [mc_send_from\(\)](#), [mc_send_with_data\(\)](#), [mc_send_with_data_from\(\)](#)

mc_get_tx_out	<i>Get details about an unspent transaction output</i>
---------------	--

Description

Returns information about a specific unspent transaction output (UTXO). This can be useful for building transactions or checking balances.

Usage

```
mc_get_tx_out(conn, txid, vout, unconfirmed = FALSE)
```

Arguments

conn	A connection object created by mc_connect .
txid	Character string. Transaction ID.
vout	Integer. Output index.
unconfirmed	Logical. If TRUE, includes unconfirmed transactions (from the mempool). Default is FALSE.

Value

A list with output details, including:

bestblock	Hash of the best block.
confirmations	Number of confirmations.
value	Amount (in native currency).
scriptPubKey	Output script details.

Returns NULL if the output does not exist or is spent.

See Also

[mc_get_raw_transaction](#) for full transaction details.

Other mempool & transactions: [mc_get_mempool_info\(\)](#), [mc_get_raw_mempool\(\)](#), [mc_get_raw_transaction\(\)](#)

Examples

```
## Not run:
# Check an output from a confirmed transaction
out <- mc_get_tx_out(conn, "abc...", vout = 0)
if (!is.null(out)) print(out$value)

## End(Not run)
```

mc_get_tx_out_data	<i>Retrieve full hex data from a transaction output</i>
--------------------	---

Description

This function is used to retrieve large data payloads (like stream data) that may have been truncated in standard transaction calls.

Usage

```
mc_get_tx_out_data(conn, txid, vout, count_bytes = NULL, start_byte = 0)
```

Arguments

conn	A connection object to the MultiChain node.
txid	Character. The transaction ID containing the output.
vout	Integer. The index of the output (starting from 0).
count_bytes	Integer. The number of bytes to retrieve. If NULL, returns all data.
start_byte	Integer. The byte offset to start reading from (default 0).

Value

A list or string containing the hex data from the transaction output.

See Also

Other transactions: [mc_get_address_balances\(\)](#), [mc_get_address_transaction\(\)](#), [mc_get_multi_balances\(\)](#), [mc_get_token_balances\(\)](#), [mc_get_total_balances\(\)](#), [mc_get_wallet_transaction\(\)](#), [mc_list_address_transactions\(\)](#), [mc_list_wallet_transactions\(\)](#), [mc_send\(\)](#), [mc_send_asset\(\)](#), [mc_send_asset_from\(\)](#), [mc_send_from\(\)](#), [mc_send_with_data\(\)](#), [mc_send_with_data_from\(\)](#)

mc_get_variable_history

List historical values of a variable

Description

Retrieves the update history of a variable, showing previous values and their transaction IDs.

Usage

```
mc_get_variable_history(
    conn,
    variable,
    verbose = FALSE,
    count = 10,
    start = NULL
)
```

Arguments

conn	A connection object created by mc_connect .
variable	Character string. Variable name or transaction ID.
verbose	Logical. If TRUE, returns detailed entries.
count	Integer. Number of historical entries to return (default 10).
start	Optional integer. Offset (positive for forward, negative for backward). If omitted, the most recent entries are returned.

Value

A data frame (via `rpc_res_to_df`) with history entries.

See Also

Other variables: `mc_create_variable()`, `mc_create_variable_from()`, `mc_get_variable_info()`, `mc_get_variable_value()`, `mc_list_variables()`, `mc_set_variable_value()`, `mc_set_variable_value_from()`

Examples

```
## Not run:
history <- mc_get_variable_history(conn, "myvar", count = 5)

## End(Not run)
```

`mc_get_variable_info` *Get information about a variable*

Description

Returns metadata about a variable, such as creator, open status, creation time.

Usage

```
mc_get_variable_info(conn, variable, verbose = FALSE)
```

Arguments

<code>conn</code>	A connection object created by <code>mc_connect</code> .
<code>variable</code>	Character string. Variable name or transaction ID.
<code>verbose</code>	Logical. If TRUE, includes additional details.

Value

A list with variable information.

See Also

Other variables: `mc_create_variable()`, `mc_create_variable_from()`, `mc_get_variable_history()`, `mc_get_variable_value()`, `mc_list_variables()`, `mc_set_variable_value()`, `mc_set_variable_value_from()`

Examples

```
## Not run:
info <- mc_get_variable_info(conn, "myvar")
print(info$open)

## End(Not run)
```

mc_get_variable_value *Retrieve the latest value of a variable*

Description

Returns the current value of a variable.

Usage

```
mc_get_variable_value(conn, variable)
```

Arguments

conn	A connection object created by mc_connect .
variable	Character string. Variable name or transaction ID.

Value

The current value (any JSON type).

See Also

[mc_get_variable_info](#), [mc_get_variable_history](#)

Other variables: [mc_create_variable\(\)](#), [mc_create_variable_from\(\)](#), [mc_get_variable_history\(\)](#), [mc_get_variable_info\(\)](#), [mc_list_variables\(\)](#), [mc_set_variable_value\(\)](#), [mc_set_variable_value_from\(\)](#)

Examples

```
## Not run:  
val <- mc_get_variable_value(conn, "myvar")  
  
## End(Not run)
```

mc_get_wallet_info *Get general wallet information*

Description

Returns metadata about the wallet, such as version, balance, and encryption status.

Usage

```
mc_get_wallet_info(conn)
```

Arguments

conn A connection object to the MultiChain node.

Value

A list of wallet information.

See Also

Other wallet: [mc_backup_wallet\(\)](#), [mc_change_wallet_passphrase\(\)](#), [mc_combine_unspent\(\)](#), [mc_dump_privkey\(\)](#), [mc_dump_wallet\(\)](#), [mc_encrypt_wallet\(\)](#), [mc_import_privkey\(\)](#), [mc_import_wallet\(\)](#), [mc_list_lock_unspent\(\)](#), [mc_list_unspent\(\)](#), [mc_lock_unspent\(\)](#), [mc_lock_wallet\(\)](#), [mc_unlock_wallet\(\)](#)

mc_get_wallet_transaction

Get details of a wallet transaction

Description

Retrieves detailed information about a transaction in the node's wallet.

Usage

```
mc_get_wallet_transaction(
    conn,
    txid,
    include_watch_only = FALSE,
    verbose = FALSE
)
```

Arguments

conn A connection object to the MultiChain node.

txid Character. The transaction ID.

include_watch_only Logical. Include watch-only addresses (default FALSE).

verbose Logical. If TRUE, provides details of inputs and outputs (default FALSE).

Value

A list containing detailed transaction data.

See Also

Other transactions: [mc_get_address_balances\(\)](#), [mc_get_address_transaction\(\)](#), [mc_get_multi_balances\(\)](#), [mc_get_token_balances\(\)](#), [mc_get_total_balances\(\)](#), [mc_get_tx_out_data\(\)](#), [mc_list_address_transactions\(\)](#), [mc_list_wallet_transactions\(\)](#), [mc_send\(\)](#), [mc_send_asset\(\)](#), [mc_send_asset_from\(\)](#), [mc_send_from\(\)](#), [mc_send_with_data\(\)](#), [mc_send_with_data_from\(\)](#)

mc_grant*Grant permissions to an address*

Description

Grants one or more permissions to a wallet address. Permissions control what actions an address can perform on the blockchain (e.g., connect, send, receive, mine, admin, etc.).

Usage

```
mc_grant(conn, address, permissions)
```

Arguments

conn	A connection object created by mc_connect .
address	Character string. The address that will receive the permissions.
permissions	Character string. A comma-separated list of permissions to grant, e.g., "connect, send, receive".

Value

A character string containing the transaction ID (txid) of the grant.

See Also

[mc_grant_from](#) to specify the grantor, [mc_revoke](#) to revoke permissions, [mc_list_permissions](#) to view permissions.

Other permissions: [mc_grant_from\(\)](#), [mc_grant_with_data\(\)](#), [mc_grant_with_data_from\(\)](#), [mc_list_permissions\(\)](#), [mc_revoke\(\)](#), [mc_revoke_from\(\)](#), [mc_verify_permission\(\)](#)

Examples

```
## Not run:  
# Grant connect and send permissions to an address  
txid <- mc_grant(conn, "1A...", "connect,send")  
  
## End(Not run)
```

mc_grant_from

Grant permissions from a specific address

Description

Grants permissions from a specified address (which must have admin or grant rights). This allows controlling which address pays for the transaction and acts as the grantor.

Usage

```
mc_grant_from(
    conn,
    from_address,
    to_address,
    permissions,
    native_amount = 0,
    start_block = NULL,
    end_block = NULL
)
```

Arguments

conn	A connection object created by mc_connect .
from_address	Character string. The address granting the permissions.
to_address	Character string. The address receiving the permissions.
permissions	Character string. Comma-separated list of permissions.
native_amount	Numeric. Amount of native currency to send along with the grant (default 0).
start_block	Optional integer. Block height from which the permission becomes valid. If NULL, the permission starts immediately.
end_block	Optional integer. Block height at which the permission expires. If NULL, the permission never expires.

Value

A character string containing the transaction ID.

See Also

[mc_grant](#), [mc_revoke_from](#).

Other permissions: [mc_grant\(\)](#), [mc_grant_with_data\(\)](#), [mc_grant_with_data_from\(\)](#), [mc_list_permissions\(\)](#), [mc_revoke\(\)](#), [mc_revoke_from\(\)](#), [mc_verify_permission\(\)](#)

Examples

```
## Not run:
# Grant permissions from a specific admin address
txid <- mc_grant_from(conn, "admin1...", "user1...", "send,receive")

# Grant with a validity window
txid <- mc_grant_from(conn, "admin1...", "user1...", "mine",
                      start_block = 1000, end_block = 2000)

## End(Not run)
```

mc_grant_with_data	<i>Grant permissions with metadata</i>
--------------------	--

Description

Grants permissions and attaches arbitrary data (metadata) to the transaction. The data can be text, JSON, or any hex-encoded value.

Usage

```
mc_grant_with_data(conn, to_address, permissions, data, native_amount = 0)
```

Arguments

conn	A connection object created by mc_connect .
to_address	Character string. The address receiving the permissions.
permissions	Character string. Comma-separated list of permissions.
data	Data to embed. Can be a character string (will be hex-encoded), a list (converted to JSON then hex), or raw binary.
native_amount	Numeric. Amount of native currency to send (default 0).

Value

A character string containing the transaction ID.

See Also

[mc_grant_with_data_from](#) to specify the grantor.

Other permissions: [mc_grant\(\)](#), [mc_grant_from\(\)](#), [mc_grant_with_data_from\(\)](#), [mc_list_permissions\(\)](#), [mc_revoke\(\)](#), [mc_revoke_from\(\)](#), [mc_verify_permission\(\)](#)

Examples

```
## Not run:
# Grant with a text note
txid <- mc_grant_with_data(conn, "1A...", "send",
                           data = "Welcome to the network!")

# Grant with JSON metadata
metadata <- list(reason = "partnership", level = "full")
txid <- mc_grant_with_data(conn, "1A...", "connect,send",
                           data = metadata, native_amount = 0.1)

## End(Not run)
```

mc_grant_with_data_from

Grant permissions from a specific address with metadata

Description

Grants permissions from a specified address and includes metadata. Combines the capabilities of [mc_grant_from](#) and [mc_grant_with_data](#).

Usage

```
mc_grant_with_data_from(
  conn,
  from_address,
  to_address,
  permissions,
  data,
  native_amount = 0
)
```

Arguments

conn	A connection object created by mc_connect .
from_address	Character string. The address granting the permissions.
to_address	Character string. The address receiving the permissions.
permissions	Character string. Comma-separated list of permissions.
data	Data to embed (string or list, automatically hex-encoded).
native_amount	Numeric. Amount of native currency to send (default 0).

Value

A character string containing the transaction ID.

See Also

[mc_grant_with_data](#), [mc_grant_from](#).

Other permissions: [mc_grant\(\)](#), [mc_grant_from\(\)](#), [mc_grant_with_data\(\)](#), [mc_list_permissions\(\)](#), [mc_revoke\(\)](#), [mc_revoke_from\(\)](#), [mc_verify_permission\(\)](#)

Examples

```
## Not run:
# Grant from a specific admin with metadata
txid <- mc_grant_with_data_from(conn, "admin1...", "user1...",
                                "send, receive",
                                data = list(note = "temporary access"),
                                native_amount = 0.01)

## End(Not run)
```

mc_help

Get help for MultiChain commands

Description

Returns a list of all available RPC commands, or detailed help for a specific command. The result is printed in a human-readable format.

Usage

```
mc_help(conn, command = NULL)
```

Arguments

conn	A connection object created by mc_connect .
command	Optional character string. The name of the command to get detailed help for. If NULL (default), returns a list of all commands.

Value

An object of class "mc_help" (inheriting from "character") that contains the help text and prints nicely via [print.mc_help](#).

Examples

```
## Not run:
# List all available commands
mc_help(conn)

# Get detailed help for the "getinfo" command
mc_help(conn, "getinfo")
```

```
## End(Not run)
```

mc_import_address	<i>Import a watch-only address</i>
-------------------	------------------------------------

Description

Adds an address (without a private key) to the node's wallet for monitoring. The node will be able to see transactions involving this address, but cannot spend funds from it.

Usage

```
mc_import_address(conn, address, label = "", rescan = TRUE)
```

Arguments

conn	A connection object created by mc_connect .
address	Character string. The wallet address to import.
label	Character string (optional). A label to assign to the address. Default is "" (no label).
rescan	Logical. If TRUE (default), the node will scan the blockchain for transactions associated with this address.

Value

Invisibly returns the RPC result (typically NULL) on success; throws an error if the import fails.

See Also

[mc_validate_address](#) to check address validity.

Other addresses: [mc_add_multisig_address\(\)](#), [mc_create_keypairs\(\)](#), [mc_create_multisig\(\)](#), [mc_get_addresses\(\)](#), [mc_get_new_address\(\)](#), [mc_list_addresses\(\)](#), [mc_validate_address\(\)](#)

Examples

```
## Not run:
mc_import_address(conn, "1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa",
                  label = "donation", rescan = TRUE)

## End(Not run)
```

mc_import_privkey	<i>Import one or more private keys</i>
-------------------	--

Description

Adds private keys to the node's wallet.

Usage

```
mc_import_privkey(conn, privkeys, label = "", rescan = TRUE)
```

Arguments

conn	A connection object to the MultiChain node.
privkeys	A character string or vector of private keys (WIF format).
label	Optional character string. A label to assign to the addresses.
rescan	Logical or Integer. If TRUE, rescans the entire blockchain. Can also be an integer representing the block height to start scanning from.

Value

Returns NULL on success.

See Also

Other wallet: [mc_backup_wallet\(\)](#), [mc_change_wallet_passphrase\(\)](#), [mc_combine_unspent\(\)](#), [mc_dump_privkey\(\)](#), [mc_dump_wallet\(\)](#), [mc_encrypt_wallet\(\)](#), [mc_get_wallet_info\(\)](#), [mc_import_wallet\(\)](#), [mc_list_lock_unspent\(\)](#), [mc_list_unspent\(\)](#), [mc_lock_unspent\(\)](#), [mc_lock_wallet\(\)](#), [mc_unlock_wallet\(\)](#)

mc_import_wallet	<i>Import keys from a wallet dump file</i>
------------------	--

Description

Import keys from a wallet dump file

Usage

```
mc_import_wallet(conn, filename, rescan = 0)
```

Arguments

conn	A connection object to the MultiChain node.
filename	Character. Path to the dump file on the node's machine.
rescan	Integer. The block number to start rescanning from (default 0).

Value

Returns NULL on success.

See Also

Other wallet: [mc_backup_wallet\(\)](#), [mc_change_wallet_passphrase\(\)](#), [mc_combine_unspent\(\)](#), [mc_dump_privkey\(\)](#), [mc_dump_wallet\(\)](#), [mc_encrypt_wallet\(\)](#), [mc_get_wallet_info\(\)](#), [mc_import_privkey\(\)](#), [mc_list_lock_unspent\(\)](#), [mc_list_unspent\(\)](#), [mc_lock_unspent\(\)](#), [mc_lock_wallet\(\)](#), [mc_unlock_wallet\(\)](#)

mc_issue	<i>Issue new asset</i>
----------	------------------------

Description

Creates a new asset on the MultiChain blockchain. The asset can be fungible or non-fungible, and its properties (e.g., open/restricted, divisible) are specified via the name parameter.

Usage

```
mc_issue(  
    conn,  
    address,  
    name,  
    quantity,  
    units = 1,  
    native_amount = NULL,  
    custom_fields = NULL  
)
```

Arguments

conn	A connection object created by mc_connect .
address	Character string. Wallet address that will receive the issued assets.
name	Either a character string (asset name) or a list of asset parameters (e.g., <code>list(name = "myasset", open = TRUE)</code>). See MultiChain documentation for supported parameters.
quantity	Numeric. Total amount to issue. For non-fungible assets, this is typically 1.
units	Numeric. The smallest divisible unit (e.g., 0.01 means the asset is divisible to two decimal places). Default is 1 (indivisible).
native_amount	Numeric (optional). Amount of native currency (coins) to send together with the asset issuance.
custom_fields	List (optional). Custom fields to attach to the asset.

Value

A character string containing the transaction ID of the issuance.

See Also

`mc_issue_from` to issue from a specific address, `mc_issue_more` to increase supply of a fungible asset.

Other assets: `mc_get_asset_info()`, `mc_get_token_info()`, `mc_issue_from()`, `mc_issue_more()`, `mc_issue_more_from()`, `mc_issue_token()`, `mc_issue_token_from()`, `mc_list_asset_issues()`, `mc_list_assets()`, `mc_update()`, `mc_update_from()`

Examples

```
## Not run:
# Issue a simple fungible asset
txid <- mc_issue(conn, "1A...", "mycoin", quantity = 1000, units = 0.01)

# Issue a restricted, non-fungible asset with custom fields
params <- list(name = "artwork", open = FALSE, restrict = TRUE)
txid <- mc_issue(conn, "1A...", params, quantity = 1, units = 1,
                 custom_fields = list(author = "Picasso"))

## End(Not run)
```

mc_issue_from	<i>Issue new asset from specific address</i>
---------------	--

Description

Issues a new asset, but allows specifying the sender address (which must have sufficient native currency to pay for the transaction). This is useful when the node has multiple addresses.

Usage

```
mc_issue_from(
  conn,
  from_address,
  to_address,
  name,
  quantity,
  units = 1,
  native_amount = NULL,
  custom_fields = NULL
)
```

Arguments

conn	A connection object created by <code>mc_connect</code> .
from_address	Character string. Address that will pay for and issue the asset.
to_address	Character string. Recipient address (can be the same as from_address).

name	Either a character string (asset name) or a list of asset parameters (e.g., <code>list(name = "myasset", open = TRUE)</code>).
quantity	Numeric. Total amount to issue.
units	Numeric. Smallest divisible unit. Default is 1.
native_amount	Numeric (optional). Amount of native currency to send along with the issuance.
custom_fields	List (optional). Custom fields.

Value

A character string containing the transaction ID.

See Also

[mc_issue](#) for simpler issuance, [mc_issue_more_from](#) to increase supply.

Other assets: [mc_get_asset_info\(\)](#), [mc_get_token_info\(\)](#), [mc_issue\(\)](#), [mc_issue_more\(\)](#), [mc_issue_more_from\(\)](#), [mc_issue_token\(\)](#), [mc_issue_token_from\(\)](#), [mc_list_asset_issues\(\)](#), [mc_list_assets\(\)](#), [mc_update\(\)](#), [mc_update_from\(\)](#)

Examples

```
## Not run:
# Issue from a specific address to the same address
txid <- mc_issue_from(conn, from_address = "1A...", to_address = "1A...",
                      name = "myasset", quantity = 100)

## End(Not run)
```

mc_issue_more	<i>Issue more of an existing fungible asset</i>
---------------	---

Description

Increases the supply of a previously issued fungible asset. The asset must be open for further issuances (i.e., its open property must be TRUE).

Usage

```
mc_issue_more(
  conn,
  address,
  asset,
  quantity,
  native_amount = NULL,
  custom_fields = NULL
)
```

Arguments

conn	A connection object created by mc_connect .
address	Character string. Address that will receive the newly issued units.
asset	Character string. Asset name, reference, or issuance transaction ID.
quantity	Numeric. Additional quantity to issue.
native_amount	Numeric (optional). Amount of native currency to send.
custom_fields	List (optional). Custom fields (overwrites existing ones if present).

Value

A character string containing the transaction ID.

See Also

[mc_issue](#) for initial issuance, [mc_issue_more_from](#) to issue from a specific address.

Other assets: [mc_get_asset_info\(\)](#), [mc_get_token_info\(\)](#), [mc_issue\(\)](#), [mc_issue_from\(\)](#), [mc_issue_more_from\(\)](#), [mc_issue_token\(\)](#), [mc_issue_token_from\(\)](#), [mc_list_asset_issues\(\)](#), [mc_list_assets\(\)](#), [mc_update\(\)](#), [mc_update_from\(\)](#)

Examples

```
## Not run:
# Increase supply of "mycoin" by 500 units
txid <- mc_issue_more(conn, "1A...", "mycoin", quantity = 500)

## End(Not run)
```

mc_issue_more_from	<i>Issue more of an asset from specific address</i>
--------------------	---

Description

Increases the supply of a fungible asset, specifying the address that pays for and initiates the transaction.

Usage

```
mc_issue_more_from(
  conn,
  from_address,
  to_address,
  asset,
  quantity,
  native_amount = NULL,
  custom_fields = NULL
)
```

Arguments

conn	A connection object created by mc_connect .
from_address	Character string. Address that will pay for and issue the additional units.
to_address	Character string. Recipient address.
asset	Character string. Asset name, reference, or issuance transaction ID.
quantity	Numeric. Additional quantity to issue.
native_amount	Numeric (optional). Amount of native currency to send.
custom_fields	List (optional). Custom fields (overwrites existing ones if present).

Value

A character string containing the transaction ID.

See Also

[mc_issue_more](#) for simpler usage, [mc_issue_from](#) for initial issuance.

Other assets: [mc_get_asset_info\(\)](#), [mc_get_token_info\(\)](#), [mc_issue\(\)](#), [mc_issue_from\(\)](#), [mc_issue_more\(\)](#), [mc_issue_token\(\)](#), [mc_issue_token_from\(\)](#), [mc_list_asset_issues\(\)](#), [mc_list_assets\(\)](#), [mc_update\(\)](#), [mc_update_from\(\)](#)

Examples

```
## Not run:
# Issue more from a specific address to another address
txid <- mc_issue_more_from(conn, "1A...", "1B...", "mycoin", quantity = 100)

## End(Not run)
```

mc_issue_token

Issue tokens for a non-fungible asset (NFT)

Description

Creates one or more tokens (non-fungible items) under a parent asset. The parent asset must be non-fungible (e.g., issued with type = "nonfungible").

Usage

```
mc_issue_token(
  conn,
  address,
  asset,
  token,
  quantity,
  native_amount = NULL,
  token_details = NULL
)
```

Arguments

conn	A connection object created by mc_connect .
address	Character string. Address that will receive the tokens.
asset	Character string. Parent asset name, reference, or issuance ID.
token	Character string. Token name (must be unique within the asset).
quantity	Numeric. Number of token units to issue (usually 1).
native_amount	Numeric (optional). Amount of native currency to send.
token_details	List (optional). Custom details for the token (e.g., <code>list(description = "My NFT")</code>).

Value

A character string containing the transaction ID.

See Also

[mc_get_token_info](#) to query token details, [mc_issue_token_from](#) for using a specific sender address.

Other assets: [mc_get_asset_info\(\)](#), [mc_get_token_info\(\)](#), [mc_issue\(\)](#), [mc_issue_from\(\)](#), [mc_issue_more\(\)](#), [mc_issue_more_from\(\)](#), [mc_issue_token_from\(\)](#), [mc_list_asset_issuances\(\)](#), [mc_list_assets\(\)](#), [mc_update\(\)](#), [mc_update_from\(\)](#)

Examples

```
## Not run:
# Issue a token "painting1" under asset "art"
txid <- mc_issue_token(conn, "1A...", "art", "painting1", quantity = 1,
                      token_details = list(artist = "Monet", year = 2025))

## End(Not run)
```

mc_issue_token_from	<i>Issue tokens from specific address</i>
---------------------	---

Description

Issues tokens (non-fungible) and specifies the sending address.

Usage

```
mc_issue_token_from(
  conn,
  from_address,
  to_address,
  asset,
  token,
  quantity,
  native_amount = NULL,
  token_details = NULL
)
```

Arguments

conn	A connection object created by mc_connect .
from_address	Character string. Address that pays for and issues the token.
to_address	Character string. Recipient address.
asset	Character string. Parent asset name, reference, or issuance ID.
token	Character string. Token name (must be unique within the asset).
quantity	Numeric. Number of token units to issue (usually 1).
native_amount	Numeric (optional). Amount of native currency to send.
token_details	List (optional). Custom details for the token (e.g., <code>list(description = "My NFT")</code>).

Value

A character string containing the transaction ID.

See Also

[mc_issue_token](#) for simpler usage.

Other assets: [mc_get_asset_info\(\)](#), [mc_get_token_info\(\)](#), [mc_issue\(\)](#), [mc_issue_from\(\)](#), [mc_issue_more\(\)](#), [mc_issue_more_from\(\)](#), [mc_issue_token\(\)](#), [mc_list_asset_issues\(\)](#), [mc_list_assets\(\)](#), [mc_update\(\)](#), [mc_update_from\(\)](#)

Examples

```
## Not run:
# Issue token from a specific address
txid <- mc_issue_token_from(conn, "1A...", "1B...", "art", "painting2",
                           quantity = 1, token_details = list(artist = "Picasso"))

## End(Not run)
```

`mc_list_address_transactions`*List transactions for a specific address*

Description

Returns a list of the most recent transactions involving the specified address.

Usage

```
mc_list_address_transactions(  
    conn,  
    address,  
    count = 10,  
    skip = 0,  
    verbose = FALSE  
)
```

Arguments

<code>conn</code>	A connection object to the MultiChain node.
<code>address</code>	Character. The MultiChain address to query.
<code>count</code>	Integer. The number of transactions to return (default 10).
<code>skip</code>	Integer. The number of transactions to skip (default 0).
<code>verbose</code>	Logical. If TRUE, provides details of inputs and outputs (default FALSE).

Value

A data frame of transaction history for the address.

See Also

Other transactions: [mc_get_address_balances\(\)](#), [mc_get_address_transaction\(\)](#), [mc_get_multi_balances\(\)](#), [mc_get_token_balances\(\)](#), [mc_get_total_balances\(\)](#), [mc_get_tx_out_data\(\)](#), [mc_get_wallet_transaction\(\)](#), [mc_list_wallet_transactions\(\)](#), [mc_send\(\)](#), [mc_send_asset\(\)](#), [mc_send_asset_from\(\)](#), [mc_send_from\(\)](#), [mc_send_with_data\(\)](#), [mc_send_with_data_from\(\)](#)

mc_list_addresses	<i>List addresses in the wallet</i>
-------------------	-------------------------------------

Description

Returns information about the addresses in the current node's wallet. This is a more flexible version of [mc_get_addresses](#), allowing filtering, pagination, and optional verbosity.

Usage

```
mc_list_addresses(
  conn,
  addresses = "*",
  verbose = FALSE,
  count = NULL,
  start = NULL
)
```

Arguments

conn	A connection object created by mc_connect .
addresses	A character vector of addresses to filter, or "*" (default) to list all addresses.
verbose	Logical. If TRUE, returns detailed information for each address.
count	Integer (optional). Maximum number of addresses to return.
start	Integer (optional). Offset for pagination.

Value

A data frame (created by [rpc_res_to_df](#)) containing address information. The exact columns depend on the verbose setting, but typically include address, label, balance, etc.

See Also

[mc_get_addresses](#) for a simpler version.

Other addresses: [mc_add_multisig_address\(\)](#), [mc_create_keypairs\(\)](#), [mc_create_multisig\(\)](#), [mc_get_addresses\(\)](#), [mc_get_new_address\(\)](#), [mc_import_address\(\)](#), [mc_validate_address\(\)](#)

Examples

```
## Not run:
# List all addresses (simple)
all_addr <- mc_list_addresses(conn)

# List detailed information for specific addresses
details <- mc_list_addresses(conn,
  addresses = c("1A...", "1B..."),
  verbose = TRUE)
```

```
# Paginate results
first_10 <- mc_list_addresses(conn, count = 10)
next_10 <- mc_list_addresses(conn, count = 10, start = 10)

## End(Not run)
```

mc_list_asset_issues *List issuance events for an asset*

Description

Returns a list of all issuance transactions (initial and subsequent) for a given asset.

Usage

```
mc_list_asset_issues(conn, asset, verbose = FALSE, count = NULL, start = NULL)
```

Arguments

conn	A connection object created by mc_connect .
asset	Character string. Asset name, reference, or issuance ID.
verbose	Logical. If TRUE, includes details about issuers and custom fields. Default is FALSE.
count	Integer (optional). Maximum number of issuances to return.
start	Integer (optional). Offset (positive for forward, negative for backward from the most recent). Use a negative value to get the most recent issuances first.

Value

A data frame (converted via `rpc_res_to_df`) with one row per issuance. Columns typically include txid, issuer, quantity, units, etc.

See Also

[mc_list_assets](#) to list assets, [mc_get_asset_info](#) for asset summary.

Other assets: [mc_get_asset_info\(\)](#), [mc_get_token_info\(\)](#), [mc_issue\(\)](#), [mc_issue_from\(\)](#), [mc_issue_more\(\)](#), [mc_issue_more_from\(\)](#), [mc_issue_token\(\)](#), [mc_issue_token_from\(\)](#), [mc_list_assets\(\)](#), [mc_update\(\)](#), [mc_update_from\(\)](#)

Examples

```
## Not run:
# Get all issuances of "mycoin"
issues <- mc_list_asset_issues(conn, "mycoin")

# Get the most recent 5 issuances with details
recent <- mc_list_asset_issues(conn, "mycoin", verbose = TRUE,
                              count = 5, start = -5)

## End(Not run)
```

mc_list_asset_transactions

List transactions involving a subscribed asset

Description

Returns a list of recent transactions that affect a subscribed asset.

Usage

```
mc_list_asset_transactions(
  conn,
  asset,
  verbose = FALSE,
  count = 10,
  start = NULL,
  local_ordering = FALSE
)
```

Arguments

conn	A connection object created by mc_connect .
asset	Character string. Subscribed asset name, reference, or issuance ID.
verbose	Logical. If TRUE, returns detailed information about each transaction.
count	Integer. Number of transactions to return (default 10).
start	Integer (optional). Offset (negative for most recent).
local_ordering	Logical. If TRUE, uses local node's transaction ordering (default FALSE).

Value

A data frame (via `rpc_res_to_df`) with transaction details.

See Also

[mc_get_asset_transaction](#) for a single transaction.

Other asset transactions: [mc_get_asset_transaction\(\)](#)

Examples

```
## Not run:
# Get the 10 most recent transactions
txs <- mc_list_asset_transactions(conn, "mycoin")

# Get next 5 transactions with details
more_txs <- mc_list_asset_transactions(conn, "mycoin", verbose = TRUE,
                                       count = 5, start = 10)

## End(Not run)
```

mc_list_assets	<i>List blockchain assets</i>
----------------	-------------------------------

Description

Returns a list of assets on the blockchain, with optional filtering and pagination.

Usage

```
mc_list_assets(conn, assets = "*", verbose = FALSE, count = NULL, start = NULL)
```

Arguments

conn	A connection object created by mc_connect .
assets	Asset filter. Can be a single asset name/ref/txid, a vector of such identifiers, or "*" (default) to list all assets.
verbose	Logical. If TRUE, returns detailed information (e.g., issuances, open status). Default is FALSE.
count	Integer (optional). Maximum number of assets to return.
start	Integer (optional). Offset for pagination.

Value

A data frame (via `rpc_res_to_df`) with asset information. If `verbose = FALSE`, columns include name, ref, issuetxid, etc. If `verbose = TRUE`, additional details like issuances and open status are included.

See Also

`mc_get_asset_info` for single asset details.

Other assets: `mc_get_asset_info()`, `mc_get_token_info()`, `mc_issue()`, `mc_issue_from()`, `mc_issue_more()`, `mc_issue_more_from()`, `mc_issue_token()`, `mc_issue_token_from()`, `mc_list_asset_issues()`, `mc_update()`, `mc_update_from()`

Examples

```
## Not run:
# List all assets
all_assets <- mc_list_assets(conn)

# Get details for a specific asset
asset_detail <- mc_list_assets(conn, assets = "mycoin", verbose = TRUE)

# List first 10 assets
first_10 <- mc_list_assets(conn, count = 10)

## End(Not run)
```

mc_list_blocks	List information about specific blocks
----------------	--

Description

Retrieves information about one or more blocks. The `blocks` parameter can be a single block height/hash, a range (e.g., "100-200"), or `~%d` to list the most recent blocks.

Usage

```
mc_list_blocks(conn, blocks, verbose = FALSE)
```

Arguments

conn	A connection object created by <code>mc_connect</code> .
blocks	Specification of which blocks to list. Can be: • a single block height (integer) or hash (string), • a range string like "100-200", • a negative integer <code>-n</code> to list the <code>n</code> most recent blocks.
verbose	Logical. If TRUE, returns detailed block information. Default is FALSE.

Value

A data frame (converted via `rpc_res_to_df`) with one row per block.

See Also

[mc_get_block](#) for a single block.

Other blockchain information: [mc_get_block\(\)](#), [mc_get_block_hash\(\)](#), [mc_get_blockchain_info\(\)](#), [mc_get_chain_totals\(\)](#), [mc_get_last_block_info\(\)](#), [mc_list_miners\(\)](#)

Examples

```
## Not run:
# List the last 5 blocks
last5 <- mc_list_blocks(conn, -5)

# List blocks 100 to 105
range <- mc_list_blocks(conn, "100-105", verbose = TRUE)

## End(Not run)
```

mc_list_libraries	<i>List libraries on the blockchain</i>
-------------------	---

Description

Returns a list of libraries with optional filtering and verbosity.

Usage

```
mc_list_libraries(conn, libraries = "*", verbose = FALSE)
```

Arguments

conn	A connection object created by mc_connect .
libraries	Character vector of library names/IDs, or "*" (default) for all libraries.
verbose	Logical. If TRUE, returns detailed information.

Value

A data frame (via `rpc_res_to_df`) with library information.

See Also

Other libraries: [mc_add_library_update\(\)](#), [mc_add_library_update_from\(\)](#), [mc_create_library\(\)](#), [mc_get_library_code\(\)](#), [mc_test_library\(\)](#)

Examples

```
## Not run:
libs <- mc_list_libraries(conn)

## End(Not run)
```

mc_list_lock_unspent *List locked unspent outputs*

Description

Returns a list of unspent transaction outputs that have been temporarily locked by [mc_lock_unspent](#).

Usage

```
mc_list_lock_unspent(conn)
```

Arguments

conn A connection object to the MultiChain node.

Value

A data frame containing columns for txid and vout.

See Also

Other wallet: [mc_backup_wallet\(\)](#), [mc_change_wallet_passphrase\(\)](#), [mc_combine_unspent\(\)](#), [mc_dump_privkey\(\)](#), [mc_dump_wallet\(\)](#), [mc_encrypt_wallet\(\)](#), [mc_get_wallet_info\(\)](#), [mc_import_privkey\(\)](#), [mc_import_wallet\(\)](#), [mc_list_unspent\(\)](#), [mc_lock_unspent\(\)](#), [mc_lock_wallet\(\)](#), [mc_unlock_wallet\(\)](#)

mc_list_miners *List miners and their status*

Description

Returns information about nodes that are mining (or have mining permission) on the blockchain.

Usage

```
mc_list_miners(conn, verbose = FALSE)
```

Arguments

conn	A connection object created by mc_connect .
verbose	Logical. If TRUE, returns additional details about each miner (e.g., mining status, last block mined). Default is FALSE.

Value

A data frame (via `rpc_res_to_df`) with miner information. Typical columns: address, status, lastblocktime, etc.

See Also

Other blockchain information: [mc_get_block\(\)](#), [mc_get_block_hash\(\)](#), [mc_get_blockchain_info\(\)](#), [mc_get_chain_totals\(\)](#), [mc_get_last_block_info\(\)](#), [mc_list_blocks\(\)](#)

Examples

```
## Not run:
miners <- mc_list_miners(conn)
print(miners)

## End(Not run)
```

mc_list_permissions	<i>List network permissions</i>
---------------------	---------------------------------

Description

Returns a list of all permissions (or filtered by type) currently active on the blockchain.

Usage

```
mc_list_permissions(conn, permissions = "*")
```

Arguments

conn	A connection object created by mc_connect .
permissions	Character string. Permission type to filter. Can be a single permission name (e.g., "send") or "*" (default) to list all permissions.

Value

A data frame (via `rpc_res_to_df`) with permission entries, typically containing columns like address, type, start, end, and txid.

See Also

[mc_grant](#), [mc_revoke](#).

Other permissions: [mc_grant\(\)](#), [mc_grant_from\(\)](#), [mc_grant_with_data\(\)](#), [mc_grant_with_data_from\(\)](#), [mc_revoke\(\)](#), [mc_revoke_from\(\)](#), [mc_verify_permission\(\)](#)

Examples

```
## Not run:
# List all permissions
all_perms <- mc_list_permissions(conn)

# List only "admin" permissions
admins <- mc_list_permissions(conn, "admin")

## End(Not run)
```

`mc_list_stored_nodes` *List known peer node addresses (MultiChain 2.3+)*

Description

Returns a list of nodes that the node has stored in its address manager. This includes peers that were connected to or manually added.

Usage

```
mc_list_stored_nodes(conn, include_old_ignores = FALSE)
```

Arguments

`conn` A connection object created by [mc_connect](#).

`include_old_ignores` Logical. If TRUE, includes nodes that were previously ignored. Default is FALSE.

Value

A data frame (via `rpc_res_to_df`) of stored nodes.

See Also

[mc_store_node](#) to add a node to the stored list.

Other networking: [mc_add_node\(\)](#), [mc_get_added_node_info\(\)](#), [mc_get_network_info\(\)](#), [mc_get_peer_info\(\)](#), [mc_ping\(\)](#), [mc_store_node\(\)](#)

Examples

```
## Not run:
nodes <- mc_list_stored_nodes(conn)

## End(Not run)
```

mc_list_stream_block_items

List items in a specific block or blocks

Description

Returns stream items that were published in one or more blocks.

Usage

```
mc_list_stream_block_items(
  conn,
  stream,
  blocks,
  verbose = FALSE,
  count = NULL,
  start = NULL
)
```

Arguments

conn	A connection object created by mc_connect .
stream	Character string. Stream name, reference, or txid.
blocks	A vector of block heights, block hashes, or timestamps.
verbose	Logical. If TRUE, returns detailed information.
count	Optional integer. Number of items to return.
start	Optional integer. Offset.

Value

A data frame (via [rpc_res_to_df](#)) with items.

See Also

Other streams: [mc_create_stream\(\)](#), [mc_create_stream_from\(\)](#), [mc_get_stream_info\(\)](#), [mc_get_stream_item\(\)](#), [mc_get_stream_key_summary\(\)](#), [mc_get_stream_publisher_summary\(\)](#), [mc_list_stream_items\(\)](#), [mc_list_stream_key_items\(\)](#), [mc_list_stream_keys\(\)](#), [mc_list_stream_publisher_items\(\)](#), [mc_list_stream_publishers\(\)](#), [mc_list_stream_query_items\(\)](#), [mc_list_stream_tx_items\(\)](#), [mc_list_streams\(\)](#), [mc_publish\(\)](#), [mc_publish_from\(\)](#), [mc_publish_multi\(\)](#), [mc_publish_multi_from\(\)](#)

Examples

```
## Not run:
# Items in block height 100
items <- mc_list_stream_block_items(conn, "mystream", 100)

# Items in multiple blocks
items <- mc_list_stream_block_items(conn, "mystream", c(100, 101, 102))

## End(Not run)
```

mc_list_stream_filters

List stream filters

Description

Returns a list of stream filters on the blockchain, with optional filtering and verbosity.

Usage

```
mc_list_stream_filters(conn, filters = "*", verbose = FALSE)
```

Arguments

conn	A connection object created by mc_connect .
filters	Character vector of filter names/IDs, or "*" (default) for all filters.
verbose	Logical. If TRUE, returns detailed information.

Value

A data frame (via `rpc_res_to_df`) with filter information.

See Also

[mc_create_stream_filter](#), [mc_list_tx_filters](#)

Other filters: [mc_approve_from\(\)](#), [mc_create_stream_filter\(\)](#), [mc_create_tx_filter\(\)](#), [mc_create_upgrade\(\)](#), [mc_get_filter_code\(\)](#), [mc_list_tx_filters\(\)](#), [mc_list_upgrades\(\)](#), [mc_run_stream_filter\(\)](#), [mc_run_tx_filter\(\)](#), [mc_test_stream_filter\(\)](#), [mc_test_tx_filter\(\)](#)

Examples

```
## Not run:
all_filters <- mc_list_stream_filters(conn)

## End(Not run)
```

mc_list_stream_items *List items in a stream (Enhanced version)*

Description

Returns a list of items published in a stream, with pagination and ordering.

Usage

```
mc_list_stream_items(
  conn,
  stream,
  verbose = FALSE,
  count = 10,
  start = NULL,
  local_ordering = FALSE
)
```

Arguments

conn	A connection object created by mc_connect .
stream	Character string. Stream name, reference, or txid.
verbose	Logical. If TRUE, returns detailed item information.
count	Integer. Number of items to return (default 10).
start	Optional integer. Offset (positive for forward, negative for backward). If omitted, the most recent items are returned.
local_ordering	Logical. If TRUE, uses local node's transaction ordering (default FALSE).

Value

A data frame (via `rpc_res_to_df`) with item details.

See Also

[mc_list_stream_key_items](#), [mc_list_stream_publisher_items](#)

Other streams: [mc_create_stream\(\)](#), [mc_create_stream_from\(\)](#), [mc_get_stream_info\(\)](#), [mc_get_stream_item\(\)](#), [mc_get_stream_key_summary\(\)](#), [mc_get_stream_publisher_summary\(\)](#), [mc_list_stream_block_items\(\)](#), [mc_list_stream_key_items\(\)](#), [mc_list_stream_keys\(\)](#), [mc_list_stream_publisher_items\(\)](#), [mc_list_stream_publishers\(\)](#), [mc_list_stream_query_items\(\)](#), [mc_list_stream_tx_items\(\)](#), [mc_list_streams\(\)](#), [mc_publish\(\)](#), [mc_publish_from\(\)](#), [mc_publish_multi\(\)](#), [mc_publish_multi_from\(\)](#)

Examples

```
## Not run:
# Get the 10 most recent items
items <- mc_list_stream_items(conn, "mystream")

# Get next 5 items after the 10th
items <- mc_list_stream_items(conn, "mystream", count = 5, start = 10)

## End(Not run)
```

mc_list_stream_key_items

List items with a specific key

Description

Returns all items in a stream that have a given key.

Usage

```
mc_list_stream_key_items(
  conn,
  stream,
  key,
  verbose = FALSE,
  count = 10,
  start = NULL,
  local_ordering = FALSE
)
```

Arguments

conn	A connection object created by mc_connect .
stream	Character string. Stream name, reference, or txid.
key	Character string. The key to filter by.
verbose	Logical. If TRUE, returns detailed information.
count	Integer. Number of items to return (default 10).
start	Optional integer. Offset for pagination.
local_ordering	Logical. Use local ordering.

Value

A data frame (via `rpc_res_to_df`) with items.

See Also

Other streams: `mc_create_stream()`, `mc_create_stream_from()`, `mc_get_stream_info()`, `mc_get_stream_item()`, `mc_get_stream_key_summary()`, `mc_get_stream_publisher_summary()`, `mc_list_stream_block_items()`, `mc_list_stream_items()`, `mc_list_stream_keys()`, `mc_list_stream_publisher_items()`, `mc_list_stream_publisher_summary()`, `mc_list_stream_query_items()`, `mc_list_stream_tx_items()`, `mc_list_streams()`, `mc_publish()`, `mc_publish_from()`, `mc_publish_multi()`, `mc_publish_multi_from()`

Examples

```
## Not run:
items <- mc_list_stream_key_items(conn, "mystream", "mykey")

## End(Not run)
```

mc_list_stream_keys	<i>List unique keys in a stream</i>
---------------------	-------------------------------------

Description

Returns the set of distinct keys used in a stream, with optional pagination.

Usage

```
mc_list_stream_keys(
  conn,
  stream,
  keys = "*",
  verbose = FALSE,
  count = NULL,
  start = NULL,
  local_ordering = FALSE
)
```

Arguments

conn	A connection object created by <code>mc_connect</code> .
stream	Character string. Stream name, reference, or txid.
keys	Character vector of keys to filter (default "*" for all).
verbose	Logical. If TRUE, returns additional metadata per key.
count	Optional integer. Number of keys to return.
start	Optional integer. Offset.
local_ordering	Logical. Use local ordering.

Value

A data frame (via `rpc_res_to_df`) with key information.

See Also

Other streams: `mc_create_stream()`, `mc_create_stream_from()`, `mc_get_stream_info()`, `mc_get_stream_item()`, `mc_get_stream_key_summary()`, `mc_get_stream_publisher_summary()`, `mc_list_stream_block_items()`, `mc_list_stream_items()`, `mc_list_stream_key_items()`, `mc_list_stream_publisher_items()`, `mc_list_stream_publishers()`, `mc_list_stream_query_items()`, `mc_list_stream_tx_items()`, `mc_list_streams()`, `mc_publish()`, `mc_publish_from()`, `mc_publish_multi()`, `mc_publish_multi_from()`

Examples

```
## Not run:
keys <- mc_list_stream_keys(conn, "mystream")

## End(Not run)
```

`mc_list_stream_publisher_items`

List items published by a specific address

Description

Returns all items in a stream that were published by a given address.

Usage

```
mc_list_stream_publisher_items(
  conn,
  stream,
  address,
  verbose = FALSE,
  count = 10,
  start = NULL,
  local_ordering = FALSE
)
```

Arguments

<code>conn</code>	A connection object created by <code>mc_connect</code> .
<code>stream</code>	Character string. Stream name, reference, or txid.
<code>address</code>	Character string. Publisher address.
<code>verbose</code>	Logical. If TRUE, returns detailed information.
<code>count</code>	Integer. Number of items to return (default 10).
<code>start</code>	Optional integer. Offset.
<code>local_ordering</code>	Logical. Use local ordering.

Value

A data frame (via `rpc_res_to_df`) with items.

See Also

Other streams: `mc_create_stream()`, `mc_create_stream_from()`, `mc_get_stream_info()`, `mc_get_stream_item()`, `mc_get_stream_key_summary()`, `mc_get_stream_publisher_summary()`, `mc_list_stream_block_items()`, `mc_list_stream_items()`, `mc_list_stream_key_items()`, `mc_list_stream_keys()`, `mc_list_stream_publishers()`, `mc_list_stream_query_items()`, `mc_list_stream_tx_items()`, `mc_list_streams()`, `mc_publish()`, `mc_publish_from()`, `mc_publish_multi()`, `mc_publish_multi_from()`

Examples

```
## Not run:
items <- mc_list_stream_publisher_items(conn, "mystream", "1A...")

## End(Not run)
```

`mc_list_stream_publishers`

List publishers who have written to a stream

Description

Returns the set of addresses that have published to a stream.

Usage

```
mc_list_stream_publishers(
  conn,
  stream,
  addresses = "*",
  verbose = FALSE,
  count = NULL,
  start = NULL,
  local_ordering = FALSE
)
```

Arguments

<code>conn</code>	A connection object created by <code>mc_connect</code> .
<code>stream</code>	Character string. Stream name, reference, or txid.
<code>addresses</code>	Character vector of addresses to filter (default <code>"*"</code>).
<code>verbose</code>	Logical. If <code>TRUE</code> , returns additional metadata.
<code>count</code>	Optional integer. Number of publishers to return.
<code>start</code>	Optional integer. Offset.
<code>local_ordering</code>	Logical. Use local ordering.

Value

A data frame (via `rpc_res_to_df`) with publisher information.

See Also

Other streams: `mc_create_stream()`, `mc_create_stream_from()`, `mc_get_stream_info()`, `mc_get_stream_item()`, `mc_get_stream_key_summary()`, `mc_get_stream_publisher_summary()`, `mc_list_stream_block_items()`, `mc_list_stream_items()`, `mc_list_stream_key_items()`, `mc_list_stream_keys()`, `mc_list_stream_publisher_items()`, `mc_list_stream_query_items()`, `mc_list_stream_tx_items()`, `mc_list_streams()`, `mc_publish()`, `mc_publish_from()`, `mc_publish_multi()`, `mc_publish_multi_from()`

Examples

```
## Not run:
publishers <- mc_list_stream_publishers(conn, "mystream")

## End(Not run)
```

mc_list_stream_query_items
<i>Query items by matching keys and publishers</i>

Description

Returns items that match a combination of key and publisher filters.

Usage

```
mc_list_stream_query_items(conn, stream, query, verbose = FALSE)
```

Arguments

conn	A connection object created by <code>mc_connect</code> .
stream	Character string. Stream name, reference, or txid.
query	A list with optional fields: • <code>keys</code> – vector of keys. • <code>publishers</code> – vector of publisher addresses.
verbose	Logical. If TRUE, returns detailed information.

Value

A data frame (via `rpc_res_to_df`) with matching items.

See Also

Other streams: [mc_create_stream\(\)](#), [mc_create_stream_from\(\)](#), [mc_get_stream_info\(\)](#), [mc_get_stream_item\(\)](#), [mc_get_stream_key_summary\(\)](#), [mc_get_stream_publisher_summary\(\)](#), [mc_list_stream_block_items\(\)](#), [mc_list_stream_items\(\)](#), [mc_list_stream_key_items\(\)](#), [mc_list_stream_keys\(\)](#), [mc_list_stream_publisher_items\(\)](#), [mc_list_stream_publishers\(\)](#), [mc_list_stream_tx_items\(\)](#), [mc_list_streams\(\)](#), [mc_publish\(\)](#), [mc_publish_from\(\)](#), [mc_publish_multi\(\)](#), [mc_publish_multi_from\(\)](#)

Examples

```
## Not run:
query <- list(keys = c("key1", "key2"), publishers = c("1A..."))
items <- mc_list_stream_query_items(conn, "mystream", query)

## End(Not run)
```

mc_list_stream_tx_items

List items in stream within a given transaction ID

Description

Returns all stream items that are part of a specific transaction (e.g., when multiple items were published in a single transaction).

Usage

```
mc_list_stream_tx_items(conn, stream, txid, verbose = FALSE)
```

Arguments

conn	A connection object created by mc_connect .
stream	Character string. Stream name, reference, or txid.
txid	Character string. Transaction ID.
verbose	Logical. If TRUE, returns detailed information.

Value

A data frame (via [rpc_res_to_df](#)) with items.

See Also

Other streams: [mc_create_stream\(\)](#), [mc_create_stream_from\(\)](#), [mc_get_stream_info\(\)](#), [mc_get_stream_item\(\)](#), [mc_get_stream_key_summary\(\)](#), [mc_get_stream_publisher_summary\(\)](#), [mc_list_stream_block_items\(\)](#), [mc_list_stream_items\(\)](#), [mc_list_stream_key_items\(\)](#), [mc_list_stream_keys\(\)](#), [mc_list_stream_publisher_items\(\)](#), [mc_list_stream_publishers\(\)](#), [mc_list_stream_query_items\(\)](#), [mc_list_streams\(\)](#), [mc_publish\(\)](#), [mc_publish_from\(\)](#), [mc_publish_multi\(\)](#), [mc_publish_multi_from\(\)](#)

Examples

```
## Not run:
items <- mc_list_stream_tx_items(conn, "mystream", "txid...")

## End(Not run)
```

mc_list_streams	<i>List streams on the blockchain</i>
-----------------	---------------------------------------

Description

Returns a list of streams (or filtered subset) with optional details.

Usage

```
mc_list_streams(
  conn,
  streams = "*",
  verbose = FALSE,
  count = NULL,
  start = NULL
)
```

Arguments

conn	A connection object created by mc_connect .
streams	Character vector of stream names/IDs, or "*" (default) for all streams.
verbose	Logical. If TRUE, returns detailed information.
count	Optional integer. Number of streams to return.
start	Optional integer. Offset for pagination.

Value

A data frame (via `rpc_res_to_df`) with stream information.

See Also

[mc_get_stream_info](#)

Other streams: [mc_create_stream\(\)](#), [mc_create_stream_from\(\)](#), [mc_get_stream_info\(\)](#), [mc_get_stream_item\(\)](#), [mc_get_stream_key_summary\(\)](#), [mc_get_stream_publisher_summary\(\)](#), [mc_list_stream_block_items\(\)](#), [mc_list_stream_items\(\)](#), [mc_list_stream_key_items\(\)](#), [mc_list_stream_keys\(\)](#), [mc_list_stream_publisher_info\(\)](#), [mc_list_stream_publishers\(\)](#), [mc_list_stream_query_items\(\)](#), [mc_list_stream_tx_items\(\)](#), [mc_publish\(\)](#), [mc_publish_from\(\)](#), [mc_publish_multi\(\)](#), [mc_publish_multi_from\(\)](#)

Examples

```
## Not run:
all_streams <- mc_list_streams(conn)
first_10 <- mc_list_streams(conn, count = 10)

## End(Not run)
```

mc_list_tx_filters	<i>List transaction filters</i>
--------------------	---------------------------------

Description

Returns a list of transaction filters on the blockchain.

Usage

```
mc_list_tx_filters(conn, filters = "*", verbose = FALSE)
```

Arguments

conn	A connection object created by mc_connect .
filters	Character vector of filter names/IDs, or "*" (default) for all filters.
verbose	Logical. If TRUE, returns detailed information.

Value

A data frame (via `rpc_res_to_df`) with filter information.

See Also

Other filters: [mc_approve_from\(\)](#), [mc_create_stream_filter\(\)](#), [mc_create_tx_filter\(\)](#), [mc_create_upgrade\(\)](#), [mc_get_filter_code\(\)](#), [mc_list_stream_filters\(\)](#), [mc_list_upgrades\(\)](#), [mc_run_stream_filter\(\)](#), [mc_run_tx_filter\(\)](#), [mc_test_stream_filter\(\)](#), [mc_test_tx_filter\(\)](#)

Examples

```
## Not run:
tx_filters <- mc_list_tx_filters(conn)

## End(Not run)
```

mc_list_unspent	<i>List unspent transaction outputs</i>
-----------------	---

Description

Returns a list of all unspent outputs (UTXOs) available in the wallet.

Usage

```
mc_list_unspent(conn, minconf = 1, maxconf = 999999, addresses = NULL)
```

Arguments

- conn A connection object to the MultiChain node.
- minconf Integer. Minimum confirmations (default 1).
- maxconf Integer. Maximum confirmations (default 999999).
- addresses Optional character vector of addresses to filter the results.

Value

A data frame containing UTXO details, including txid, vout, address, amount, and associated asset/permission data.

See Also

Other wallet: [mc_backup_wallet\(\)](#), [mc_change_wallet_passphrase\(\)](#), [mc_combine_unspent\(\)](#), [mc_dump_privkey\(\)](#), [mc_dump_wallet\(\)](#), [mc_encrypt_wallet\(\)](#), [mc_get_wallet_info\(\)](#), [mc_import_privkey\(\)](#), [mc_import_wallet\(\)](#), [mc_list_lock_unspent\(\)](#), [mc_lock_unspent\(\)](#), [mc_lock_wallet\(\)](#), [mc_unlock_wallet\(\)](#)

mc_list_upgrades	<i>List upgrades</i>
------------------	----------------------

Description

Returns a list of upgrade proposals on the blockchain.

Usage

```
mc_list_upgrades(conn, upgrades = "*")
```

Arguments

- conn A connection object created by [mc_connect](#).
- upgrades Character vector of upgrade names/IDs, or "*" (default) for all upgrades.

Value

A data frame (via `rpc_res_to_df`) with upgrade information.

See Also

Other filters: `mc_approve_from()`, `mc_create_stream_filter()`, `mc_create_tx_filter()`, `mc_create_upgrade()`, `mc_get_filter_code()`, `mc_list_stream_filters()`, `mc_list_tx_filters()`, `mc_run_stream_filter()`, `mc_run_tx_filter()`, `mc_test_stream_filter()`, `mc_test_tx_filter()`

Examples

```
## Not run:
upgrades <- mc_list_upgrades(conn)

## End(Not run)
```

mc_list_variables	<i>List variables created on the blockchain</i>
-------------------	---

Description

Returns a list of variables, with optional filtering, verbosity, and pagination.

Usage

```
mc_list_variables(
  conn,
  variables = "*",
  verbose = FALSE,
  count = NULL,
  start = NULL
)
```

Arguments

conn	A connection object created by <code>mc_connect</code> .
variables	Character vector of variable names/IDs, or "*" (default) for all variables.
verbose	Logical. If TRUE, returns detailed information.
count	Optional integer. Maximum number of variables to return.
start	Optional integer. Offset for pagination.

Value

A data frame (via `rpc_res_to_df`) with variable information.

See Also

Other variables: `mc_create_variable()`, `mc_create_variable_from()`, `mc_get_variable_history()`, `mc_get_variable_info()`, `mc_get_variable_value()`, `mc_set_variable_value()`, `mc_set_variable_value_from()`

Examples

```
## Not run:
all_vars <- mc_list_variables(conn)
first_10 <- mc_list_variables(conn, count = 10)

## End(Not run)
```

```
mc_list_wallet_transactions
```

List transactions in the wallet

Description

Returns a list of the most recent transactions in the wallet.

Usage

```
mc_list_wallet_transactions(
  conn,
  count = 10,
  skip = 0,
  include_watch_only = FALSE,
  verbose = FALSE
)
```

Arguments

<code>conn</code>	A connection object to the MultiChain node.
<code>count</code>	Integer. The number of transactions to return (default 10).
<code>skip</code>	Integer. The number of transactions to skip (default 0).
<code>include_watch_only</code>	Logical. Include watch-only addresses (default FALSE).
<code>verbose</code>	Logical. If TRUE, provides details of inputs and outputs (default FALSE).

Value

A data frame of transaction history for the wallet.

See Also

Other transactions: [mc_get_address_balances\(\)](#), [mc_get_address_transaction\(\)](#), [mc_get_multi_balances\(\)](#), [mc_get_token_balances\(\)](#), [mc_get_total_balances\(\)](#), [mc_get_tx_out_data\(\)](#), [mc_get_wallet_transaction\(\)](#), [mc_list_address_transactions\(\)](#), [mc_send\(\)](#), [mc_send_asset\(\)](#), [mc_send_asset_from\(\)](#), [mc_send_from\(\)](#), [mc_send_with_data\(\)](#), [mc_send_with_data_from\(\)](#)

mc_lock_unspent	<i>Lock or unlock unspent outputs</i>
-----------------	---------------------------------------

Description

Prevents specific UTXOs from being spent in automated transactions, or releases them if they were previously locked.

Usage

`mc_lock_unspent(conn, unlock, outputs = NULL)`

Arguments

- | | |
|---------|--|
| conn | A connection object to the MultiChain node. |
| unlock | Logical. If TRUE, unlocks the outputs; if FALSE, locks them. |
| outputs | Optional list of outputs to (un)lock. Expected format: <code>list(list(txid="id", vout=0), ...)</code> . If empty and unlock=TRUE, all outputs are unlocked. |

Value

Logical TRUE on success.

See Also

Other wallet: [mc_backup_wallet\(\)](#), [mc_change_wallet_passphrase\(\)](#), [mc_combine_unspent\(\)](#), [mc_dump_privkey\(\)](#), [mc_dump_wallet\(\)](#), [mc_encrypt_wallet\(\)](#), [mc_get_wallet_info\(\)](#), [mc_import_privkey\(\)](#), [mc_import_wallet\(\)](#), [mc_list_lock_unspent\(\)](#), [mc_list_unspent\(\)](#), [mc_lock_wallet\(\)](#), [mc_unlock_wallet\(\)](#)

mc_lock_wallet	<i>Immediately lock the wallet</i>
----------------	------------------------------------

Description

Removes the wallet encryption key from memory, requiring a passphrase for further private key operations.

Usage

```
mc_lock_wallet(conn)
```

Arguments

conn	A connection object to the MultiChain node.
------	---

Value

Invisibly returns the RPC result (typically NULL) on success.

See Also

Other wallet: [mc_backup_wallet\(\)](#), [mc_change_wallet_passphrase\(\)](#), [mc_combine_unspent\(\)](#), [mc_dump_privkey\(\)](#), [mc_dump_wallet\(\)](#), [mc_encrypt_wallet\(\)](#), [mc_get_wallet_info\(\)](#), [mc_import_privkey\(\)](#), [mc_import_wallet\(\)](#), [mc_list_lock_unspent\(\)](#), [mc_list_unspent\(\)](#), [mc_lock_unspent\(\)](#), [mc_unlock_wallet\(\)](#)

mc_node_init	<i>Initialize a new MultiChain blockchain</i>
--------------	---

Description

Creates a new blockchain using the multichain-util command. The new blockchain is set up in the MultiChain data directory (platform-specific).

Usage

```
mc_node_init(chain_name)
```

Arguments

chain_name	Character string. Name of the blockchain to create.
------------	---

Value

Invisibly returns the output of the multichain-util create command (a character vector). If the creation fails, the function stops with an error.

See Also

[mc_node_start](#) to start the created node, [mc_node_stop](#) to stop it.
Other node operations: [mc_node_start\(\)](#), [mc_node_stop\(\)](#)

Examples

```
## Not run:  
# Create a blockchain called "my_chain"  
mc_node_init("my_chain")  
  
## End(Not run)
```

mc_node_start	<i>Start a MultiChain node</i>
---------------	--------------------------------

Description

Launches a MultiChain node for a given blockchain. The node is started in daemon mode (-daemon). If a custom data directory is provided, it is passed via the -datadir argument.

Usage

```
mc_node_start(chain_name, datadir = NULL)
```

Arguments

- chain_name Character string. Name of the blockchain to start.
- datadir Optional character string. Custom data directory for the blockchain. If NULL (default), the default MultiChain data location is used.

Value

Invisibly returns TRUE after issuing the start command.

See Also

[mc_node_init](#) to create the blockchain, [mc_node_stop](#) to stop the node.
Other node operations: [mc_node_init\(\)](#), [mc_node_stop\(\)](#)

Examples

```
## Not run:
# Start the node for "my_chain"
mc_node_start("my_chain")

# Start with a custom data directory
mc_node_start("my_chain", datadir = "/path/to/data")

## End(Not run)
```

mc_node_stop	<i>Stop a MultiChain node</i>
--------------	-------------------------------

Description

Stops a running MultiChain node. The function accepts either a connection object (created by `mc_connect()`) or a chain name. When a chain name is provided, it first retrieves the configuration and establishes a connection automatically.

Usage

```
mc_node_stop(x)
```

Arguments

- `x` Either:
- A character string (chain name) — automatically retrieves the configuration and establishes a connection before stopping.
 - An object of class "multichain_conn" — sends the stop command directly to that connected node.

Value

Invisibly returns the result of the RPC stop command.

See Also

[mc_connect](#), [mc_node_start](#) to start a node.

Other node operations: [mc_node_init\(\)](#), [mc_node_start\(\)](#)

Examples

```
## Not run:
# Stop by chain name
mc_node_stop("my_chain")

# Stop using a connection object
conn <- mc_connect(mc_get_config("my_chain"))
mc_node_stop(conn)

## End(Not run)
```

mc_pause	<i>Pause specified node tasks</i>
----------	-----------------------------------

Description

Temporarily suspends certain node operations without shutting down the node. Tasks that can be paused include mining, incoming connections, and off-chain data handling.

Usage

```
mc_pause(conn, tasks)
```

Arguments

conn	A connection object created by mc_connect .
tasks	A character vector or a comma-separated string of tasks to pause. Valid task names are: • "mining" – stop mining new blocks. • "incoming" – stop accepting incoming connections. • "offchain" – stop processing off-chain data. Multiple tasks can be specified, e.g., <code>c("mining", "incoming")</code> .

Value

Invisibly returns the RPC result (typically NULL) on success.

See Also

[mc_resume](#) to restart paused tasks, [mc_clear_mempool](#) for use after pausing.

Other node control: [mc_resume\(\)](#), [mc_set_last_block\(\)](#)

Examples

```
## Not run:
# Pause mining only
mc_pause(conn, "mining")

# Pause both incoming connections and mining
mc_pause(conn, c("incoming", "mining"))

## End(Not run)
```

mc_ping

*Ping all connected peers***Description**

Sends a ping message to all connected peers. The ping time (latency) is recorded and can be viewed in the pingtime column of [mc_get_peer_info](#).

Usage

```
mc_ping(conn)
```

Arguments

conn A connection object created by [mc_connect](#).

Value

Invisibly returns the RPC result (typically NULL) on success.

See Also

[mc_get_peer_info](#) to view ping times.

Other networking: [mc_add_node\(\)](#), [mc_get_added_node_info\(\)](#), [mc_get_network_info\(\)](#), [mc_get_peer_info\(\)](#), [mc_list_stored_nodes\(\)](#), [mc_store_node\(\)](#)

Examples

```
## Not run:
mc_ping(conn)
Sys.sleep(1)
peers <- mc_get_peer_info(conn)
print(peers$pingtime)

## End(Not run)
```

mc_prepare_lock_unspent

Prepare an unspent transaction output for exchange

Description

Locks one or more unspent outputs (assets or native currency) to be used as part of an atomic exchange. The locked output is prepared in a way that it can only be spent as part of an atomic exchange transaction.

Usage

```
mc_prepare_lock_unspent(conn, amounts, lock = TRUE)
```

Arguments

conn	A connection object created by mc_connect .
amounts	A list specifying the assets or native currency to lock. Format: <code>list(asset_name = quantity, ...)</code> or a numeric value for native currency.
lock	Logical. If TRUE (default), the output is locked for automatic spending (i.e., it will be used only in the exchange). If FALSE, the output is simply prepared but not locked.

Value

A list with two elements:

txid	Transaction ID of the prepared/locked output.
vout	Output index.

See Also

[mc_prepare_lock_unspent_from](#), [mc_create_raw_exchange](#)

Other atomic exchange: [mc_append_raw_exchange\(\)](#), [mc_complete_raw_exchange\(\)](#), [mc_create_raw_exchange\(\)](#), [mc_decode_raw_exchange\(\)](#), [mc_disable_raw_transaction\(\)](#), [mc_prepare_lock_unspent_from\(\)](#)

Examples

```
## Not run:
# Lock 10 units of 'myasset' and 0.5 native currency
locked <- mc_prepare_lock_unspent(conn, amounts = list(myasset = 10, 0.5))
# Now use locked$txid and locked$vout in an exchange

## End(Not run)
```

mc_prepare_lock_unspent_from

Prepare an unspent output from a specific address

Description

Similar to [mc_prepare_lock_unspent](#), but allows specifying the source address from which to lock the outputs. This is useful when the node has multiple addresses and you want to control which address's funds are used.

Usage

```
mc_prepare_lock_unspent_from(conn, from_address, amounts, lock = TRUE)
```

Arguments

conn	A connection object created by mc_connect .
from_address	Character string. The address that will provide the assets/native currency.
amounts	A list specifying the assets or native currency to lock.
lock	Logical. If TRUE (default), the output is locked for automatic spending.

Value

A list with txid and vout.

See Also

[mc_prepare_lock_unspent](#)

Other atomic exchange: [mc_append_raw_exchange\(\)](#), [mc_complete_raw_exchange\(\)](#), [mc_create_raw_exchange\(\)](#), [mc_decode_raw_exchange\(\)](#), [mc_disable_raw_transaction\(\)](#), [mc_prepare_lock_unspent\(\)](#)

Examples

```
## Not run:
locked <- mc_prepare_lock_unspent_from(conn, "1A...",
                                     amounts = list(myasset = 5))

## End(Not run)
```

mc_publish	<i>Publish an item to a stream</i>
------------	------------------------------------

Description

Writes a key-value item to a stream. The item is stored on the blockchain (or optionally off-chain) and can be retrieved by its key.

Usage

```
mc_publish(conn, stream, keys, data, options = NULL)
```

Arguments

conn	A connection object created by mc_connect .
stream	Character string. Stream name, reference, or creation txid.
keys	A single key (string) or a vector of keys (for multi-key items).
data	Data to publish. Can be a hex string, or a list with text (will be hex-encoded) or json (will be converted to JSON then hex).
options	Optional character string. Use "offchain" to publish as an off-chain item (requires off-chain capability on the blockchain).

Value

A character string containing the transaction ID of the published item.

See Also

[mc_publish_from](#) to specify publisher address, [mc_publish_multi](#) for multiple items.

Other streams: [mc_create_stream\(\)](#), [mc_create_stream_from\(\)](#), [mc_get_stream_info\(\)](#), [mc_get_stream_item\(\)](#), [mc_get_stream_key_summary\(\)](#), [mc_get_stream_publisher_summary\(\)](#), [mc_list_stream_block_items\(\)](#), [mc_list_stream_items\(\)](#), [mc_list_stream_key_items\(\)](#), [mc_list_stream_keys\(\)](#), [mc_list_stream_publisher_items\(\)](#), [mc_list_stream_publishers\(\)](#), [mc_list_stream_query_items\(\)](#), [mc_list_stream_tx_items\(\)](#), [mc_list_streams\(\)](#), [mc_publish_from\(\)](#), [mc_publish_multi\(\)](#), [mc_publish_multi_from\(\)](#)

Examples

```
## Not run:
# Publish with a text key and simple text data
mc_publish(conn, "mystream", "greeting", list(text = "Hello world!"))

# Publish with JSON data
mc_publish(conn, "mystream", "data", list(json = list(a = 1, b = 2)))

# Publish off-chain
mc_publish(conn, "mystream", "large", list(text = "big data"), options = "offchain")
```

```
## End(Not run)
```

mc_publish_from

Publish an item to a stream from a specific address

Description

Similar to [mc_publish](#), but specifies the publishing address.

Usage

```
mc_publish_from(conn, from_address, stream, keys, data, options = NULL)
```

Arguments

conn	A connection object created by mc_connect .
from_address	Character string. Address that will publish the item.
stream	Character string. Stream name, reference, or txid.
keys	A single key or vector of keys.
data	Data to publish (string or list).
options	Optional "offchain" string.

Value

A character string containing the transaction ID.

See Also

[mc_publish](#)

Other streams: [mc_create_stream\(\)](#), [mc_create_stream_from\(\)](#), [mc_get_stream_info\(\)](#), [mc_get_stream_item\(\)](#), [mc_get_stream_key_summary\(\)](#), [mc_get_stream_publisher_summary\(\)](#), [mc_list_stream_block_items\(\)](#), [mc_list_stream_items\(\)](#), [mc_list_stream_key_items\(\)](#), [mc_list_stream_keys\(\)](#), [mc_list_stream_publisher_items\(\)](#), [mc_list_stream_publishers\(\)](#), [mc_list_stream_query_items\(\)](#), [mc_list_stream_tx_items\(\)](#), [mc_list_streams\(\)](#), [mc_publish\(\)](#), [mc_publish_multi\(\)](#), [mc_publish_multi_from\(\)](#)

Examples

```
## Not run:
mc_publish_from(conn, "1A...", "mystream", "key", list(text = "data"))

## End(Not run)
```

mc_publish_multi	<i>Publish multiple items to a stream in one transaction</i>
------------------	--

Description

Publishes several key-value items in a single transaction. This is more efficient than publishing each item separately.

Usage

```
mc_publish_multi(conn, stream, items, options = NULL)
```

Arguments

conn	A connection object created by mc_connect .
stream	Character string. Default stream for all items (if not overridden per item).
items	A list of item objects, each containing: • key or keys – the key(s) for the item. • data – the data to publish. • (optional) stream – override the default stream.
options	Optional default "offchain" string.

Value

A character string containing the transaction ID.

See Also

[mc_publish_multi_from](#) for specifying the sender.

Other streams: [mc_create_stream\(\)](#), [mc_create_stream_from\(\)](#), [mc_get_stream_info\(\)](#), [mc_get_stream_item\(\)](#), [mc_get_stream_key_summary\(\)](#), [mc_get_stream_publisher_summary\(\)](#), [mc_list_stream_block_items\(\)](#), [mc_list_stream_items\(\)](#), [mc_list_stream_key_items\(\)](#), [mc_list_stream_keys\(\)](#), [mc_list_stream_publisher_items\(\)](#), [mc_list_stream_publishers\(\)](#), [mc_list_stream_query_items\(\)](#), [mc_list_stream_tx_items\(\)](#), [mc_list_streams\(\)](#), [mc_publish\(\)](#), [mc_publish_from\(\)](#), [mc_publish_multi_from\(\)](#)

Examples

```
## Not run:
items <- list(
  list(key = "k1", data = list(text = "item1")),
  list(key = "k2", data = list(json = list(value = 2)))
)
txid <- mc_publish_multi(conn, "mystream", items)

## End(Not run)
```

mc_publish_multi_from *Publish multiple items from a specific address*

Description

Publishes multiple items in one transaction from a specified address.

Usage

```
mc_publish_multi_from(conn, from_address, stream, items, options = NULL)
```

Arguments

conn	A connection object created by mc_connect .
from_address	Character string. Address that will publish the items.
stream	Character string. Default stream.
items	List of item objects.
options	Optional default "offchain" string.

Value

A character string containing the transaction ID.

See Also

[mc_publish_multi](#)

Other streams: [mc_create_stream\(\)](#), [mc_create_stream_from\(\)](#), [mc_get_stream_info\(\)](#), [mc_get_stream_item\(\)](#), [mc_get_stream_key_summary\(\)](#), [mc_get_stream_publisher_summary\(\)](#), [mc_list_stream_block_items\(\)](#), [mc_list_stream_items\(\)](#), [mc_list_stream_key_items\(\)](#), [mc_list_stream_keys\(\)](#), [mc_list_stream_publisher_items\(\)](#), [mc_list_stream_publishers\(\)](#), [mc_list_stream_query_items\(\)](#), [mc_list_stream_tx_items\(\)](#), [mc_list_streams\(\)](#), [mc_publish\(\)](#), [mc_publish_from\(\)](#), [mc_publish_multi\(\)](#)

Examples

```
## Not run:
items <- list(list(key = "k1", data = "value1"))
txid <- mc_publish_multi_from(conn, "1A...", "mystream", items)

## End(Not run)
```

mc_resume	<i>Resume specified node tasks</i>
-----------	------------------------------------

Description

Restarts node tasks that were previously paused with [mc_pause](#).

Usage

```
mc_resume(conn, tasks)
```

Arguments

conn	A connection object created by mc_connect .
tasks	A character vector or a comma-separated string of tasks to resume. Valid task names: "mining", "incoming", "offchain".

Value

Invisibly returns the RPC result (typically NULL) on success.

See Also

[mc_pause](#) to pause tasks.

Other node control: [mc_pause\(\)](#), [mc_set_last_block\(\)](#)

Examples

```
## Not run:  
# Resume mining after a pause  
mc_resume(conn, "mining")  
  
# Resume all paused tasks  
mc_resume(conn, c("mining", "incoming", "offchain"))  
  
## End(Not run)
```

mc_revoke	<i>Revoke permissions from an address</i>
-----------	---

Description

Removes one or more permissions from a wallet address. The revoking address must have admin rights.

Usage

```
mc_revoke(conn, address, permissions)
```

Arguments

conn	A connection object created by mc_connect .
address	Character string. The address from which permissions are revoked.
permissions	Character string. Comma-separated list of permissions to revoke.

Value

A character string containing the transaction ID.

See Also

[mc_revoke_from](#) to specify the revoker, [mc_grant](#) for granting.

Other permissions: [mc_grant\(\)](#), [mc_grant_from\(\)](#), [mc_grant_with_data\(\)](#), [mc_grant_with_data_from\(\)](#), [mc_list_permissions\(\)](#), [mc_revoke_from\(\)](#), [mc_verify_permission\(\)](#)

Examples

```
## Not run:  
# Revoke send and receive permissions  
txid <- mc_revoke(conn, "1A...", "send, receive")  
  
## End(Not run)
```

mc_revoke_from	<i>Revoke permissions from a specific address</i>
----------------	---

Description

Revokes permissions from an address, specifying the address that issues the revocation. This allows controlling which address pays for the transaction.

Usage

```
mc_revoke_from(conn, from_address, to_address, permissions, native_amount = 0)
```

Arguments

conn	A connection object created by mc_connect .
from_address	Character string. The address revoking the permissions.
to_address	Character string. The address losing the permissions.
permissions	Character string. Comma-separated list of permissions to revoke.
native_amount	Numeric. Amount of native currency to send along with the revocation (default 0).

Value

A character string containing the transaction ID.

See Also

[mc_revoke](#), [mc_grant_from](#).

Other permissions: [mc_grant\(\)](#), [mc_grant_from\(\)](#), [mc_grant_with_data\(\)](#), [mc_grant_with_data_from\(\)](#), [mc_list_permissions\(\)](#), [mc_revoke\(\)](#), [mc_verify_permission\(\)](#)

Examples

```
## Not run:
# Revoke from a specific admin address
txid <- mc_revoke_from(conn, "admin1...", "user1...", "send")

## End(Not run)
```

mc_run_stream_filter *Run an existing stream filter against an item*

Description

Executes an existing stream filter on a specific stream item (transaction and optional vout).

Usage

```
mc_run_stream_filter(conn, filter, tx, vout = NULL)
```

Arguments

conn	A connection object created by mc_connect .
filter	Character string. Filter name or transaction ID.
tx	Character string. Transaction ID or hex representation.
vout	Optional integer. Output index if the stream item is in a transaction output.

Value

The output of the filter.

See Also

[mc_test_stream_filter](#), [mc_create_stream_filter](#)

Other filters: [mc_approve_from\(\)](#), [mc_create_stream_filter\(\)](#), [mc_create_tx_filter\(\)](#), [mc_create_upgrade\(\)](#), [mc_get_filter_code\(\)](#), [mc_list_stream_filters\(\)](#), [mc_list_tx_filters\(\)](#), [mc_list_upgrades\(\)](#), [mc_run_tx_filter\(\)](#), [mc_test_stream_filter\(\)](#), [mc_test_tx_filter\(\)](#)

Examples

```
## Not run:
result <- mc_run_stream_filter(conn, "myfilter", "txid...", vout = 0)

## End(Not run)
```

mc_run_tx_filter	<i>Run an existing transaction filter against a transaction</i>
------------------	---

Description

Executes an existing transaction filter on a given transaction, without performing a blockchain operation.

Usage

```
mc_run_tx_filter(conn, filter, tx)
```

Arguments

conn	A connection object created by mc_connect .
filter	Character string. Filter name or transaction ID.
tx	Character string. Transaction ID or hex representation.

Value

The output of the filter (e.g., boolean, transformed transaction).

See Also

[mc_test_tx_filter](#), [mc_create_tx_filter](#)

Other filters: [mc_approve_from\(\)](#), [mc_create_stream_filter\(\)](#), [mc_create_tx_filter\(\)](#), [mc_create_upgrade\(\)](#), [mc_get_filter_code\(\)](#), [mc_list_stream_filters\(\)](#), [mc_list_tx_filters\(\)](#), [mc_list_upgrades\(\)](#), [mc_run_stream_filter\(\)](#), [mc_test_stream_filter\(\)](#), [mc_test_tx_filter\(\)](#)

Examples

```
## Not run:
result <- mc_run_tx_filter(conn, "myfilter", "txid...")

## End(Not run)
```

mc_send	<i>Send payment or assets to an address</i>
---------	---

Description

Sends a payment (native currency or assets) to a specified address. This is a general-purpose sending function that can handle multiple asset types in a single transaction.

Usage

```
mc_send(conn, address, amounts, comment = "", comment_to = "")
```

Arguments

conn	A connection object created by mc_connect .
address	Character string. Recipient address.
amounts	Either a numeric value (for native currency) or a named list specifying assets and quantities, e.g., <code>list(asset1 = 10, asset2 = 5)</code> . For metadata, see mc_send_with_data .
comment	Character string. Optional transaction comment (stored on chain).
comment_to	Character string. Optional comment-to field.

Value

A character string containing the transaction ID (txid).

See Also

[mc_send_from](#) to specify the sender, [mc_send_asset](#) for single-asset convenience.

Other transactions: [mc_get_address_balances\(\)](#), [mc_get_address_transaction\(\)](#), [mc_get_multi_balances\(\)](#), [mc_get_token_balances\(\)](#), [mc_get_total_balances\(\)](#), [mc_get_tx_out_data\(\)](#), [mc_get_wallet_transaction\(\)](#), [mc_list_address_transactions\(\)](#), [mc_list_wallet_transactions\(\)](#), [mc_send_asset\(\)](#), [mc_send_asset_from\(\)](#), [mc_send_from\(\)](#), [mc_send_with_data\(\)](#), [mc_send_with_data_from\(\)](#)

Examples

```
## Not run:
# Send 1.5 native coins
txid <- mc_send(conn, "1A...", 1.5)

# Send assets only
txid <- mc_send(conn, "1A...", list(myasset = 100))

# Send both native and assets
txid <- mc_send(conn, "1A...", list(0.5, myasset = 50))

## End(Not run)
```

mc_send_asset	<i>Send a single asset to an address</i>
---------------	--

Description

Convenience function to send a single asset (or native currency) to an address. Equivalent to [mc_send](#) but with a simpler interface.

Usage

```
mc_send_asset(
    conn,
    address,
    asset,
    quantity,
    native_amount = 0,
    comment = "",
    comment_to = ""
)
```

Arguments

conn	A connection object created by mc_connect .
address	Character string. Recipient address.
asset	Character string. Asset name, reference, or issuance transaction ID.
quantity	Numeric. Amount of the asset to send.
native_amount	Numeric. Amount of native currency to send (default 0).
comment	Character string. Optional transaction comment.
comment_to	Character string. Optional comment-to field.

Value

A character string containing the transaction ID.

See Also

[mc_send_asset_from](#) to specify sender, [mc_send](#) for multiple assets.

Other transactions: [mc_get_address_balances\(\)](#), [mc_get_address_transaction\(\)](#), [mc_get_multi_balances\(\)](#), [mc_get_token_balances\(\)](#), [mc_get_total_balances\(\)](#), [mc_get_tx_out_data\(\)](#), [mc_get_wallet_transaction\(\)](#), [mc_list_address_transactions\(\)](#), [mc_list_wallet_transactions\(\)](#), [mc_send\(\)](#), [mc_send_asset_from\(\)](#), [mc_send_from\(\)](#), [mc_send_with_data\(\)](#), [mc_send_with_data_from\(\)](#)

Examples

```
## Not run:
# Send 100 units of "myasset"
txid <- mc_send_asset(conn, "1A...", "myasset", 100)

# Send asset along with 0.5 native coins
txid <- mc_send_asset(conn, "1A...", "myasset", 100, native_amount = 0.5)

## End(Not run)
```

mc_send_asset_from	<i>Send a single asset from a specific address</i>
--------------------	--

Description

Sends an asset (or native currency) from a specific sender address. Useful when the node has multiple addresses and you want to control which address the funds are taken from.

Usage

```
mc_send_asset_from(
  conn,
  from_address,
  to_address,
  asset,
  quantity,
  native_amount = 0,
  comment = "",
  comment_to = ""
)
```

Arguments

conn	A connection object created by mc_connect .
from_address	Character string. Sender address (must belong to the node's wallet).
to_address	Character string. Recipient address.
asset	Character string. Asset name, reference, or issuance transaction ID.
quantity	Numeric. Amount of the asset to send.
native_amount	Numeric. Amount of native currency to send (default 0).
comment	Character string. Optional transaction comment.
comment_to	Character string. Optional comment-to field.

Value

A character string containing the transaction ID.

See Also

[mc_send_asset](#), [mc_send_from](#).

Other transactions: [mc_get_address_balances\(\)](#), [mc_get_address_transaction\(\)](#), [mc_get_multi_balances\(\)](#), [mc_get_token_balances\(\)](#), [mc_get_total_balances\(\)](#), [mc_get_tx_out_data\(\)](#), [mc_get_wallet_transaction\(\)](#), [mc_list_address_transactions\(\)](#), [mc_list_wallet_transactions\(\)](#), [mc_send\(\)](#), [mc_send_asset\(\)](#), [mc_send_from\(\)](#), [mc_send_with_data\(\)](#), [mc_send_with_data_from\(\)](#)

Examples

```
## Not run:
# Send from a specific address
txid <- mc_send_asset_from(conn, "1A...", "1B...", "myasset", 50)

## End(Not run)
```

mc_send_from	<i>Send payment from a specific address</i>
--------------	---

Description

Sends a payment (native currency or assets) from a specific sender address. This is the counterpart of [mc_send](#) for multi-asset transactions with a chosen source address.

Usage

```
mc_send_from(
  conn,
  from_address,
  to_address,
  amounts,
  comment = "",
  comment_to = ""
)
```

Arguments

- | | |
|--------------|---|
| conn | A connection object created by mc_connect . |
| from_address | Character string. Sender address. |
| to_address | Character string. Recipient address. |
| amounts | Either a numeric value (native) or a named list of assets. |
| comment | Character string. Optional transaction comment. |
| comment_to | Character string. Optional comment-to field. |

Value

A character string containing the transaction ID.

See Also

[mc_send](#), [mc_send_asset_from](#).

Other transactions: [mc_get_address_balances\(\)](#), [mc_get_address_transaction\(\)](#), [mc_get_multi_balances\(\)](#), [mc_get_token_balances\(\)](#), [mc_get_total_balances\(\)](#), [mc_get_tx_out_data\(\)](#), [mc_get_wallet_transaction\(\)](#), [mc_list_address_transactions\(\)](#), [mc_list_wallet_transactions\(\)](#), [mc_send\(\)](#), [mc_send_asset\(\)](#), [mc_send_asset_from\(\)](#), [mc_send_with_data\(\)](#), [mc_send_with_data_from\(\)](#)

Examples

```
## Not run:
# Send 2 native coins from one address to another
txid <- mc_send_from(conn, "1A...", "1B...", 2)

# Send assets from a specific address
txid <- mc_send_from(conn, "1A...", "1B...", list(myasset = 100, other = 50))

## End(Not run)
```

mc_send_raw_transaction

Send a signed raw transaction to the network

Description

Broadcasts a signed raw transaction to the blockchain network. The transaction must be complete (all inputs signed) before sending.

Usage

```
mc_send_raw_transaction(conn, tx_hex)
```

Arguments

conn	A connection object created by mc_connect .
tx_hex	Character string. The signed raw transaction hex to send.

Value

A character string containing the transaction ID (txid).

See Also

[mc_sign_raw_transaction](#), [mc_create_raw_transaction](#)

Other raw transactions: [mc_append_raw_change\(\)](#), [mc_append_raw_data\(\)](#), [mc_append_raw_transaction\(\)](#), [mc_create_raw_send_from\(\)](#), [mc_create_raw_transaction\(\)](#), [mc_decode_raw_transaction\(\)](#), [mc_sign_raw_transaction\(\)](#)

Examples

```
## Not run:
# Create, sign, and send a transaction
tx_hex <- mc_create_raw_transaction(conn, inputs, outputs)
signed <- mc_sign_raw_transaction(conn, tx_hex)
if (signed$complete) {
  txid <- mc_send_raw_transaction(conn, signed$hex)
}

## End(Not run)
```

mc_send_with_data	<i>Send payment with inline metadata</i>
-------------------	--

Description

Sends a transaction that includes arbitrary data (metadata) attached to the output. The data can be text, JSON, or any binary data (hex-encoded). This is useful for storing small amounts of information on the blockchain.

Usage

```
mc_send_with_data(conn, address, amounts, data)
```

Arguments

conn	A connection object created by mc_connect .
address	Character string. Recipient address.
amounts	Either a numeric value (native) or a named list of assets.
data	Data to embed. Can be a character string (will be hex-encoded), a list (will be converted to JSON then hex), or raw binary (not directly). The function automatically converts lists to JSON and then to hex.

Value

A character string containing the transaction ID.

See Also

`mc_send_with_data_from` to specify sender, `mc_send` for simple payments.

Other transactions: `mc_get_address_balances()`, `mc_get_address_transaction()`, `mc_get_multi_balances()`, `mc_get_token_balances()`, `mc_get_total_balances()`, `mc_get_tx_out_data()`, `mc_get_wallet_transaction()`, `mc_list_address_transactions()`, `mc_list_wallet_transactions()`, `mc_send()`, `mc_send_asset()`, `mc_send_asset_from()`, `mc_send_from()`, `mc_send_with_data_from()`

Examples

```
## Not run:
# Send with a text note
txid <- mc_send_with_data(conn, "1A...", 0.1, "Hello, blockchain!")

# Send with structured JSON metadata
metadata <- list(id = 123, action = "transfer", tag = "payment")
txid <- mc_send_with_data(conn, "1A...", list(myasset = 10), metadata)

## End(Not run)
```

`mc_send_with_data_from`

Send payment from specific address with metadata

Description

Sends a transaction with inline metadata from a specified sender address. Combines the capabilities of `mc_send_from` and `mc_send_with_data`.

Usage

```
mc_send_with_data_from(conn, from_address, to_address, amounts, data)
```

Arguments

<code>conn</code>	A connection object created by <code>mc_connect</code> .
<code>from_address</code>	Character string. Sender address.
<code>to_address</code>	Character string. Recipient address.
<code>amounts</code>	Either a numeric value (native) or a named list of assets.
<code>data</code>	Data to embed. Can be a string, list (converted to JSON), etc.

Value

A character string containing the transaction ID.

See Also

[mc_send_with_data](#), [mc_send_from](#).

Other transactions: [mc_get_address_balances\(\)](#), [mc_get_address_transaction\(\)](#), [mc_get_multi_balances\(\)](#), [mc_get_token_balances\(\)](#), [mc_get_total_balances\(\)](#), [mc_get_tx_out_data\(\)](#), [mc_get_wallet_transaction\(\)](#), [mc_list_address_transactions\(\)](#), [mc_list_wallet_transactions\(\)](#), [mc_send\(\)](#), [mc_send_asset\(\)](#), [mc_send_asset_from\(\)](#), [mc_send_from\(\)](#), [mc_send_with_data\(\)](#)

Examples

```
## Not run:
# Send from a specific address with metadata
txid <- mc_send_with_data_from(conn, "1A...", "1B...", 0.5,
                              list(reference = "invoice123"))

## End(Not run)
```

mc_set_last_block	<i>Rewind the node's active chain</i>
-------------------	---------------------------------------

Description

Moves the node's active chain to a previous block, effectively rolling back the blockchain state. This is a powerful operation typically used for testing or recovery. The node must be paused with `mc_pause(conn, "incoming,mining")` before calling.

Usage

```
mc_set_last_block(conn, hash_or_height)
```

Arguments

`conn` A connection object created by [mc_connect](#).

`hash_or_height` Either a block hash (character string) or a block height (integer) to rewind to.

Value

Character string. The hash of the last block after the rewind.

See Also

[mc_pause](#), [mc_resume](#), [mc_get_block](#) to inspect blocks.

Other node control: [mc_pause\(\)](#), [mc_resume\(\)](#)

Examples

```
## Not run:
# Pause the node first
mc_pause(conn, "incoming,mining")
# Rewind to block height 100
last_hash <- mc_set_last_block(conn, 100)
# Resume after rewind
mc_resume(conn, "incoming,mining")

## End(Not run)
```

mc_set_path

*Set path to MultiChain binaries***Description**

This function sets the global option `multichain.path` to the directory containing the MultiChain executables (`multichaind` and `multichain-util`). All other functions that need to locate the binaries will use this option.

Usage

```
mc_set_path(path)
```

Arguments

path	Character string. Path to the folder containing the MultiChain executables. Must be an existing directory.
------	--

Value

Invisibly returns the normalized path (as set in the option) or throws an error if the directory does not exist.

See Also

[mc_connect](#) for establishing a connection.

Examples

```
## Not run:
# Set path to MultiChain installation (example on Unix-like systems)
mc_set_path("/usr/local/bin")

# Check that the option was set correctly
getOption("multichain.path")

## End(Not run)
```

mc_set_runtime_param *Set node runtime parameter*

Description

Changes a runtime parameter of the node without requiring a restart. Only a predefined set of parameters can be modified.

Usage

```
mc_set_runtime_param(conn, name, value)
```

Arguments

conn	A connection object created by mc_connect .
name	Character string. The name of the parameter to change. Must be one of: acceptfiltertimeout Timeout for filter acceptance. autosubscribe Automatically subscribe to streams. bantx Ban transactions from the mempool. handshakelocal Local handshake behaviour. hideknownopdrops Hide known opdrops. lockadminmin rounds Lock admin mining rounds. lockblock Lock block creation. lockinlinemetdata Lock inline metadata. maxshowndata Maximum shown data size. maxqueryscanitems Maximum items to scan in queries. mineemptyrounds Number of empty mining rounds. miningrequirespeers Require peers for mining. miningturnover Mining turnover. sendfiltertimeout Timeout for sending filters.
value	The new value for the parameter. Type depends on the parameter: logical, numeric, or character.

Value

Invisibly returns the RPC result (typically NULL) on success; throws an error if the parameter name is invalid or the value is inappropriate.

See Also

[mc_get_runtime_params](#) to inspect current settings.

Other node configuration: [mc_get_blockchain_params\(\)](#), [mc_get_runtime_params\(\)](#)

Examples

```
## Not run:
# Turn off auto-subscription
mc_set_runtime_param(conn, "autosubscribe", FALSE)

# Set maximum connections to 50
mc_set_runtime_param(conn, "maxconnections", 50)

## End(Not run)
```

mc_set_variable_value *Set the value of a variable*

Description

Updates the value of an existing variable. The new value can be any JSON-compatible object (list, number, string, etc.).

Usage

```
mc_set_variable_value(conn, variable, value = NULL)
```

Arguments

conn	A connection object created by mc_connect .
variable	Character string. Variable name or transaction ID.
value	Optional. New value (any JSON structure). If NULL, the variable is deleted.

Value

A list containing the result of the RPC call (usually transaction ID).

See Also

[mc_create_variable](#), [mc_get_variable_value](#)

Other variables: [mc_create_variable\(\)](#), [mc_create_variable_from\(\)](#), [mc_get_variable_history\(\)](#), [mc_get_variable_info\(\)](#), [mc_get_variable_value\(\)](#), [mc_list_variables\(\)](#), [mc_set_variable_value_from\(\)](#)

Examples

```
## Not run:
mc_set_variable_value(conn, "myvar", value = list(updated = TRUE))

## End(Not run)
```

`mc_set_variable_value_from`*Set variable value from specific address*

Description

Updates a variable's value, specifying the address that pays for the transaction.

Usage

```
mc_set_variable_value_from(conn, from_address, variable, value = NULL)
```

Arguments

<code>conn</code>	A connection object created by mc_connect .
<code>from_address</code>	Character string. Address that pays for the update.
<code>variable</code>	Character string. Variable name or transaction ID.
<code>value</code>	Optional. New value (any JSON structure). If NULL, the variable is deleted.

Value

A list containing the result of the RPC call (usually transaction ID).

See Also

Other variables: [mc_create_variable\(\)](#), [mc_create_variable_from\(\)](#), [mc_get_variable_history\(\)](#), [mc_get_variable_info\(\)](#), [mc_get_variable_value\(\)](#), [mc_list_variables\(\)](#), [mc_set_variable_value\(\)](#)

Examples

```
## Not run:
mc_set_variable_value_from(conn, "1A...", "myvar", value = 100)

## End(Not run)
```

mc_sign_message	<i>Sign a message with a private key</i>
-----------------	--

Description

Generates a base64-encoded digital signature for a message using a private key. The signature proves that the message was approved by the owner of the address or the holder of the private key.

Usage

```
mc_sign_message(conn, address_or_key, message)
```

Arguments

conn	A connection object created by mc_connect .
address_or_key	Character string. Either a wallet address (must belong to the node's wallet) or a private key in Wallet Import Format (WIF).
message	Character string. The text message to sign.

Value

A character string containing the base64-encoded signature.

See Also

[mc_verify_message](#) to verify a signature, [mc_get_new_address](#) to generate a new address.

Other cryptography: [mc_verify_message\(\)](#)

Examples

```
## Not run:  
# Sign using an address in the wallet  
sig <- mc_sign_message(conn, "1A...", "Hello, MultiChain!")  
  
# Sign using a raw private key  
sig <- mc_sign_message(conn, "L5...", "Important agreement")  
  
## End(Not run)
```

mc_sign_raw_transaction

Sign a raw transaction

Description

Signs a raw transaction using the node's wallet or provided private keys. Returns the signed hex and a boolean indicating whether all inputs are signed.

Usage

```
mc_sign_raw_transaction(
    conn,
    tx_hex,
    parents = NULL,
    private_keys = NULL,
    sighashtype = "ALL"
)
```

Arguments

conn	A connection object created by mc_connect .
tx_hex	Character string. The raw transaction hex to sign.
parents	Optional list of parent outputs for signing, each containing txid, vout, and scriptPubKey.
private_keys	Optional character vector of private keys in Wallet Import Format (WIF). If provided, these are used instead of the node's wallet.
sighashtype	Character string. Signature hash type (default "ALL").

Value

A list with two elements:

hex	The signed raw transaction hex (if all inputs are signed).
complete	Logical; TRUE if all inputs are signed.

See Also

[mc_send_raw_transaction](#), [mc_create_raw_transaction](#)

Other raw transactions: [mc_append_raw_change\(\)](#), [mc_append_raw_data\(\)](#), [mc_append_raw_transaction\(\)](#), [mc_create_raw_send_from\(\)](#), [mc_create_raw_transaction\(\)](#), [mc_decode_raw_transaction\(\)](#), [mc_send_raw_transaction\(\)](#)

Examples

```
## Not run:
# Sign using the node's wallet
signed <- mc_sign_raw_transaction(conn, tx_hex)

# Sign with explicit private keys
signed <- mc_sign_raw_transaction(conn, tx_hex,
                                private_keys = c("L5...", "K3..."))

## End(Not run)
```

mc_store_node

Add an IP address to known peer nodes (MultiChain 2.3+)

Description

Stores a node address in the node's address manager. This can be used to manually add a peer for future connections or to ignore a node.

Usage

```
mc_store_node(conn, node, command = c("tryconnect", "ignore"))
```

Arguments

conn	A connection object created by mc_connect .
node	Character string. The IP address and port of the peer node.
command	Character string. Action to perform: • "tryconnect" – store the node and try to connect to it (default). • "ignore" – ignore the node (do not attempt connections).

Value

Invisibly returns the RPC result (typically NULL) on success.

See Also

[mc_list_stored_nodes](#) to list stored nodes.

Other networking: [mc_add_node\(\)](#), [mc_get_added_node_info\(\)](#), [mc_get_network_info\(\)](#), [mc_get_peer_info\(\)](#), [mc_list_stored_nodes\(\)](#), [mc_ping\(\)](#)

Examples

```
## Not run:
# Add a node to the address manager
mc_store_node(conn, "192.168.1.20:8571", command = "tryconnect")

## End(Not run)
```

mc_subscribe

Subscribe to MultiChain assets or streams

Description

Instructs the MultiChain node to start tracking one or more asset(s) or stream(s). This is often required before you can retrieve items or balances for specific entities.

Usage

```
mc_subscribe(conn, entities, rescan = TRUE)
```

Arguments

conn	A connection object to the MultiChain node (typically created via <code>mc_connect</code>).
entities	A character string or vector of strings representing asset/stream names, references, or transaction IDs (txids).
rescan	Logical. If TRUE (default), the node reindexes all items from the point of creation of the entities. If FALSE, only new items will be tracked.

Value

Returns NULL invisibly on success, or an error if the RPC call fails.

See Also

[mc_unsubscribe](#)

Other subscriptions: [mc_unsubscribe\(\)](#)

mc_test_library	<i>Manage testing of libraries and updates locally</i>
-----------------	--

Description

Tests a library's code or a specific update without permanently creating it. The behaviour depends on the arguments:

- If only `js_code` is provided, tests that code as a new library.
- If `library` and optionally `updatename` are given, tests the existing library's code (or a specific update).

Usage

```
mc_test_library(conn, library = NULL, updatename = NULL, js_code = NULL)
```

Arguments

<code>conn</code>	A connection object created by mc_connect .
<code>library</code>	Optional character string. Library name or transaction ID.
<code>updatename</code>	Optional character string. Update name (if testing an update).
<code>js_code</code>	Optional character string. JavaScript code (if testing a new library).

Value

The result of the test (e.g., compiled code, validation output).

See Also

[mc_create_library](#), [mc_add_library_update](#)

Other libraries: [mc_add_library_update\(\)](#), [mc_add_library_update_from\(\)](#), [mc_create_library\(\)](#), [mc_get_library_code\(\)](#), [mc_list_libraries\(\)](#)

Examples

```
## Not run:
# Test a new library
mc_test_library(conn, js_code = "function add(a, b) { return a + b; }")

# Test an existing library's active code
mc_test_library(conn, library = "math")

## End(Not run)
```

mc_test_stream_filter *Test a stream filter before creation*

Description

Tests a stream filter's JavaScript code against a specific stream item (transaction and optional vout) without permanently creating the filter.

Usage

```
mc_test_stream_filter(conn, options, js_code, tx = NULL, vout = NULL)
```

Arguments

conn	A connection object created by mc_connect .
options	List of filter options (similar to mc_create_stream_filter).
js_code	Character string. JavaScript code to test.
tx	Optional character string. Transaction ID or hex representation of the stream item's transaction.
vout	Optional integer. Output index if the stream item is in a transaction output.

Value

The result of the filter evaluation.

See Also

[mc_create_stream_filter](#), [mc_run_stream_filter](#)

Other filters: [mc_approve_from\(\)](#), [mc_create_stream_filter\(\)](#), [mc_create_tx_filter\(\)](#), [mc_create_upgrade\(\)](#), [mc_get_filter_code\(\)](#), [mc_list_stream_filters\(\)](#), [mc_list_tx_filters\(\)](#), [mc_list_upgrades\(\)](#), [mc_run_stream_filter\(\)](#), [mc_run_tx_filter\(\)](#), [mc_test_tx_filter\(\)](#)

Examples

```
## Not run:
result <- mc_test_stream_filter(conn, list(libraries = list()),
                                "function filter(stream, item) { return true; }",
                                tx = "txid...", vout = 0)

## End(Not run)
```

mc_txout_to_binary_cache

Extract transaction output data to binary cache

Description

Copies data directly from a blockchain transaction output into a binary cache item. This is efficient for retrieving binary data stored in a transaction (e.g., via [mc_publish](#)) without having to decode it in R.

Usage

```
mc_txout_to_binary_cache(
  conn,
  identifier,
  txid,
  vout,
  count_bytes = NULL,
  start_byte = 0
)
```

Arguments

conn	A connection object created by mc_connect .
identifier	Character string. Target cache item identifier. The cache item must be empty (created but not yet written to).
txid	Character string. Transaction ID containing the output.
vout	Integer. Output index (vout) of the transaction to extract.
count_bytes	Integer (optional). Number of bytes to extract. If NULL (default), the entire output data is copied.
start_byte	Integer (optional). Byte offset from which to start copying. Default is 0 (beginning of the output data).

Value

Integer. The resulting size of the cache item after extraction.

See Also

[mc_create_binary_cache](#), [mc_append_binary_cache](#)

Other binary cache: [mc_append_binary_cache\(\)](#), [mc_create_binary_cache\(\)](#), [mc_delete_binary_cache\(\)](#)

Examples

```
## Not run:
# Create an empty cache item
id <- mc_create_binary_cache(conn)

# Copy the entire data from a transaction output
size <- mc_txout_to_binary_cache(conn, id, txid = "abc...", vout = 0)

# Copy only the first 100 bytes
size <- mc_txout_to_binary_cache(conn, id, txid = "abc...", vout = 0,
                                count_bytes = 100)

## End(Not run)
```

mc_unlock_wallet	<i>Unlock the wallet with a passphrase</i>
------------------	--

Description

Stores the wallet decryption key in memory for a specified duration.

Usage

```
mc_unlock_wallet(conn, passphrase, timeout)
```

Arguments

conn	A connection object to the MultiChain node.
passphrase	Character. The wallet password.
timeout	Integer. Time in seconds to keep the wallet unlocked.

Value

Invisibly returns the RPC result (typically NULL) on success.

See Also

Other wallet: [mc_backup_wallet\(\)](#), [mc_change_wallet_passphrase\(\)](#), [mc_combine_unspent\(\)](#), [mc_dump_privkey\(\)](#), [mc_dump_wallet\(\)](#), [mc_encrypt_wallet\(\)](#), [mc_get_wallet_info\(\)](#), [mc_import_privkey\(\)](#), [mc_import_wallet\(\)](#), [mc_list_lock_unspent\(\)](#), [mc_list_unspent\(\)](#), [mc_lock_unspent\(\)](#), [mc_lock_wallet\(\)](#)

mc_unsubscribe	<i>Unsubscribe from MultiChain assets or streams</i>
----------------	--

Description

Instructs the MultiChain node to stop tracking one or more asset(s) or stream(s).

Usage

```
mc_unsubscribe(conn, entities, purge = FALSE)
```

Arguments

conn	A connection object to the MultiChain node.
entities	A character string or vector of strings representing asset/stream names, references, or transaction IDs (txids).
purge	Logical. If TRUE, any off-chain data retrieved for this stream will be permanently purged from the node's local storage. Defaults to FALSE.

Value

Returns NULL invisibly on success, or an error if the RPC call fails.

See Also

[mc_subscribe](#)

Other subscriptions: [mc_subscribe\(\)](#)

mc_update	<i>Update asset status (open/closed)</i>
-----------	--

Description

Changes the status of an asset (e.g., to open or close further issuances). The asset must have been created with the ability to be updated.

Usage

```
mc_update(conn, asset, params)
```

Arguments

conn	A connection object created by mc_connect .
asset	Character string. Asset name, reference, or issuance ID.
params	A list of parameters to update, typically <code>list(open = FALSE)</code> to close further issuances.

Value

A character string containing the transaction ID.

See Also

[mc_update_from](#) to update from a specific address.

Other assets: [mc_get_asset_info\(\)](#), [mc_get_token_info\(\)](#), [mc_issue\(\)](#), [mc_issue_from\(\)](#), [mc_issue_more\(\)](#), [mc_issue_more_from\(\)](#), [mc_issue_token\(\)](#), [mc_issue_token_from\(\)](#), [mc_list_asset_issues\(\)](#), [mc_list_assets\(\)](#), [mc_update_from\(\)](#)

Examples

```
## Not run:
# Close the asset "mycoin" to prevent further issuances
txid <- mc_update(conn, "mycoin", list(open = FALSE))

## End(Not run)
```

mc_update_from	<i>Update asset status from specific address</i>
----------------	--

Description

Changes the status of an asset (e.g., open/close) and specifies the address that pays for and authorizes the update.

Usage

```
mc_update_from(conn, from_address, asset, params)
```

Arguments

conn	A connection object created by mc_connect .
from_address	Character string. Address that pays for the transaction.
asset	Character string. Asset name, reference, or issuance ID.
params	A list of parameters to update.

Value

A character string containing the transaction ID.

See Also

[mc_update](#) for simpler usage.

Other assets: [mc_get_asset_info\(\)](#), [mc_get_token_info\(\)](#), [mc_issue\(\)](#), [mc_issue_from\(\)](#), [mc_issue_more\(\)](#), [mc_issue_more_from\(\)](#), [mc_issue_token\(\)](#), [mc_issue_token_from\(\)](#), [mc_list_asset_issues\(\)](#), [mc_list_assets\(\)](#), [mc_update\(\)](#)

Examples

```
## Not run:
# Close the asset from a specific address
txid <- mc_update_from(conn, "1A...", "mycoin", list(open = FALSE))

## End(Not run)
```

mc_validate_address	<i>Validate or inspect an address</i>
---------------------	---------------------------------------

Description

Returns information about a given address, private key, or public key. Useful for checking whether an address is valid, whether it belongs to the current node, and for inspecting its associated redeem script.

Usage

```
mc_validate_address(conn, address_or_key)
```

Arguments

`conn` A connection object created by [mc_connect](#).

`address_or_key` Character string. An address, private key, or public key.

Value

A list with information about the input, including:

<code>isvalid</code>	Logical indicating whether the input is valid.
<code>address</code>	The canonical address (if valid).
<code>ismine</code>	Logical indicating whether the address belongs to the node.
<code>...</code>	Other fields depending on the input type (e.g., pubkey, script).

See Also

[mc_import_address](#) to add an address to the wallet.

Other addresses: [mc_add_multisig_address\(\)](#), [mc_create_keypairs\(\)](#), [mc_create_multisig\(\)](#), [mc_get_addresses\(\)](#), [mc_get_new_address\(\)](#), [mc_import_address\(\)](#), [mc_list_addresses\(\)](#)

Examples

```
## Not run:
# Validate an address
info <- mc_validate_address(conn, "1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa")
print(info$isvalid)

# Validate a private key (if known)
key_info <- mc_validate_address(conn, "L5...")

## End(Not run)
```

mc_verify_message	<i>Verify a signed message</i>
-------------------	--------------------------------

Description

Checks whether a message was signed by the owner of a given address. The signature must have been created by [mc_sign_message](#).

Usage

```
mc_verify_message(conn, address, signature, message)
```

Arguments

conn	A connection object created by mc_connect .
address	Character string. The address that allegedly signed the message.
signature	Character string. The base64-encoded signature (as returned by mc_sign_message).
message	Character string. The original text message.

Value

A logical value: TRUE if the signature is valid, FALSE otherwise.

See Also

[mc_sign_message](#) to create a signature.

Other cryptography: [mc_sign_message\(\)](#)

Examples

```
## Not run:
# Sign a message
sig <- mc_sign_message(conn, "1A...", "Hello")
# Verify it
valid <- mc_verify_message(conn, "1A...", sig, "Hello")
print(valid) # should be TRUE

## End(Not run)
```

mc_verify_permission *Verify if an address has a specific permission*

Description

Checks whether a given address has a particular permission on the blockchain.

Usage

```
mc_verify_permission(conn, address, permission)
```

Arguments

conn	A connection object created by mc_connect .
address	Character string. The address to check.
permission	Character string. The permission name (e.g., "send", "admin", "connect").

Value

A logical value: TRUE if the address has the permission, FALSE otherwise.

See Also

[mc_list_permissions](#) to see all permissions.

Other permissions: [mc_grant\(\)](#), [mc_grant_from\(\)](#), [mc_grant_with_data\(\)](#), [mc_grant_with_data_from\(\)](#), [mc_list_permissions\(\)](#), [mc_revoke\(\)](#), [mc_revoke_from\(\)](#)

Examples

```
## Not run:
# Check if an address can send
can_send <- mc_verify_permission(conn, "1A...", "send")
if (can_send) cat("Address can send assets")

## End(Not run)
```

`mc_wait_for_confirmation`
Wait for transaction confirmation

Description

Blocks execution until a transaction is included in a block.

Usage

```
mc_wait_for_confirmation(conn, txid, timeout = 30)
```

Arguments

<code>conn</code>	A connection object.
<code>txid</code>	Character string. Transaction ID.
<code>timeout</code>	Integer. Maximum time to wait in seconds (default 30).

Value

Logical TRUE if confirmed, throws error if timeout reached.

`null_default`
Null-default operator

Description

This infix operator provides a convenient way to handle NULL values by providing a default value.

Usage

```
a %||% b
```

Arguments

<code>a</code>	An object to check for NULL.
<code>b</code>	The default value to return if <code>a</code> is NULL.

Value

`a` if it is not NULL, otherwise `b`.

Examples

```
"value" %||% "default"
NULL %||% "default"
```

print.mc_help	<i>Print MultiChain help</i>
---------------	------------------------------

Description

S3 method for printing objects returned by [mc_help](#). Displays the help text in a clean format.

Usage

```
## S3 method for class 'mc_help'  
print(x, ...)
```

Arguments

x	An object of class "mc_help".
...	Additional arguments (ignored).

Value

Invisibly returns x.

print.multichain_conn	<i>Print MultiChain connection</i>
-----------------------	------------------------------------

Description

S3 method for printing multichain_conn objects. Hides the password for security.

Usage

```
## S3 method for class 'multichain_conn'  
print(x, ...)
```

Arguments

x	An object of class "multichain_conn".
...	Additional arguments passed to print (ignored).

Value

Invisibly returns the object x.

See Also

[mc_connect](#) for creating connections.

Examples

```
## Not run:  
conn <- mc_connect(config)  
print(conn)  # or simply conn  
  
## End(Not run)
```

Index

* addresses

- mc_add_multisig_address, 8
- mc_create_keypairs, 21
- mc_create_multisig, 23
- mc_get_addresses, 42
- mc_get_new_address, 59
- mc_import_address, 80
- mc_list_addresses, 90
- mc_validate_address, 153

* asset transactions

- mc_get_asset_transaction, 44
- mc_list_asset_transactions, 92

* assets

- mc_get_asset_info, 43
- mc_get_token_info, 68
- mc_issue, 82
- mc_issue_from, 83
- mc_issue_more, 84
- mc_issue_more_from, 85
- mc_issue_token, 86
- mc_issue_token_from, 87
- mc_list_asset_issues, 91
- mc_list_assets, 93
- mc_update, 151
- mc_update_from, 152

* atomic exchange

- mc_append_raw_exchange, 12
- mc_complete_raw_exchange, 18
- mc_create_raw_exchange, 24
- mc_decode_raw_exchange, 34
- mc_disable_raw_transaction, 37
- mc_prepare_lock_unspent, 119
- mc_prepare_lock_unspent_from, 120

* binary cache

- mc_append_binary_cache, 10
- mc_create_binary_cache, 20
- mc_delete_binary_cache, 36
- mc_txout_to_binary_cache, 149

* blockchain information

- mc_get_block, 45
- mc_get_block_hash, 46
- mc_get_blockchain_info, 46
- mc_get_chain_totals, 48
- mc_get_last_block_info, 54
- mc_list_blocks, 94
- mc_list_miners, 96

* configuration

- mc_get_config, 51

* connection management

- mc_connect, 19

* cryptography

- mc_sign_message, 142
- mc_verify_message, 154

* filters

- mc_approve_from, 14
- mc_create_stream_filter, 28
- mc_create_tx_filter, 30
- mc_create_upgrade, 31
- mc_get_filter_code, 52
- mc_list_stream_filters, 100
- mc_list_tx_filters, 109
- mc_list_upgrades, 110
- mc_run_stream_filter, 128
- mc_run_tx_filter, 129
- mc_test_stream_filter, 147
- mc_test_tx_filter, 148

* libraries

- mc_add_library_update, 6
- mc_add_library_update_from, 7
- mc_create_library, 22
- mc_get_library_code, 55
- mc_list_libraries, 95
- mc_test_library, 146

* mempool & transactions

- mc_get_mempool_info, 56
- mc_get_raw_mempool, 60
- mc_get_raw_transaction, 61
- mc_get_tx_out, 69

- * **mempool management**
 - mc_clear_mempool, 16
- * **networking**
 - mc_add_node, 9
 - mc_get_added_node_info, 40
 - mc_get_network_info, 58
 - mc_get_peer_info, 59
 - mc_list_stored_nodes, 98
 - mc_ping, 118
 - mc_store_node, 144
- * **node configuration**
 - mc_get_blockchain_params, 47
 - mc_get_runtime_params, 62
 - mc_set_runtime_param, 139
- * **node control**
 - mc_pause, 117
 - mc_resume, 125
 - mc_set_last_block, 137
- * **node information**
 - mc_get_info, 52
 - mc_get_init_status, 53
- * **node operations**
 - mc_node_init, 114
 - mc_node_start, 115
 - mc_node_stop, 116
- * **node utilities**
 - mc_help, 79
- * **off-chain data**
 - mc_get_chunk_queue_info, 49
 - mc_get_chunk_queue_totals, 50
- * **path management**
 - mc_set_path, 138
- * **permissions**
 - mc_grant, 75
 - mc_grant_from, 76
 - mc_grant_with_data, 77
 - mc_grant_with_data_from, 78
 - mc_list_permissions, 97
 - mc_revoke, 126
 - mc_revoke_from, 127
 - mc_verify_permission, 155
- * **raw transactions**
 - mc_append_raw_change, 10
 - mc_append_raw_data, 11
 - mc_append_raw_transaction, 13
 - mc_create_raw_send_from, 25
 - mc_create_raw_transaction, 26
 - mc_decode_raw_transaction, 35
 - mc_send_raw_transaction, 134
 - mc_sign_raw_transaction, 143
- * **streams**
 - mc_create_stream, 27
 - mc_create_stream_from, 29
 - mc_get_stream_info, 63
 - mc_get_stream_item, 64
 - mc_get_stream_key_summary, 65
 - mc_get_stream_publisher_summary, 66
 - mc_list_stream_block_items, 99
 - mc_list_stream_items, 101
 - mc_list_stream_key_items, 102
 - mc_list_stream_keys, 103
 - mc_list_stream_publisher_items, 104
 - mc_list_stream_publishers, 105
 - mc_list_stream_query_items, 106
 - mc_list_stream_tx_items, 107
 - mc_list_streams, 108
 - mc_publish, 121
 - mc_publish_from, 122
 - mc_publish_multi, 123
 - mc_publish_multi_from, 124
- * **subscriptions**
 - mc_subscribe, 145
 - mc_unsubscribe, 151
- * **transactions**
 - mc_get_address_balances, 41
 - mc_get_address_transaction, 41
 - mc_get_multi_balances, 57
 - mc_get_token_balances, 67
 - mc_get_total_balances, 69
 - mc_get_tx_out_data, 70
 - mc_get_wallet_transaction, 74
 - mc_list_address_transactions, 89
 - mc_list_wallet_transactions, 112
 - mc_send, 130
 - mc_send_asset, 131
 - mc_send_asset_from, 132
 - mc_send_from, 133
 - mc_send_with_data, 135
 - mc_send_with_data_from, 136
- * **variables**
 - mc_create_variable, 32
 - mc_create_variable_from, 33
 - mc_get_variable_history, 71
 - mc_get_variable_info, 72

- mc_get_variable_value, 73
- mc_list_variables, 111
- mc_set_variable_value, 140
- mc_set_variable_value_from, 141
- * **wallet**
 - mc_backup_wallet, 15
 - mc_change_wallet_passphrase, 16
 - mc_combine_unspent, 17
 - mc_dump_privkey, 38
 - mc_dump_wallet, 38
 - mc_encrypt_wallet, 39
 - mc_get_wallet_info, 73
 - mc_import_privkey, 81
 - mc_import_wallet, 81
 - mc_list_lock_unspent, 96
 - mc_list_unspent, 110
 - mc_lock_unspent, 113
 - mc_lock_wallet, 114
 - mc_unlock_wallet, 150
- hex_to_char, 5
- mc_add_library_update, 6, 7, 22, 55, 95, 146
- mc_add_library_update_from, 6, 7, 22, 55, 95, 146
- mc_add_multisig_address, 8, 21, 23, 42, 59, 80, 90, 153
- mc_add_node, 9, 40, 58, 60, 98, 118, 144
- mc_append_binary_cache, 10, 20, 36, 149
- mc_append_raw_change, 10, 12, 14, 25, 26, 35, 135, 143
- mc_append_raw_data, 11, 11, 14, 25, 26, 35, 135, 143
- mc_append_raw_exchange, 12, 18, 24, 34, 37, 119, 120
- mc_append_raw_transaction, 11, 12, 13, 25, 26, 35, 135, 143
- mc_approve_from, 14, 28, 30, 31, 52, 100, 109, 111, 128, 129, 147, 148
- mc_backup_wallet, 15, 16, 18, 38, 39, 74, 81, 82, 96, 110, 113, 114, 150
- mc_change_wallet_passphrase, 15, 16, 18, 38, 39, 74, 81, 82, 96, 110, 113, 114, 150
- mc_clear_mempool, 16, 117
- mc_combine_unspent, 15, 16, 17, 38, 39, 74, 81, 82, 96, 110, 113, 114, 150
- mc_complete_raw_exchange, 13, 18, 24, 34, 37, 119, 120
- mc_connect, 6–14, 16, 18, 19, 20–37, 40, 42–56, 58–66, 68, 70–73, 75–80, 82, 83, 85–88, 90–95, 97–111, 116–144, 146–149, 151–155, 157
- mc_create_binary_cache, 10, 20, 36, 149
- mc_create_keypairs, 8, 21, 23, 42, 59, 80, 90, 153
- mc_create_library, 6, 7, 22, 55, 95, 146
- mc_create_multisig, 8, 21, 23, 42, 59, 80, 90, 153
- mc_create_raw_exchange, 13, 18, 24, 34, 37, 119, 120
- mc_create_raw_send_from, 11, 12, 14, 25, 26, 35, 135, 143
- mc_create_raw_transaction, 11, 12, 14, 25, 26, 35, 135, 143
- mc_create_stream, 27, 29, 63–66, 99, 101, 103–108, 121–124
- mc_create_stream_filter, 15, 28, 30, 31, 52, 100, 109, 111, 128, 129, 147, 148
- mc_create_stream_from, 27, 29, 63–66, 99, 101, 103–108, 121–124
- mc_create_tx_filter, 15, 28, 30, 31, 52, 100, 109, 111, 128, 129, 147, 148
- mc_create_upgrade, 15, 28, 30, 31, 52, 100, 109, 111, 128, 129, 147, 148
- mc_create_variable, 32, 33, 72, 73, 112, 140, 141
- mc_create_variable_from, 32, 33, 72, 73, 112, 140, 141
- mc_decode_raw_exchange, 13, 18, 24, 34, 37, 119, 120
- mc_decode_raw_transaction, 11, 12, 14, 25, 26, 35, 135, 143
- mc_delete_binary_cache, 10, 20, 36, 149
- mc_disable_raw_transaction, 13, 18, 24, 34, 37, 119, 120
- mc_dump_privkey, 15, 16, 18, 38, 39, 74, 81, 82, 96, 110, 113, 114, 150
- mc_dump_wallet, 15, 16, 18, 38, 39, 74, 81, 82, 96, 110, 113, 114, 150
- mc_encrypt_wallet, 15, 16, 18, 38, 39, 39, 74, 81, 82, 96, 110, 113, 114, 150
- mc_get_added_node_info, 9, 40, 58, 60, 98, 118, 144
- mc_get_address_balances, 41, 42, 57, 67, 69, 71, 74, 89, 113, 130, 131, 133, 134, 136, 137

- mc_get_address_transaction, [41](#), [41](#), [57](#),
[67](#), [69](#), [71](#), [74](#), [89](#), [113](#), [130](#), [131](#),
[133](#), [134](#), [136](#), [137](#)
- mc_get_addresses, [8](#), [21](#), [23](#), [42](#), [59](#), [80](#), [90](#),
[153](#)
- mc_get_asset_info, [43](#), [68](#), [83–88](#), [91](#), [94](#),
[152](#)
- mc_get_asset_transaction, [44](#), [93](#)
- mc_get_block, [45](#), [46](#), [47](#), [49](#), [55](#), [95](#), [97](#), [137](#)
- mc_get_block_hash, [45](#), [46](#), [47](#), [49](#), [55](#), [95](#), [97](#)
- mc_get_blockchain_info, [45](#), [46](#), [46](#), [49](#), [53](#),
[55](#), [95](#), [97](#)
- mc_get_blockchain_params, [47](#), [62](#), [139](#)
- mc_get_chain_totals, [45–47](#), [48](#), [55](#), [95](#), [97](#)
- mc_get_chunk_queue_info, [49](#), [50](#)
- mc_get_chunk_queue_totals, [49](#), [50](#)
- mc_get_config, [19](#), [51](#)
- mc_get_filter_code, [15](#), [28](#), [30](#), [31](#), [52](#), [55](#),
[100](#), [109](#), [111](#), [128](#), [129](#), [147](#), [148](#)
- mc_get_info, [52](#), [54](#)
- mc_get_init_status, [53](#), [53](#)
- mc_get_last_block_info, [45–47](#), [49](#), [54](#), [95](#),
[97](#)
- mc_get_library_code, [6](#), [7](#), [22](#), [52](#), [55](#), [95](#),
[146](#)
- mc_get_mempool_info, [17](#), [56](#), [61](#), [62](#), [70](#)
- mc_get_multi_balances, [41](#), [42](#), [57](#), [67](#), [69](#),
[71](#), [74](#), [89](#), [113](#), [130](#), [131](#), [133](#), [134](#),
[136](#), [137](#)
- mc_get_network_info, [9](#), [40](#), [58](#), [60](#), [98](#), [118](#),
[144](#)
- mc_get_new_address, [8](#), [21](#), [23](#), [42](#), [59](#), [80](#),
[90](#), [142](#), [153](#)
- mc_get_peer_info, [9](#), [40](#), [58](#), [59](#), [98](#), [118](#), [144](#)
- mc_get_raw_mempool, [56](#), [60](#), [62](#), [70](#)
- mc_get_raw_transaction, [35](#), [56](#), [61](#), [61](#), [70](#)
- mc_get_runtime_params, [48](#), [53](#), [62](#), [139](#)
- mc_get_stream_info, [27](#), [29](#), [63](#), [64–66](#), [99](#),
[101](#), [103–108](#), [121–124](#)
- mc_get_stream_item, [27](#), [29](#), [63](#), [64](#), [65](#), [66](#),
[99](#), [101](#), [103–108](#), [121–124](#)
- mc_get_stream_key_summary, [27](#), [29](#), [63](#), [64](#),
[65](#), [66](#), [99](#), [101](#), [103–108](#), [121–124](#)
- mc_get_stream_publisher_summary, [27](#), [29](#),
[63–65](#), [66](#), [99](#), [101](#), [103–108](#),
[121–124](#)
- mc_get_token_balances, [41](#), [42](#), [57](#), [67](#), [69](#),
[71](#), [74](#), [89](#), [113](#), [130](#), [131](#), [133](#), [134](#),
[136](#), [137](#)
- mc_get_token_info, [43](#), [68](#), [83–88](#), [91](#), [94](#),
[152](#)
- mc_get_total_balances, [41](#), [42](#), [57](#), [67](#), [69](#),
[71](#), [74](#), [89](#), [113](#), [130](#), [131](#), [133](#), [134](#),
[136](#), [137](#)
- mc_get_tx_out, [56](#), [61](#), [62](#), [69](#)
- mc_get_tx_out_data, [41](#), [42](#), [57](#), [67](#), [69](#), [70](#),
[74](#), [89](#), [113](#), [130](#), [131](#), [133](#), [134](#), [136](#),
[137](#)
- mc_get_variable_history, [32](#), [33](#), [71](#), [72](#),
[73](#), [112](#), [140](#), [141](#)
- mc_get_variable_info, [32](#), [33](#), [72](#), [72](#), [73](#),
[112](#), [140](#), [141](#)
- mc_get_variable_value, [32](#), [33](#), [72](#), [73](#), [112](#),
[140](#), [141](#)
- mc_get_wallet_info, [15](#), [16](#), [18](#), [38](#), [39](#), [73](#),
[81](#), [82](#), [96](#), [110](#), [113](#), [114](#), [150](#)
- mc_get_wallet_transaction, [41](#), [42](#), [57](#), [67](#),
[69](#), [71](#), [74](#), [89](#), [113](#), [130](#), [131](#), [133](#),
[134](#), [136](#), [137](#)
- mc_grant, [75](#), [76](#), [77](#), [79](#), [98](#), [126](#), [127](#), [155](#)
- mc_grant_from, [75](#), [76](#), [77–79](#), [98](#), [126](#), [127](#),
[155](#)
- mc_grant_with_data, [75](#), [76](#), [77](#), [78](#), [79](#), [98](#),
[126](#), [127](#), [155](#)
- mc_grant_with_data_from, [75–77](#), [78](#), [98](#),
[126](#), [127](#), [155](#)
- mc_help, [79](#), [157](#)
- mc_import_address, [8](#), [21](#), [23](#), [42](#), [59](#), [80](#), [90](#),
[153](#)
- mc_import_privkey, [15](#), [16](#), [18](#), [38](#), [39](#), [74](#),
[81](#), [82](#), [96](#), [110](#), [113](#), [114](#), [150](#)
- mc_import_wallet, [15](#), [16](#), [18](#), [38](#), [39](#), [74](#), [81](#),
[81](#), [96](#), [110](#), [113](#), [114](#), [150](#)
- mc_issue, [43](#), [68](#), [82](#), [84–88](#), [91](#), [94](#), [152](#)
- mc_issue_from, [43](#), [68](#), [83](#), [83](#), [85–88](#), [91](#), [94](#),
[152](#)
- mc_issue_more, [43](#), [68](#), [83](#), [84](#), [84](#), [86–88](#), [91](#),
[94](#), [152](#)
- mc_issue_more_from, [43](#), [68](#), [83](#), [84](#), [85](#), [85](#),
[87](#), [88](#), [91](#), [94](#), [152](#)
- mc_issue_token, [43](#), [68](#), [83–85](#), [86](#), [86](#), [88](#),
[91](#), [94](#), [152](#)
- mc_issue_token_from, [43](#), [68](#), [83–86](#), [87](#), [87](#),
[91](#), [94](#), [152](#)
- mc_list_address_transactions, [41](#), [42](#), [57](#),
[67](#), [69](#), [71](#), [74](#), [89](#), [113](#), [130](#), [131](#),

- [133, 134, 136, 137](#)
- `mc_list_addresses`, [8, 21, 23, 42, 59, 80, 90, 153](#)
- `mc_list_asset_issues`, [43, 68, 83–88, 91, 94, 152](#)
- `mc_list_asset_transactions`, [44, 92](#)
- `mc_list_assets`, [43, 68, 83–88, 91, 93, 152](#)
- `mc_list_blocks`, [45–47, 49, 55, 94, 97](#)
- `mc_list_libraries`, [6, 7, 22, 55, 95, 146](#)
- `mc_list_lock_unspent`, [15, 16, 18, 38, 39, 74, 81, 82, 96, 110, 113, 114, 150](#)
- `mc_list_miners`, [45–47, 49, 55, 95, 96](#)
- `mc_list_permissions`, [75–77, 79, 97, 126, 127, 155](#)
- `mc_list_stored_nodes`, [9, 40, 58, 60, 98, 118, 144](#)
- `mc_list_stream_block_items`, [27, 29, 63–66, 99, 101, 103–108, 121–124](#)
- `mc_list_stream_filters`, [15, 28, 30, 31, 52, 100, 109, 111, 128, 129, 147, 148](#)
- `mc_list_stream_items`, [28, 29, 63–66, 99, 101, 103–108, 121–124](#)
- `mc_list_stream_key_items`, [28, 29, 63–66, 99, 101, 102, 104–108, 121–124](#)
- `mc_list_stream_keys`, [28, 29, 63–66, 99, 101, 103, 103, 105–108, 121–124](#)
- `mc_list_stream_publisher_items`, [28, 29, 63–66, 99, 101, 103, 104, 104, 106–108, 121–124](#)
- `mc_list_stream_publishers`, [28, 29, 63–66, 99, 101, 103, 104, 105, 105, 107, 108, 121–124](#)
- `mc_list_stream_query_items`, [28, 29, 63–66, 99, 101, 103–105, 106, 106–108, 121–124](#)
- `mc_list_stream_tx_items`, [28, 29, 63–66, 99, 101, 103–106, 107, 107, 108, 121–124](#)
- `mc_list_streams`, [28, 29, 63–66, 99, 101, 103–107, 108, 121–124](#)
- `mc_list_tx_filters`, [15, 28, 30, 31, 52, 100, 109, 111, 128, 129, 147, 148](#)
- `mc_list_unspent`, [15, 16, 18, 38, 39, 74, 81, 82, 96, 110, 113, 114, 150](#)
- `mc_list_upgrades`, [15, 28, 30, 31, 52, 100, 109, 110, 128, 129, 147, 148](#)
- `mc_list_variables`, [32, 33, 72, 73, 111, 140, 141](#)
- `mc_list_wallet_transactions`, [41, 42, 57, 67, 69, 71, 74, 89, 112, 130, 131, 133, 134, 136, 137](#)
- `mc_lock_unspent`, [15, 16, 18, 38, 39, 74, 81, 82, 96, 110, 113, 114, 150](#)
- `mc_lock_wallet`, [15, 16, 18, 38, 39, 74, 81, 82, 96, 110, 113, 114, 150](#)
- `mc_node_init`, [114, 115, 116](#)
- `mc_node_start`, [115, 115, 116](#)
- `mc_node_stop`, [115, 116](#)
- `mc_pause`, [17, 117, 125, 137](#)
- `mc_ping`, [9, 40, 58, 60, 98, 118, 144](#)
- `mc_prepare_lock_unspent`, [13, 18, 24, 34, 37, 119, 120](#)
- `mc_prepare_lock_unspent_from`, [13, 18, 24, 34, 37, 119, 120](#)
- `mc_publish`, [28, 29, 63–66, 99, 101, 103–108, 121, 122–124, 149](#)
- `mc_publish_from`, [28, 29, 63–66, 99, 101, 103–108, 121, 122, 123, 124](#)
- `mc_publish_multi`, [28, 29, 63–66, 99, 101, 103–108, 121, 122, 123, 124](#)
- `mc_publish_multi_from`, [28, 29, 63–66, 99, 101, 103–108, 121–123, 124](#)
- `mc_resume`, [17, 117, 125, 137](#)
- `mc_revoke`, [75–77, 79, 98, 126, 127, 155](#)
- `mc_revoke_from`, [75–77, 79, 98, 126, 127, 155](#)
- `mc_run_stream_filter`, [15, 28, 30, 31, 52, 100, 109, 111, 128, 129, 147, 148](#)
- `mc_run_tx_filter`, [15, 28, 30, 31, 52, 100, 109, 111, 128, 129, 147, 148](#)
- `mc_send`, [41, 42, 57, 67, 69, 71, 74, 89, 113, 130, 131, 133, 134, 136, 137](#)
- `mc_send_asset`, [41, 42, 57, 67, 69, 71, 74, 89, 113, 130, 131, 133, 134, 136, 137](#)
- `mc_send_asset_from`, [41, 42, 57, 67, 69, 71, 74, 89, 113, 130, 131, 132, 134, 136, 137](#)
- `mc_send_from`, [41, 42, 57, 67, 69, 71, 74, 89, 113, 130, 131, 133, 133, 136, 137](#)
- `mc_send_raw_transaction`, [11, 12, 14, 18, 25, 26, 35, 134, 143](#)
- `mc_send_with_data`, [41, 42, 57, 67, 69, 71, 74, 89, 113, 130, 131, 133, 134, 135, 136, 137](#)
- `mc_send_with_data_from`, [41, 42, 57, 67, 69, 71, 74, 89, 113, 130, 131, 133, 134, 136, 136](#)

mc_set_last_block, [117](#), [125](#), [137](#)
mc_set_path, [138](#)
mc_set_runtime_param, [48](#), [62](#), [139](#)
mc_set_variable_value, [32](#), [33](#), [72](#), [73](#), [112](#),
[140](#), [141](#)
mc_set_variable_value_from, [32](#), [33](#), [72](#),
[73](#), [112](#), [140](#), [141](#)
mc_sign_message, [142](#), [154](#)
mc_sign_raw_transaction, [11](#), [12](#), [14](#), [25](#),
[26](#), [35](#), [135](#), [143](#)
mc_store_node, [9](#), [40](#), [58](#), [60](#), [98](#), [118](#), [144](#)
mc_subscribe, [145](#), [151](#)
mc_test_library, [6](#), [7](#), [22](#), [55](#), [95](#), [146](#)
mc_test_stream_filter, [15](#), [28](#), [30](#), [31](#), [52](#),
[100](#), [109](#), [111](#), [128](#), [129](#), [147](#), [148](#)
mc_test_tx_filter, [15](#), [28](#), [30](#), [31](#), [52](#), [100](#),
[109](#), [111](#), [128](#), [129](#), [147](#), [148](#)
mc_txout_to_binary_cache, [10](#), [20](#), [36](#), [149](#)
mc_unlock_wallet, [15](#), [16](#), [18](#), [38](#), [39](#), [74](#), [81](#),
[82](#), [96](#), [110](#), [113](#), [114](#), [150](#)
mc_unsubscribe, [145](#), [151](#)
mc_update, [43](#), [68](#), [83–88](#), [91](#), [94](#), [151](#), [152](#)
mc_update_from, [43](#), [68](#), [83–88](#), [91](#), [94](#), [152](#),
[152](#)
mc_validate_address, [8](#), [21](#), [23](#), [42](#), [59](#), [80](#),
[90](#), [153](#)
mc_verify_message, [142](#), [154](#)
mc_verify_permission, [75–77](#), [79](#), [98](#), [126](#),
[127](#), [155](#)
mc_wait_for_confirmation, [156](#)

null_default, [156](#)

print.mc_help, [79](#), [157](#)
print.multichain_conn, [19](#), [157](#)