BBc1 Documentation

Release 1.1

beyond-blockchain.org

Nov 21, 2018

Contents:

CHAPTER 1

bbc1 package

1.1 Subpackages

1.1.1 bbc1.core package

1.1.1.1 Subpackages

1.1.1.2 Submodules

bbc1.core.bbc_app module

class `bbc1.core.bbc_app.`**`BBcAppClient`**(*host='127.0.0.1', port=9000, multiq=True, logname='-', loglevel='none'*)

Bases: `object`

Basic functions for a client of bbc_core

`cancel_insert_completion_notification`(*asset_group_id*)

Cancel notification when a transaction has been inserted (as a copy of transaction)

Parameters **`asset_group_id`** (*`bytes`*) – asset_group_id for requesting notification about insertion

Returns query_id

Return type bytes

count_transactions(*asset_group_id=None*, *asset_id=None*, *user_id=None*)
Count transactions that matches the given conditions

If multiple conditions are specified, they are considered as AND condition.

Parameters

- **asset_group_id** (*bytes*) – asset_group_id in BBcEvent and BBcRelations
- **asset_id** (*bytes*) – asset_id in BBcAsset
- **user_id** (*bytes*) – user_id in BBcAsset that means the owner of the asset

Returns the number of transactions

Return type int

domain_close(*domain_id=None*)
Close domain leading to remove_domain in the core

Parameters **domain_id** (*bytes*) – domain_id to delete

Returns query_id

Return type bytes

domain_setup(*domain_id*, *config=None*)
Set up domain with the specified network module and storage

This method should be used by a system administrator.

Parameters

- **domain_id** (*bytes*) – domain_id to create
- **config** (*str*) – system config in json format

Returns query_id

Return type bytes

exchange_key()
Perform ECDH (key exchange algorithm)

Returns query_id

Return type bytes

gather_signatures(*txobj*, *reference_obj=None*, *asset_files=None*, *destinations=None*, *anycast=False*)
Request to gather signatures from the specified user_ids

Parameters

- **txobj** (BBcTransaction) –
- **reference_obj** (BBcReference) – BBcReference object that includes the information about destinations
- **asset_files** (*dict*) – mapping from asset_id to its file content
- **destinations** (*list*) – list of destination user_ids
- **anycast** (*bool*) – True if this message is for anycasting

Returns query_id

Return type bytes

get_bbc_config()
Get config file of bbc_core

This method should be used by a system administrator.

Returns query_id

Return type bytes

get_domain_list()
Get domain_id list in bbc_core

Returns query_id

Return type bytes

get_domain_neighborlist(*domain_id*)
Get peer list of the domain from the core node

This method should be used by a system administrator.

Parameters **domain_id** (*bytes*) – domain_id of the neighbor list

Returns query_id

Return type bytes

get_forwarding_list()
Get forwarding_list of the domain in the core node

Returns query_id

Return type bytes

get_node_id()
Get node_id of the connecting core node

Returns query_id

Return type bytes

get_notification_list()
Get notification_list of the core node

Returns query_id

Return type bytes

get_stats()
Get statistics of bbc_core

Returns query_id

Return type bytes

get_user_list()
Get user_ids in the domain that are connecting to the core node

Returns query_id

Return type bytes

include_admin_info(*dat*, *admin_info*, *keypair*)

include_cross_ref(*txobj*)
Include BBcCrossRef from other domains in the transaction

If the client object has one or more cross_ref objects, one of them is included in the given transaction. This method should be voluntarily called for inter-domain weak collaboration.

Parameters **`txobj`** (`BBcTransaction`) – Transaction object to include cross_ref

`insert_transaction`(*tx_obj*)

Request to insert a legitimate transaction

Parameters **`tx_obj`** (`BBcTransaction`) – Transaction object to insert

Returns query_id

Return type bytes

`manipulate_ledger_subsystem`(*enable=False*, *domain_id=None*)

Start/stop ledger_subsystem on the bbc_core

This method should be used by a system administrator.

Parameters

- **`enable`** (*`bool`*) – True->start, False->stop
- **`domain_id`** (*`bytes`*) – target domain_id to enable/disable ledger_subsystem

Returns query_id

Return type bytes

`notify_domain_key_update`()

Notify update of bbc_core

This method should be used by a system administrator.

Returns query_id

Return type bytes

`receiver_loop`()

`register_in_ledger_subsystem`(*asset_group_id*, *transaction_id*)

Register transaction_id in the ledger_subsystem

Parameters

- **`asset_group_id`** (*`bytes`*) –
- **`transaction_id`** (*`bytes`*) – the target transaction_id

Returns query_id

Return type bytes

`register_to_core`(*on_multiple_nodes=False*)

Register the client (user_id) to the core node

After that, the client can communicate with the core node.

Parameters **`on_multiple_nodes`** (*`bool`*) – True if this user_id is for multicast address

Returns True

Return type bool

`request_cross_ref_holders_list`()

Request the list of transaction_ids that are registered as cross_ref in outer domains

Returns query_id

Return type bytes

`request_insert_completion_notification`(*asset_group_id*)

Request notification when a transaction has been inserted (as a copy of transaction)

Parameters **`asset_group_id`** (*`bytes`*) – asset_group_id for requesting notification about insertion

Returns query_id

Return type bytes

`request_to_repair_asset`(*asset_group_id*, *asset_id*)

Request to repair compromised asset file

Parameters

- **`asset_group_id`** (*`bytes`*) – the asset_group_id of the target asset
- **`asset_id`** (*`bytes`*) – the target asset_id

Returns query_id

Return type bytes

`request_to_repair_transaction`(*transaction_id*)

Request to repair compromised transaction data

Parameters **`transaction_id`** (*`bytes`*) – the target transaction to repair

Returns query_id

Return type bytes

`request_verify_by_cross_ref`(*transaction_id*)

Request to verify the transaction by Cross_ref in transaction of outer domain

Parameters **`transaction_id`** (*`bytes`*) – the target transaction_id

Returns query_id

Return type bytes

`search_transaction`(*transaction_id*)

Search request for a transaction

Parameters **`transaction_id`** (*`bytes`*) – the target transaction to retrieve

Returns query_id

Return type bytes

`search_transaction_with_condition`(*asset_group_id=None*, *asset_id=None*, *user_id=None*, *direction=0*, *count=1*)

Search transaction data by asset_group_id/asset_id/user_id

If multiple conditions are specified, they are considered as AND condition.

Parameters

- **`asset_group_id`** (*`bytes`*) – asset_group_id in BBcEvent and BBcRelations
- **`asset_id`** (*`bytes`*) – asset_id in BBcAsset
- **`user_id`** (*`bytes`*) – user_id in BBcAsset that means the owner of the asset
- **`direction`** (*`int`*) – 0: descend, 1: ascend
- **`count`** (*`int`*) – the number of transactions to retrieve

Returns query_id

Return type bytes

send_domain_ping(*domain_id*, *ipv4=None*, *ipv6=None*, *port=6641*)
Send domain ping to notify the existence of the node

This method should be used by a system administrator.

Parameters

- **domain_id** (*bytes*) – target domain_id to send ping
- **ipv4** (*str*) – IPv4 address of the node
- **ipv6** (*str*) – IPv6 address of the node
- **port** (*int*) – Port number to wait messages UDP

Returns query_id

Return type bytes

send_message(*msg*, *dst_user_id*, *is_anycast=False*)
Send a message to the specified user_id

Parameters

- **msg** (*dict*) – message to send
- **dst_user_id** (*bytes*) – destination user_id
- **is_anycast** (*bool*) – If true, the message is treated as an anycast message.

Returns query_id

Return type bytes

sendback_denial_of_sign(*dest_user_id=None*, *transaction_id=None*, *reason_text=None*, *query_id=None*)
Send back the denial of sign the transaction

This method is called if the receiver (signer) denies the transaction.

Parameters

- **dest_user_id** (*bytes*) – destination user_id to send back
- **transaction_id** (*bytes*) –
- **reason_text** (*str*) – message to the requester about why the node denies the transaction
- **query_id** – The query_id that was in the received SIGN_REQUEST message

Returns query_id

Return type bytes

sendback_signature(*dest_user_id=None*, *transaction_id=None*, *ref_index=-1*, *signature=None*, *query_id=None*)
Send back the signed transaction to the source

This method is called if the receiver (signer) approves the transaction.

Parameters

- **dest_user_id** (*bytes*) – destination user_id to send back
- **transaction_id** (*bytes*) –

- **ref_index** (*int*) – (optional) which reference in transaction the signature is for
- **signature** (BBcSignature) – Signature that expresses approval of the transaction with transaction_id
- **query_id** – The query_id that was in the received SIGN_REQUEST message

Returns query_id

Return type bytes

set_callback(*callback_obj*)
Set callback object that implements message processing functions

Parameters **callback_obj** (*obj*) – callback method object

set_domain_id(*domain_id*)
Set domain_id to this client to include it in all messages

Parameters **domain_id** (*bytes*) – domain_id to join in

set_domain_static_node(*domain_id*, *node_id*, *ipv4*, *ipv6*, *port*)
Set static node to the core node

IPv6 is used for socket communication if both IPv4 and IPv6 is specified. This method should be used by a system administrator.

Parameters

- **domain_id** (*bytes*) – target domain_id to set static neighbor entry
- **node_id** (*bytes*) – node_id to register
- **ipv4** (*str*) – IPv4 address of the node
- **ipv6** (*str*) – IPv6 address of the node
- **port** (*int*) – Port number to wait messages (UDP/TCP)

Returns query_id

Return type bytes

set_keypair(*keypair*)
Set keypair for the user

Parameters **keypair** (KeyPair) – KeyPair object for signing

set_node_key(*pem_file=None*)
Set node_key to this client

Parameters **pem_file** (*str*) – path string for the pem file

set_user_id(*identifier*)
Set user_id of the object

Parameters **identifier** (*bytes*) – user_id of this clients

start_receiver_loop()

traverse_transactions(*transaction_id*, *asset_group_id=None*, *user_id=None*, *direction=1*, *hop_count=3*)
Search request for transactions

The method traverses the transaction graph in the ledger. The response from the bbc_core includes the list of transactions.

Parameters

- **`transaction_id`** (*bytes*) – the target transaction to retrieve
- **`asset_group_id`** (*bytes*) – asset_group_id that target transactions should have
- **`user_id`** (*bytes*) – user_id that target transactions should have
- **`direction`** (*int*) – 1:backforward, non-1:forward
- **`hop_count`** (*int*) – hop count to traverse from the specified origin point

Returns query_id

Return type bytes

`unregister_from_core`()
Unregister and disconnect from the core node

Returns True

Return type bool

`verify_in_ledger_subsystem`(*asset_group_id*, *transaction_id*)
Verify transaction_id in the ledger_subsystem

Parameters

- **`asset_group_id`** (*bytes*) –
- **`transaction_id`** (*bytes*) – the target transaction_id

Returns query_id

Return type bytes

class `bbc1.core.bbc_app.`**`Callback`**(*log=None*)
Bases: `object`

Set of callback functions for processing received message

If you want to implement your own way to process messages, inherit this class.

`create_queue`(*query_id*)

`dispatch`(*dat*, *payload_type*)

`get_from_queue`(*query_id*, *timeout=None*, *no_delete=False*)

`proc_cmd_sign_request`(*dat*)
Callback for message REQUEST_SIGNATURE

This method should be overridden if you want to process the message asynchronously.

Parameters **`dat`** (*dict*) – received message

`proc_notify_cross_ref`(*dat*)
Callback for message NOTIFY_CROSS_REF

This method must not be overridden.

Parameters **`dat`** (*dict*) – received message

`proc_notify_inserted`(*dat*)
Callback for message NOTIFY_INSERTED

This method should be overridden if you want to process the message asynchronously.

Parameters **`dat`** (*dict*) – received message

proc_resp_count_transactions(*dat*)
Callback for message RESPONSE_COUNT_TRANSACTIONS

This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (*dict*) – received message

proc_resp_cross_ref_list(*dat*)
Callback for message RESPONSE_CROSS_REF_LIST

This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (*dict*) – received message

proc_resp_domain_close(*dat*)
Callback for message RESPONSE_CLOSE_DOMAIN

This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (*dict*) – received message

proc_resp_domain_setup(*dat*)
Callback for message RESPONSE_SETUP_DOMAIN

This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (*dict*) – received message

proc_resp_ecdh_key_exchange(*dat*)
Callback for message RESPONSE_ECDH_KEY_EXCHANGE

This method must not be overridden.

Parameters **dat** (*dict*) – received message

proc_resp_gather_signature(*dat*)
Callback for message RESPONSE_GATHER_SIGNATURE

This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (*dict*) – received message

proc_resp_get_config(*dat*)
Callback for message RESPONSE_GET_CONFIG

This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (*dict*) – received message

proc_resp_get_domainlist(*dat*)
Callback for message RESPONSE_GET_DOMAINLIST

List of domain_ids is queued rather than message itself. This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (*dict*) – received message

proc_resp_get_forwardinglist(*dat*)
Callback for message RESPONSE_GET_FORWARDING_LIST

List of user_ids in other core nodes is queued rather than message itself. This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (*dict*) – received message

proc_resp_get_neighborlist(*dat*)

Callback for message RESPONSE_GET_NEIGHBORLIST

List of neighbor node info (the first one is that of the connecting core) is queued rather than message itself. This method must not be overridden.

Parameters **dat** (`dict`) – received message

proc_resp_get_node_id(*dat*)

Callback for message RESPONSE_GET_NODEID

Node_id is queued rather than message itself. This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (`dict`) – received message

proc_resp_get_notificationlist(*dat*)

Callback for message RESPONSE_GET_NOTIFICATION_LIST

List of user_ids in other core nodes is queued rather than message itself. This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (`dict`) – received message

proc_resp_get_stats(*dat*)

Callback for message RESPONSE_GET_STATS

This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (`dict`) – received message

proc_resp_get_userlist(*dat*)

Callback for message RESPONSE_GET_USERS

List of user_ids is queued rather than message itself. This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (`dict`) – received message

proc_resp_insert(*dat*)

Callback for message RESPONSE_INSERT

This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (`dict`) – received message

proc_resp_ledger_subsystem(*dat*)

Callback for message RESPONSE_MANIP_LEDGER_SUBSYS

This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (`dict`) – received message

proc_resp_register_hash(*dat*)

Callback for message RESPONSE_REGISTER_HASH_IN_SUBSYS

This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (`dict`) – received message

proc_resp_search_transaction(*dat*)

Callback for message RESPONSE_SEARCH_TRANSACTION

This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (`dict`) – received message

proc_resp_search_with_condition(*dat*)
Callback for message RESPONSE_SEARCH_WITH_CONDITIONS

This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (*dict*) – received message

proc_resp_set_neighbor(*dat*)
Callback for message RESPONSE_SET_STATIC_NODE

This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (*dict*) – received message

proc_resp_sign_request(*dat*)
Callback for message RESPONSE_SIGNATURE

This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (*dict*) – received message

proc_resp_traverse_transactions(*dat*)
Callback for message RESPONSE_TRAVERSE_TRANSACTIONS

This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (*dict*) – received message

proc_resp_verify_cross_ref(*dat*)
Callback for message RESPONSE_CROSS_REF_VERIFY

This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (*dict*) – received message

proc_resp_verify_hash(*dat*)
Callback for message RESPONSE_VERIFY_HASH_IN_SUBSYS

This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (*dict*) – received message

proc_user_message(*dat*)
Callback for message MESSAGE

This method should be overridden if you want to process the message asynchronously.

Parameters **dat** (*dict*) – received message

set_client(*client*)

set_logger(*log*)

sync_by_queryid(*query_id*, *timeout=None*, *no_delete_q=False*)
Wait for the message with specified query_id

This method creates a queue for the query_id and waits for the response

Parameters

- **query_id** (*byte*) – timeout for waiting a message in seconds
- **timeout** (*int*) – timeout for waiting a message in seconds
- **no_delete_q** (*bool*) – If True, the queue for the query_id remains after popping a message

Returns a received message

Return type dict

synchronize(*timeout=None*)
Wait for receiving message with a common queue

Parameters **timeout** (`int`) – timeout for waiting a message in seconds

Returns a received message

Return type dict

bbc1.core.bbc_config module

class bbc1.core.bbc_config.**BBcConfig**(*directory=None*, *file=None*, *default_confpath=None*)
Bases: `object`

System configuration

get_config()
Return config dictionary

get_domain_config(*domain_id*, *create_if_new=False*)
Return the part of specified domain_id in the config dictionary

get_json_config()
Get config in json format

read_config()
Read config file

remove_domain_config(*domain_id*)
Remove the part of specified domain_id in the config dictionary

update_config()
Write config to file (config.json)

bbc1.core.bbc_config.**load_config**(*filepath*)

bbc1.core.bbc_config.**update_deep**(*d*, *u*)
Utility for updating nested dictionary

bbc1.core.bbc_core module

:" .

exec python "$0" "$@"

class `bbc1.core.bbc_core.`**`BBcCoreService`**(*p2p_port=None*, *core_port=None*, *use_domain0=False*, *ip4addr=None*, *ip6addr=None*, *workingdir='.bbc1'*, *configfile=None*, *use_nodekey=None*, *use_ledger_subsystem=False*, *default_conffile=None*, *loglevel='all'*, *logname='-'*, *server_start=True*)

Bases: `object`

Base service object of BBc-1

`count_transactions`(*domain_id*, *asset_group_id=None*, *asset_id=None*, *user_id=None*)

Count transactions that match given conditions

When Multiple conditions are given, they are considered as AND condition.

Parameters

- **`domain_id`** (*`bytes`*) – target domain_id
- **`asset_group_id`** (*`bytes`*) – asset_group_id that target transactions should have
- **`asset_id`** (*`bytes`*) – asset_id that target transactions should have
- **`user_id`** (*`bytes`*) – user_id that target transactions should have

Returns the number of transactions

Return type int

`insert_transaction`(*domain_id*, *txdata*, *asset_files*)

Insert transaction into ledger

Parameters

- **`domain_id`** (*`bytes`*) – target domain_id
- **`txdata`** (*`bytes`*) – serialized transaction data
- **`asset_files`** (*`dict`*) – dictionary of {asset_id: content} for the transaction

Returns inserted transaction_id or error message

Return type dict|str

`quit_program`()

Processes when quiting program

`remove_from_notification_list`(*domain_id*, *asset_group_id*, *user_id*)

Remove entry from insert completion notification list

This method checks validation only.

Parameters

- **`domain_id`** (*`bytes`*) – target domain_id
- **`asset_group_id`** (*`bytes`*) – target asset_group_id of which you want to get notification about the insertion
- **`user_id`** (*`bytes`*) – user_id that registers in the list

`search_transaction_with_condition`(*domain_id*, *asset_group_id=None*, *asset_id=None*, *user_id=None*, *direction=0*, *count=1*)

Search transactions that match given conditions

When Multiple conditions are given, they are considered as AND condition.

Parameters

- **domain_id** (*bytes*) – target domain_id
- **asset_group_id** (*bytes*) – asset_group_id that target transactions should have
- **asset_id** (*bytes*) – asset_id that target transactions should have
- **user_id** (*bytes*) – user_id that target transactions should have
- **direction** (*int*) – 0: descend, 1: ascend
- **count** (*int*) – The maximum number of transactions to retrieve (self.search_max_count is the upper bound)

Returns dictionary having transaction_id, serialized transaction data, asset files

Return type dict

send_inserted_notification(*domain_id*, *asset_group_ids*, *transaction_id*, *only_registered_user=False*)
Broadcast NOTIFY_INSERTED

Parameters

- **domain_id** (*bytes*) – target domain_id
- **asset_group_ids** (*list*) – list of asset_group_ids
- **transaction_id** (*bytes*) – transaction_id that has just inserted
- **only_registered_user** (*bool*) – If True, notification is not sent to other nodes

validate_transaction(*txdata*, *asset_files=None*)
Validate transaction by verifying signature

Parameters

- **txdata** (*bytes*) – serialized transaction data
- **asset_files** (*dict*) – dictionary of {asset_id: content} for the transaction

Returns if validation fails, None returns.

Return type *BBcTransaction*

bbc1.core.bbc_core.**activate_ledgersubsystem**()
Load module of ledger_subsystem if installed

bbc1.core.bbc_core.**daemonize**(*pidfile='/tmp/bbc1.pid'*)
Run in background

bbc1.core.bbc_error module

bbc1.core.bbc_network module

class bbc1.core.bbc_network.**BBcNetwork**(*config*, *core=None*, *p2p_port=None*, *external_ip4addr=None*, *external_ip6addr=None*, *loglevel='all'*, *logname=None*)

Bases: `object`

Socket and thread management for infrastructure layers

CONFIRM_KEY_EXCHANGE = b'\x00\x03'

NOTIFY_LEAVE = b'\x00\x00'

REQUEST_KEY_EXCHANGE = b'\x00\x01'

RESPONSE_KEY_EXCHANGE = b'\x00\x02'

add_neighbor(*domain_id*, *node_id*, *ipv4=None*, *ipv6=None*, *port=None*, *is_static=False*)

Add node in the neighbor list

Parameters

- **domain_id** (*bytes*) – target domain_id
- **node_id** (*bytes*) – target node_id
- **ipv4** (*str*) – IPv4 address of the node
- **ipv6** (*str*) – IPv6 address of the node
- **port** (*int*) – Port number that the node is waiting at
- **is_static** (*bool*) – If true, the entry is treated as static one and will be saved in config.json

Returns True if it is a new entry, None if error.

Return type bool

broadcast_message_in_network(*domain_id*, *payload_type=1*, *msg=None*)

Send message to all neighbor nodes

Parameters

- **payload_type** (*bytes*) – message format type
- **domain_id** (*bytes*) – target domain_id
- **msg** (*dict*) – message to send

Returns True if successful

Return type bool

check_admin_signature(*domain_id*, *msg*)

Check admin signature in the message

Parameters

- **`domain_id`** (*`bytes`*) – target domain_id
- **`msg`** (*`dict`*) – received message

Returns True if valid

Return type bool

`create_domain`(*domain_id=b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0*
config=None)
Create domain and register user in the domain

Parameters

- **`domain_id`** (*`bytes`*) – target domain_id to create
- **`config`** (*`dict`*) – configuration for the domain

Returns

Return type bool

`get_domain_keypair`(*domain_id*)
Get domain_keys (private key and public key)

Parameters **`domain_id`** (*`bytes`*) – target domain_id

`include_admin_info_into_message_if_needed`(*domain_id*, *msg*, *admin_info*)
Serialize admin info into one binary object and add signature

`remove_domain`(*domain_id=b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0*
Leave the domain and remove it

Parameters **`domain_id`** (*`bytes`*) – target domain_id to remove

Returns True if successful

Return type bool

`save_all_static_node_list`()
Save all static nodes in the config file

`send_domain_ping`(*domain_id*, *ipv4*, *ipv6*, *port*, *is_static=False*)
Send domain ping to the specified node

Parameters

- **`domain_id`** (*`bytes`*) – target domain_id
- **`ipv4`** (*`str`*) – IPv4 address of the node
- **`ipv6`** (*`str`*) – IPv6 address of the node
- **`port`** (*`int`*) – Port number
- **`is_static`** (*`bool`*) – If true, the entry is treated as static one and will be saved in config.json

Returns True if successful

Return type bool

`send_key_exchange_message`(*domain_id*, *node_id*, *command*, *pubkey*, *nonce*, *random_val*, *key_name*)
Send ECDH key exchange message

send_message_in_network(*nodeinfo=None*, *payload_type=1*, *domain_id=None*, *msg=None*)
Send message over a domain network

Parameters

- **nodeinfo** (NodeInfo) – NodeInfo object of the destination
- **payload_type** (*bytes*) – message format type
- **domain_id** (*bytes*) – target domain_id
- **msg** (*dict*) – message to send

Returns True if successful

Return type bool

send_message_to_a_domain0_manager(*domain_id*, *msg*)
Choose one of domain0_managers and send msg to it

Parameters

- **domain_id** (*bytes*) – target domain_id
- **msg** (*bytes*) – message to send

setup_tcp_server()
Start tcp server

setup_udp_socket()
Setup UDP socket

tcpserver_loop()
Message loop for TCP socket

udp_message_loop()
Message loop for UDP socket

class bbc1.core.bbc_network.**NeighborInfo**(*network=None*, *domain_id=None*, *node_id=None*, *my_info=None*)
Bases: object

Manage information of neighbor nodes

NODEINFO_LIFETIME = 900

PURGE_INTERVAL_SEC = 300

add(*node_id*, *ipv4=None*, *ipv6=None*, *port=None*, *is_static=False*, *domain0=None*)
Add or update an neighbor node entry

purge(*query_entry*)
Purge obsoleted entry in nodeinfo_list

remove(*node_id*)
Remove entry in the nodeinfo_list

show_list()
Return nodeinfo list in human readable format

class bbc1.core.bbc_network.**NodeInfo**(*node_id=None*, *ipv4=None*, *ipv6=None*, *port=None*, *is_static=False*, *domain0=False*)
Bases: object

Node information entry

SECURITY_STATE_CONFIRMING = 2

SECURITY_STATE_ESTABLISHED = 3

SECURITY_STATE_NONE = 0

SECURITY_STATE_REQUESTING = 1

get_nodeinfo()
Return a list of node info

Returns [node_id, ipv4, ipv6, port, domain0_flag, update_at]

Return type list

touch()

update(*ipv4=None*, *ipv6=None*, *port=None*, *seq=None*, *domain0=None*)
Update the entry

Parameters

- **ipv4** (*str*) – IPv4 address of the sender node
- **ipv6** (*str*) – IPv6 address of the sender node
- **port** (*int*) – Port number of the sender
- **sec** (*int*) – message sequence number
- **domain0** (*bool or None*) – If True, the node is domain0 manager

Returns True if the entry has changed

Return type bool

bbc1.core.bbc_network.**is_less_than**(*val_a*, *val_b*)
Return True if val_a is less than val_b (evaluate as integer)

bbc1.core.bbc_stats module

class bbc1.core.bbc_stats.**BBcStats**
Bases: object

clear_stats()

get_stats()

remove_stat_category(*category*)

remove_stat_item(*category*, *name*)

update_stats(*category*, *name*, *value*)

update_stats_decrement(*category*, *name*, *value*)

update_stats_increment(*category*, *name*, *value*)

bbc1.core.bbclib module

---

class `bbc1.core.bbclib.`**`BBcAsset`**(*user_id=None*, *asset_file=None*, *asset_body=None*, *format_type=0*, *id_length=32*)

Bases: `object`

Asset part in a transaction

`add`(*user_id=None*, *asset_file=None*, *asset_body=None*)

Add parts in this object

`deserialize`(*data*)

Deserialize into this object

Parameters **`data`** (*`bytes`*) – serialized binary data

Returns True if successful

Return type bool

`deserialize_obj`(*obj*)

Deserialize bson/msgpack data into this object

Parameters **`obj`** (*`bytes`*) – object data

Returns True if successful

Return type bool

`digest`()

Calculate the digest

The digest corresponds to the asset_id of this object

Returns asset_id (or digest)

Return type bytes

`get_asset_file`()

Get asset file content and its digest

Returns digest of the file content bytes: the file content

Return type bytes

`get_dict`(*for_digest_calculation=False*)

Serialize this object

`recover_asset_file`(*asset_file*, *id_length=32*)

Recover asset file info from the given raw content

`serialize`(*for_digest_calculation=False*)

Serialize this object

Parameters **`for_digest_calculation`** (*`bool`*) – True if digest calculation

Returns serialized binary data

Return type bytes

class `bbc1.core.bbclib.`**`BBcCrossRef`**(*domain_id=None*, *transaction_id=None*, *deserialize=None*, *format_type=0*)

Bases: `object`

CrossRef part in a transaction

`deserialize`(*data*)

Deserialize into this object

Parameters **`data`** (*`bytes`*) – serialized binary data

Returns True if successful

Return type bool

`deserialize_obj`(*obj*)

Deserialize bson/msgpack data into this object

Parameters **`obj`** (*`bytes`*) – object data

Returns True if successful

Return type bool

`get_dict`()

Serialize this object into bson format

`serialize`()

Serialize this object

Returns serialized binary data

Return type bytes

class `bbc1.core.bbclib.`**`BBcEvent`**(*asset_group_id=None*, *format_type=0*, *id_length=32*)

Bases: `object`

Event part in a transaction

`add`(*asset_group_id=None*, *reference_index=None*, *mandatory_approver=None*, *option_approver_num_numerator=0*, *option_approver_num_denominator=0*, *option_approver=None*, *asset=None*)

Add parts

`deserialize`(*data*)

Deserialize into this object

Parameters **`data`** (*`bytes`*) – serialized binary data

Returns True if successful

Return type bool

`deserialize_obj`(*obj*)

Deserialize bson/msgpack data into this object

Parameters **`obj`** (*`bytes`*) – object data

Returns True if successful

Return type bool

get_dict()
Serialize this object

serialize()
Serialize this object

Returns serialized binary data

Return type bytes

class bbc1.core.bbclib.**BBcFormat**
Bases: `object`

FORMAT_BINARY = 0

FORMAT_BSON = 1

FORMAT_BSON_COMPRESS_BZ2 = 2

FORMAT_BSON_COMPRESS_ZLIB = 3

FORMAT_MSGPACK = 4

FORMAT_MSGPACK_COMPRESS_BZ2 = 5

FORMAT_MSGPACK_COMPRESS_ZLIB = 6

class bbc1.core.bbclib.**BBcPointer**(*transaction_id=None*, *asset_id=None*, *format_type=0*, *id_length=32*)
Bases: `object`

Pointer part in a transaction

add(*transaction_id=None*, *asset_id=None*)
Add parts

deserialize(*data*)
Deserialize into this object

Parameters **data** (*bytes*) – serialized binary data

Returns True if successful

Return type bool

deserialize_obj(*obj*)
Deserialize bson/msgpack data into this object

Parameters **obj** (*bytes*) – object data

Returns True if successful

Return type bool

get_dict()
Serialize this object

serialize()
Serialize this object

Returns serialized binary data

Return type bytes

class bbc1.core.bbclib.**BBcReference**(*asset_group_id*, *transaction*, *ref_transaction=None*, *event_index_in_ref=0*, *format_type=0*, *id_length=32*)
Bases: `object`

Reference part in a transaction

add_signature(*user_id=None*, *signature=None*)
Add signature in the reserved space

Parameters

- **user_id** (*bytes*) – user_id of the signature owner
- **signature** (BBcSignature) – signature

deserialize(*data*)
Deserialize into this object

Parameters **data** (*bytes*) – serialized binary data

Returns True if successful

Return type bool

deserialize_obj(*obj*)
Deserialize bson/msgpack data into this object

Parameters **obj** (*bytes*) – object data

Returns True if successful

Return type bool

get_destinations()
Return the list of approvers in the referred transaction

get_dict()
Serialize this object

get_referred_transaction()
Return referred transaction in serialized format

prepare_reference(*ref_transaction*)
Read the previous referencing transaction

serialize()
Serialize this object

Returns serialized binary data

Return type bytes

class bbc1.core.bbclib.**BBcRelation**(*asset_group_id=None*, *format_type=0*, *id_length=32*)
Bases: object

Relation part in a transaction

add(*asset_group_id=None*, *asset=None*, *pointer=None*)
Add parts

deserialize(*data*)
Deserialize bson data into this object

Parameters **data** (*dict*) – bson data

Returns True if successful

Return type bool

deserialize_obj(*obj*)
Deserialize bson/msgpack data into this object

Parameters **`data`** (*`bytes`*) – object data

Returns True if successful

Return type bool

`get_dict`()
Serialize this object

`serialize`()
Serialize this object

Returns serialized binary data

Return type bytes

class `bbc1.core.bbclib.`**`BBcSignature`**(*key_type=2*, *deserialize=None*, *format_type=0*)
Bases: `object`

Signature part in a transaction

`add`(*signature=None*, *pubkey=None*)
Add signature and public key

`deserialize`(*data*)
Deserialize into this object

Parameters **`data`** (*`bytes`*) – serialized binary data

Returns True if successful

Return type bool

`deserialize_obj`(*obj*)
Deserialize bson/msgpack data into this object

Parameters **`obj`** (*`bytes`*) – object data

Returns True if successful

Return type bool

`get_dict`()
Serialize this object

`serialize`()
Serialize this object

`verify`(*digest*)
Verify digest using pubkey in signature

Parameters **`digest`** (*`bytes`*) – digest to verify

Returns 0:invalid, 1:valid

Return type int

class `bbc1.core.bbclib.`**`BBcTransaction`**(*version=1*, *deserialize=None*, *format_type=0*, *id_length=32*)
Bases: `object`

Transaction object

`add`(*event=None*, *reference=None*, *relation=None*, *witness=None*, *cross_ref=None*)
Add parts

`add_signature`(*user_id=None*, *signature=None*)
Add signature in the reserved space

Parameters

- **`user_id`** (*`bytes`*) – user_id of the signature owner
- **`signature`** (`BBcSignature`) – signature

Returns True if successful

Return type bool

`deserialize`(*data*)
Deserialize into this object

Parameters **`data`** (*`bytes`*) – serialized binary data

Returns True if successful

Return type bool

`deserialize_obj`(*data*)
Deserialize bson/msgpack data into this object

Parameters **`data`** (*`bytes`*) – object data

Returns True if successful

Return type bool

`digest`()
Calculate the digest

The digest corresponds to the transaction_id of this object

Returns transaction_id (or digest)

Return type bytes

`get_sig_index`(*user_id*)
Reserve a space for signature for the specified user_id

Parameters **`user_id`** (*`bytes`*) – user_id whose signature will be added to the signature part

Returns position (index) in the signature part

Return type int

`serialize`(*for_id=False*)
Serialize the whole parts

`serialize_obj`(*for_id=False*, *no_header=False*)
Serialize the whole parts

`set_format_type`(*format_type*)

`sign`(*key_type=2*, *private_key=None*, *public_key=None*, *keypair=None*)
Sign the transaction

Parameters

- **`key_type`** (*`int`*) – Type of encryption key's curve
- **`private_key`** (*`bytes`*) –
- **`public_key`** (*`bytes`*) –
- **`keypair`** (`KeyPair`) – keypair or set of private_key and public_key needs to be given

Returns

Return type *BBcSignature*

class bbc1.core.bbclib.**BBcWitness**(*format_type=0*, *id_length=32*)

Bases: `object`

Witness part in a transaction

add_signature(*user_id=None*, *signature=None*)

Add signature in the reserved space for the user_id that was registered before

Parameters

- **user_id** (*bytes*) – user_id of the signature owner
- **signature** (*bytes*) – signature

add_witness(*user_id*)

Register user_id in the list

deserialize(*data*)

Deserialize into this object

Parameters **data** (*bytes*) – serialized binary data

Returns True if successful

Return type bool

deserialize_obj(*obj*)

Deserialize bson/msgpack data into this object

Parameters **obj** (*bytes*) – object data

Returns True if successful

Return type bool

get_dict()

Serialize this object

serialize()

Serialize this object

Returns serialized binary data

Return type bytes

class bbc1.core.bbclib.**KeyPair**(*curvetype=2*, *compression=False*, *privkey=None*, *pubkey=None*)

Bases: `object`

POINT_CONVERSION_COMPRESSED = 2

POINT_CONVERSION_UNCOMPRESSED = 4

Key pair container

generate()

Generate a new key pair

get_private_key_in_der()

Return private key in DER format

get_private_key_in_pem()

Return private key in PEM format

get_public_key_in_pem()

Return public key in PEM format

import_publickey_cert_pem(*cert_pemstring*, *privkey_pemstring=None*)
Verify and import X509 public key certificate in pem format

mk_keyobj_from_private_key()
Make a keypair object from the binary data of private key

mk_keyobj_from_private_key_der(*derdat*)
Make a keypair object from the private key in DER format

mk_keyobj_from_private_key_pem(*pemdat_string*)
Make a keypair object from the private key in PEM format

sign(*digest*)
Sign to the given value

Parameters **digest** (`bytes`) – given value

Returns signature

Return type bytes

to_binary(*dat*)

verify(*digest*, *sig*)
Verify the digest and the signature using the rivate key in this object

class bbc1.core.bbclib.**KeyType**
Bases: `object`

ECDSA_P256v1 = 2

ECDSA_SECP256k1 = 1

NOT_INITIALIZED = 0

class bbc1.core.bbclib.**MsgType**
Bases: `object`

Message types for between core node and client

CANCEL_INSERT_NOTIFICATION = 16

DOMAIN_PING = 12

MESSAGE = 66

NOTIFY_CROSS_REF = 74

NOTIFY_DOMAIN_KEY_UPDATE = 19

NOTIFY_INSERTED = 73

REGISTER = 64

REQUEST_CLOSE_DOMAIN = 31

REQUEST_COUNT_TRANSACTIONS = 95

REQUEST_CROSS_REF_LIST = 92

REQUEST_CROSS_REF_VERIFY = 90

REQUEST_ECDH_KEY_EXCHANGE = 33

REQUEST_GATHER_SIGNATURE = 67

REQUEST_GET_CONFIG = 8

REQUEST_GET_DOMAINLIST = 13

`REQUEST_GET_FORWARDING_LIST = 25`

`REQUEST_GET_NEIGHBORLIST = 21`

`REQUEST_GET_NODEID = 27`

`REQUEST_GET_NOTIFICATION_LIST = 29`

`REQUEST_GET_STATS = 17`

`REQUEST_GET_USERS = 23`

`REQUEST_INSERT = 71`

`REQUEST_INSERT_NOTIFICATION = 15`

`REQUEST_MANIP_LEDGER_SUBSYS = 10`

`REQUEST_REGISTER_HASH_IN_SUBSYS = 128`

`REQUEST_REPAIR = 94`

`REQUEST_SEARCH_TRANSACTION = 82`

`REQUEST_SEARCH_WITH_CONDITIONS = 86`

`REQUEST_SETUP_DOMAIN = 0`

`REQUEST_SET_STATIC_NODE = 4`

`REQUEST_SIGNATURE = 69`

`REQUEST_TRAVERSE_TRANSACTIONS = 88`

`REQUEST_VERIFY_HASH_IN_SUBSYS = 130`

`RESPONSE_CLOSE_DOMAIN = 32`

`RESPONSE_COUNT_TRANSACTIONS = 95`

`RESPONSE_CROSS_REF_LIST = 93`

`RESPONSE_CROSS_REF_VERIFY = 91`

`RESPONSE_ECDH_KEY_EXCHANGE = 34`

`RESPONSE_GATHER_SIGNATURE = 68`

`RESPONSE_GET_CONFIG = 9`

`RESPONSE_GET_DOMAINLIST = 14`

`RESPONSE_GET_FORWARDING_LIST = 26`

`RESPONSE_GET_NEIGHBORLIST = 22`

`RESPONSE_GET_NODEID = 28`

`RESPONSE_GET_NOTIFICATION_LIST = 30`

`RESPONSE_GET_STATS = 18`

`RESPONSE_GET_USERS = 24`

`RESPONSE_INSERT = 72`

`RESPONSE_MANIP_LEDGER_SUBSYS = 11`

`RESPONSE_REGISTER_HASH_IN_SUBSYS = 129`

`RESPONSE_SEARCH_TRANSACTION = 83`

RESPONSE_SEARCH_WITH_CONDITIONS = 87

RESPONSE_SETUP_DOMAIN = 1

RESPONSE_SET_STATIC_NODE = 5

RESPONSE_SIGNATURE = 70

RESPONSE_TRAVERSE_TRANSACTIONS = 89

RESPONSE_VERIFY_HASH_IN_SUBSYS = 131

UNREGISTER = 65

`bbc1.core.bbclib.`**`add_event_asset`**(*transaction*, *event_idx*, *asset_group_id*, *user_id*, *asset_body=None*, *asset_file=None*)
Utility to add BBcEvent object with BBcAsset in the transaction

`bbc1.core.bbclib.`**`add_pointer_in_relation`**(*relation*, *ref_transaction_id=None*, *ref_asset_id=None*)
Utility to add BBcRelation object with BBcPointer in the BBcRelation object

`bbc1.core.bbclib.`**`add_reference_to_transaction`**(*transaction*, *asset_group_id*, *ref_transaction_obj*, *event_index_in_ref*)
Utility to add BBcReference object in the transaction

Returns

Return type *BBcReference*

`bbc1.core.bbclib.`**`add_relation_asset`**(*transaction*, *relation_idx*, *asset_group_id*, *user_id*, *asset_body=None*, *asset_file=None*)
Utility to add BBcRelation object with BBcAsset in the transaction

`bbc1.core.bbclib.`**`add_relation_pointer`**(*transaction*, *relation_idx*, *ref_transaction_id=None*, *ref_asset_id=None*)
Utility to add BBcRelation object with BBcPointer in the transaction

`bbc1.core.bbclib.`**`bin2str_base64`**(*dat*)

`bbc1.core.bbclib.`**`convert_id_to_string`**(*data*, *bytelen=32*)
Convert binary data to hex string

`bbc1.core.bbclib.`**`convert_idstring_to_bytes`**(*datastr*, *bytelen=32*)
Convert hex string to binary data

`bbc1.core.bbclib.`**`deep_copy_with_key_stringify`**(*u*, *d=None*)
Utility for updating nested dictionary

`bbc1.core.bbclib.`**`get_bigint`**(*ptr*, *dat*)

`bbc1.core.bbclib.`**`get_n_byte_int`**(*ptr*, *n*, *dat*)

`bbc1.core.bbclib.`**`get_n_bytes`**(*ptr*, *n*, *dat*)

`bbc1.core.bbclib.`**`get_new_id`**(*seed_str=None*, *include_timestamp=True*)
Return 256-bit binary data

Parameters

- **seed_str** (`str`) – seed string that is hashed by SHA256
- **include_timestamp** (`bool`) – if True, timestamp (current time) is appended to the seed string

Returns 256-bit binary

Return type bytes

`bbc1.core.bbclib.`**`get_random_id`**`()`

Return 256-bit binary data

Returns 256-bit random binary

Return type bytes

`bbc1.core.bbclib.`**`get_random_value`**`(`*length=32*`)`

Return 1-byte random value

`bbc1.core.bbclib.`**`make_relation_with_asset`**`(`*asset_group_id*, *user_id*, *asset_body=None*, *asset_file=None*, *format_type=0*, *id_length=32*`)`

Utility to make BBcRelation object

`bbc1.core.bbclib.`**`make_transaction`**`(`*event_num=0*, *relation_num=0*, *witness=False*, *format_type=0*, *id_length=32*`)`

Utility to make transaction object

Parameters

- **`event_num`** (*`int`*) – the number of BBcEvent object to include in the transaction
- **`relation_num`** (*`int`*) – the number of BBcRelation object to include in the transaction
- **`witness`** (*`bool`*) – If true, BBcWitness object is included in the transaction
- **`format_type`** (*`int`*) – Data format defined in BBcFormat class
- **`id_length`** (*`int`*) – If <32, IDs will be truncated

Returns

Return type *BBcTransaction*

`bbc1.core.bbclib.`**`recover_signature_object`**`(`*data*, *format_type=0*`)`

Deserialize signature data

`bbc1.core.bbclib.`**`reset_error`**`()`

`bbc1.core.bbclib.`**`set_error`**`(`*code=-1*, *txt=”*`)`

`bbc1.core.bbclib.`**`str_binary`**`(`*dat*`)`

`bbc1.core.bbclib.`**`to_1byte`**`(`*val*`)`

`bbc1.core.bbclib.`**`to_2byte`**`(`*val*`)`

`bbc1.core.bbclib.`**`to_4byte`**`(`*val*`)`

`bbc1.core.bbclib.`**`to_8byte`**`(`*val*`)`

`bbc1.core.bbclib.`**`to_bigint`**`(`*val*, *size=32*`)`

`bbc1.core.bbclib.`**`validate_transaction_object`**`(`*txobj*, *asset_files=None*`)`

Validate transaction and its asset

Parameters

- **`txobj`** (`BBcTransaction`) – target transaction object
- **`asset_files`** (*`dict`*) – dictionary containing the asset file contents

Returns True if valid tuple: list of valid assets tuple: list of invalid assets

Return type bool

`bbc1.core.bbclib.`**`verify_using_cross_ref`**(*domain_id*, *transaction_id*, *transaction_base_digest*, *cross_ref_data*, *sigdata*, *format_type=0*)

Confirm the existence of the transaction using cross_ref

Parameters

- **`domain_id`** (*bytes*) – target domain_id
- **`transaction_id`** (*bytes*) – target transaction_id of which existence you want to confirm
- **`transaction_base_digest`** (*bytes*) – digest obtained from the outer domain
- **`cross_ref_data`** (*bytes*) – serialized BBcCrossRef object
- **`sigdata`** (*bytes*) – serialized signature
- **`format_type`** (*int*) – Data format type when calculating the digest (transaction_id)

Returns True if valid

Return type bool

bbc1.core.command module

`bbc1.core.command.`**`parser`**()

bbc1.core.data_handler module

`class` `bbc1.core.data_handler.`**`DataHandler`**(*networking=None*, *config=None*, *workingdir=None*, *domain_id=None*, *loglevel='all'*, *logname=None*)

Bases: `object`

DB and storage handler

`NOTIFY_INSERTED = b'\x00\x04'`

`REPAIR_TRANSACTION_DATA = b'\x00\x05'`

REPLICATION_ALL = 0

REPLICATION_CROSS_REF = b'\x00\x06'

REPLICATION_EXT = 2

REPLICATION_P2P = 1

REQUEST_REPLICATION_INSERT = b'\x00\x00'

REQUEST_SEARCH = b'\x00\x02'

RESPONSE_REPLICATION_INSERT = b'\x00\x01'

RESPONSE_SEARCH = b'\x00\x03'

count_domain_in_cross_ref(*outer_domain_id*)
Count the number of domains in the cross_ref table

count_transactions(*asset_group_id=None*, *asset_id=None*, *user_id=None*, *db_num=0*)
Count transactions that matches the given conditions

When Multiple conditions are given, they are considered as AND condition.

Parameters

- **asset_group_id** (*bytes*) – asset_group_id that target transactions should have
- **asset_id** (*bytes*) – asset_id that target transactions should have
- **user_id** (*bytes*) – user_id that target transactions should have
- **db_num** (*int*) – index of DB if multiple DBs are used

Returns the number of transactions

Return type int

exec_sql(*db_num=0*, *sql=None*, *args=()*, *commit=False*, *fetch_one=False*)
Execute sql sentence

Parameters

- **db_num** (*int*) – index of DB if multiple DBs are used
- **sql** (*str*) – SQL string
- **args** (*list*) – Args for the SQL
- **commit** (*bool*) – If True, commit is performed
- **fetch_one** (*bool*) – If True, fetch just one record

Returns list of records

Return type list

get_asset_info(*txobj*)
Retrieve asset information from transaction object

Parameters **txobj** (BBcTransaction) – transaction object to analyze

Returns list of list [asset_group_id, asset_id, user_id, file_size, file_digest]

Return type list

get_in_storage(*asset_group_id*, *asset_id*)
Get the asset file with the asset_id from local storage

Parameters

- **`asset_group_id`** (*`bytes`*) – asset_group_id of the asset
- **`asset_id`** (*`bytes`*) – asset_id of the asset

Returns the file content

Return type bytes or None

`insert_cross_ref`(*transaction_id*, *outer_domain_id*, *txid_having_cross_ref*, *no_replication=False*)

Insert cross_ref information into cross_ref_table

Parameters

- **`transaction_id`** (*`bytes`*) – target transaction_id
- **`outer_domain_id`** (*`bytes`*) – domain_id that holds cross_ref about the transaction_id
- **`txid_having_cross_ref`** (*`bytes`*) – transaction_id in the outer_domain that includes the cross_ref
- **`no_replication`** (*`bool`*) – If False, the replication is sent to other nodes in the domain

`insert_transaction`(*txdata*, *txobj=None*, *asset_files=None*, *no_replication=False*)

Insert transaction data and its asset files

Either txdata or txobj must be given to insert the transaction.

Parameters

- **`txdata`** (*`bytes`*) – serialized transaction data
- **`txobj`** (`BBcTransaction`) – transaction object to insert
- **`asset_files`** (*`dict`*) – asset files in the transaction

Returns set of asset_group_ids in the transaction

Return type set

`process_message`(*msg*)

Process received message

Parameters **`msg`** (*`dict`*) – received message

`remove`(*transaction_id*, *txobj=None*, *db_num=-1*)

Delete all data regarding the specified transaction_id

This method requires either transaction_id or txobj.

Parameters

- **`transaction_id`** (*`bytes`*) – target transaction_id
- **`txobj`** (`BBcTransaction`) – transaction object to remove
- **`db_num`** (*`int`*) – index of DB if multiple DBs are used

`restore_transaction_data`(*db_num*, *transaction_id*, *txobj*)

Remove and insert a transaction

`search_domain_having_cross_ref`(*transaction_id=None*)

Search domain_id that holds cross_ref about the specified transaction_id

Parameters **`transaction_id`** (*`bytes`*) – target transaction_id

Returns records of cross_ref_tables [“id”,”transaction_id”, “outer_domain_id”, “txid_having_cross_ref”]

Return type list

search_transaction(*transaction_id=None*, *asset_group_id=None*, *asset_id=None*, *user_id=None*, *direction=0*, *count=1*, *db_num=0*)
Search transaction data

When Multiple conditions are given, they are considered as AND condition.

Parameters

- **transaction_id** (*bytes*) – target transaction_id
- **asset_group_id** (*bytes*) – asset_group_id that target transactions should have
- **asset_id** (*bytes*) – asset_id that target transactions should have
- **user_id** (*bytes*) – user_id that target transactions should have
- **direction** (*int*) – 0: descend, 1: ascend
- **count** (*int*) – The maximum number of transactions to retrieve
- **db_num** (*int*) – index of DB if multiple DBs are used

Returns mapping from transaction_id to serialized transaction data dict: dictionary of {asset_id: content} for the transaction

Return type dict

search_transaction_topology(*transaction_id*, *traverse_to_past=True*)
Search in topology info

Parameters

- **transaction_id** (*bytes*) – base transaction_id
- **traverse_to_past** (*bool*) – True: search backward (to past), False: search forward (to future)

Returns list of records of topology table

Return type list

store_in_storage(*asset_group_id*, *asset_id*, *content*, *do_overwrite=False*)
Store asset file in local storage

Parameters

- **asset_group_id** (*bytes*) – asset_group_id of the asset
- **asset_id** (*bytes*) – asset_id of the asset
- **content** (*bytes*) – the content of the asset file
- **do_overwrite** (*bool*) – If True, file is overwritten

Returns True if successful

Return type bool

class bbc1.core.data_handler.**DataHandlerDomain0**(*networking=None*, *config=None*, *workingdir=None*, *domain_id=None*, *loglevel='all'*, *logname=None*)

Bases: *bbc1.core.data_handler.DataHandler*

Data handler for domain_global_0

exec_sql(*sql*, **args*)
Execute sql sentence

Parameters

- **db_num** (*int*) – index of DB if multiple DBs are used
- **sql** (*str*) – SQL string
- **args** (*list*) – Args for the SQL
- **commit** (*bool*) – If True, commit is performed
- **fetch_one** (*bool*) – If True, fetch just one record

Returns list of records

Return type list

get_asset_info(*txobj*)

Retrieve asset information from transaction object

Parameters **txobj** (BBcTransaction) – transaction object to analyze

Returns list of list [asset_group_id, asset_id, user_id, file_size, file_digest]

Return type list

get_in_storage(*asset_group_id*, *asset_id*)

Get the asset file with the asset_id from local storage

Parameters

- **asset_group_id** (*bytes*) – asset_group_id of the asset
- **asset_id** (*bytes*) – asset_id of the asset

Returns the file content

Return type bytes or None

insert_transaction(*txdata*, *txobj=None*, *asset_files=None*, *no_replication=False*)

Insert transaction data and its asset files

Either txdata or txobj must be given to insert the transaction.

Parameters

- **txdata** (*bytes*) – serialized transaction data
- **txobj** (BBcTransaction) – transaction object to insert
- **asset_files** (*dict*) – asset files in the transaction

Returns set of asset_group_ids in the transaction

Return type set

process_message(*msg*)

Process received message

Parameters **msg** (*dict*) – received message

remove(*transaction_id*)

Delete all data regarding the specified transaction_id

This method requires either transaction_id or txobj.

Parameters

- **transaction_id** (*bytes*) – target transaction_id
- **txobj** (BBcTransaction) – transaction object to remove

- **db_num** (*int*) – index of DB if multiple DBs are used

search_transaction(*transaction_id=None*, *asset_group_id=None*, *asset_id=None*, *user_id=None*, *count=1*)

Search transaction data

When Multiple conditions are given, they are considered as AND condition.

Parameters

- **transaction_id** (*bytes*) – target transaction_id
- **asset_group_id** (*bytes*) – asset_group_id that target transactions should have
- **asset_id** (*bytes*) – asset_id that target transactions should have
- **user_id** (*bytes*) – user_id that target transactions should have
- **direction** (*int*) – 0: descend, 1: ascend
- **count** (*int*) – The maximum number of transactions to retrieve
- **db_num** (*int*) – index of DB if multiple DBs are used

Returns mapping from transaction_id to serialized transaction data dict: dictionary of {asset_id: content} for the transaction

Return type dict

search_transaction_topology(*transaction_id*, *reverse_link=False*)

Search in topology info

Parameters

- **transaction_id** (*bytes*) – base transaction_id
- **traverse_to_past** (*bool*) – True: search backward (to past), False: search forward (to future)

Returns list of records of topology table

Return type list

store_in_storage(*asset_group_id*, *asset_id*, *content*)

Store asset file in local storage

Parameters

- **asset_group_id** (*bytes*) – asset_group_id of the asset
- **asset_id** (*bytes*) – asset_id of the asset
- **content** (*bytes*) – the content of the asset file
- **do_overwrite** (*bool*) – If True, file is overwritten

Returns True if successful

Return type bool

class bbc1.core.data_handler.**DbAdaptor**(*handler=None*, *db_name=None*, *db_num=0*, *loglevel='all'*, *logname=None*)

Bases: object

Base class for DB adaptor

check_table_existence(*tblname*)

Check whether the table exists or not

create_table (*tbl*, *tbl_definition*, *primary_key=0*, *indices=[]*)
Create a table

open_db ()
Open the DB

class bbc1.core.data_handler.**MysqlAdaptor** (*handler=None*, *db_name=None*, *db_num=None*, *server_info=None*, *loglevel='all'*, *logname=None*)
Bases: *bbc1.core.data_handler.DbAdaptor*

DB adaptor for MySQL

check_table_existence (*tblname*)
Check whether the table exists or not

create_table (*tbl*, *tbl_definition*, *primary_key=0*, *indices=[]*)
Create a table

Parameters

- **tbl** (*str*) – table name
- **tbl_definition** (*list*) – schema of the table [["column_name", "data type"],["colmun_name", "data type"],,]
- **primary_key** (*int*) – index (column) of the primary key of the table
- **indices** (*list*) – list of indices to create index

open_db ()
Open the DB

class bbc1.core.data_handler.**SqliteAdaptor** (*handler=None*, *db_name=None*, *loglevel='all'*, *logname=None*)
Bases: *bbc1.core.data_handler.DbAdaptor*

DB adaptor for SQLite3

check_table_existence (*tblname*)
Check whether the table exists or not

create_table (*tbl*, *tbl_definition*, *primary_key=0*, *indices=[]*)
Create a table

Parameters

- **tbl** (*str*) – table name
- **tbl_definition** (*list*) – schema of the table [["column_name", "data type"],["colmun_name", "data type"],,]
- **primary_key** (*int*) – index (column) of the primary key of the table
- **indices** (*list*) – list of indices to create index

open_db ()
Open the DB (create DB file if not exists)

bbc1.core.domain0_manager module

Copyright (c) 2017 beyond-blockchain.org.

class `bbc1.core.domain0_manager.`**`Domain0Manager`**(*networking=None*, *node_id=None*, *loglevel='all'*, *logname=None*)

Bases: `object`

Management for inter-domain collaboration over domain_global_0

`ADV_DOMAIN_LIST = b'\x00\x00'`

`CROSS_REF_PROBABILITY = 0.1`

`DISTRIBUTE_CROSS_REF = b'\x00\x01'`

`DOMAIN_ACCEPTANCE_RECOVER_INTERVAL = 600`

`DOMAIN_INFO_ADVERTISE_INTERVAL = 1800`

`DOMAIN_INFO_LIFETIME = 3600`

`INITIAL_ACCEPT_LIMIT = 10`

`NOTIFY_CROSS_REF_REGISTERED = b'\x00\x02'`

`NUM_OF_COPIES = 3`

`REQUEST_VERIFY = b'\x00\x04'`

`REQUEST_VERIFY_FROM_OUTER_DOMAIN = b'\x00\x05'`

`RESPONSE_VERIFY_FROM_OUTER_DOMAIN = b'\x00\x06'`

`cross_ref_registered`(*domain_id*, *transaction_id*, *cross_ref*)

Notify cross_ref inclusion in a transaction of the outer domain and insert the info into DB

Parameters

- **`domain_id`** (*`bytes`*) – domain_id where the cross_ref is from
- **`transaction_id`** (*`bytes`*) – transaction_id that the cross_ref proves
- **`cross_ref`** (*`bytes`*) – the registered cross_ref in other domain

`distribute_cross_ref_in_domain0`(*domain_id*, *transaction_id*)

Determine if the node distributes the cross_ref (into domain_global_0)

Parameters

- **`domain_id`** (*`bytes`*) – target domain_id
- **`transaction_id`** (*`bytes`*) – target transaction_id

`process_message`(*msg*)

Process received message

Parameters **`msg`** (*`dict`*) – received message

`stop_all_timers`()

Invalidate all running timers

update_domain_belong_to()
Update the list domain_belong_to

domain_belong_to holds all domain_ids that this node belongs to

bbc1.core.key_exchange_manager module

class bbc1.core.key_exchange_manager.**KeyExchangeManager**(*networking*, *domain_id*, *counter_node_id*)
Bases: `object`

ECDH (Elliptic Curve Diffie-Hellman) key exchange manager

KEY_EXCHANGE_INVOKE_MAX_BACKOFF = 6

KEY_EXCHANGE_RETRY_INTERVAL = 5

KEY_OBSOLETE_TIMER = 10

KEY_REFRESH_INTERVAL = 604800

STATE_CONFIRMING = 2

STATE_ESTABLISHED = 3

STATE_NONE = 0

STATE_REQUESTING = 1

receive_confirmation()
Confirm that the key has been agreed

receive_exchange_request(*pubkey*, *nonce*, *random_val*, *hint*)
Procedure when receiving message with BBcNetwork.REQUEST_KEY_EXCHANGE

Parameters

- **pubkey** (*bytes*) – public key
- **nonce** (*bytes*) – nonce value
- **random_val** (*bytes*) – random value in calculating key

receive_exchange_response(*pubkey*, *random_val*, *hint*)
Process ECDH procedure (receiving response)

set_cipher(*key_name*, *hint*)
Set key to the encryptor and decryptor

set_invoke_timer(*timeout*, *retry_entry=False*)
Set timer for key refreshment

stop_all_timers()
Stop all timers

unset_cipher(*key_name=None*)
Unset key from the encryptor and decryptor

`bbc1.core.key_exchange_manager.`**remove_old_key**(*query_entry*)

bbc1.core.logger module

`bbc1.core.logger.`**get_logger**(*key="*, *logname='-'*, *level='none'*)

bbc1.core.message_key_types module

class `bbc1.core.message_key_types.`**InfraMessageCategory**
Bases: `object`

Types of message for inter-core nodes messaging

CATEGORY_DATA = b'\x00\x03'

CATEGORY_DOMAIN0 = b'\x00\x04'

CATEGORY_NETWORK = b'\x00\x00'

CATEGORY_TOPOLOGY = b'\x00\x01'

CATEGORY_USER = b'\x00\x02'

class `bbc1.core.message_key_types.`**KeyType**
Bases: `object`

Types of items in a message

admin_info = b'\x00\x00\x00\x17'

all_asset_files = b'\x00\x00\x00u'

all_included = b'\x00\x00\x00h'

anycast_ttl = b'\x00\x00\x00\x1a'

asset_file = b'\x00\x00\x00t'

```
asset_group_id = b'\x00\x00\x00c'
asset_group_ids = b'\x00\x00\x00d'
asset_id = b'\x00\x00\x00e'
bbc_configuration = b'\x00\x00\x00<'
command = b'\x00\x00\x00\t'
compromised_asset_files = b'\x00\x00\x00\x92'
compromised_transaction_data = b'\x00\x00\x00\x90'
compromised_transaction_ids = b'\x00\x00\x00\x93'
compromised_transactions = b'\x00\x00\x00\x91'
count = b'\x00\x00\x00\x0e'
cross_ref = b'\x00\x00\x00w'
cross_ref_verification_info = b'\x00\x00\x00{'
destination_node_id = b'\x00\x00\x00V'
destination_user_id = b'\x00\x00\x00R'
destination_user_ids = b'\x00\x00\x00S'
direction = b'\x00\x00\x00f'
domain_id = b'\x00\x00\x00P'
domain_list = b'\x00\x00\x007'
domain_ping = b'\x00\x00\x00\x15'
ecdh = b'\x00\x00\x00\x11'
external_ip4addr = b'\x00\x00\x004'
external_ip6addr = b'\x00\x00\x005'
forwarding_list = b'\x00\x00\x008'
hint = b'\x00\x00\x00\x10'
hop_count = b'\x00\x00\x00g'
infra_command = b'\x00\x00\x00\n'
infra_msg_type = b'\x00\x00\x00\x08'
ipv4_address = b'\x00\x00\x001'
ipv6_address = b'\x00\x00\x002'
is_anycast = b'\x00\x00\x00\x19'
is_replication = b'\x00\x00\x00\x1b'
ledger_subsys_manip = b'\x00\x00\x00\xa0'
ledger_subsys_register = b'\x00\x00\x00\xa1'
ledger_subsys_verify = b'\x00\x00\x00\xa2'
merkle_tree = b'\x00\x00\x00\xa3'
message = b'\x00\x00\x00\x0c'
```

message_seq = b'\x00\x00\x00\x14'

neighbor_list = b'\x00\x00\x00:'

node_id = b'\x00\x00\x00T'

node_info = b'\x00\x00\x006'

nodekey_signature = b'\x00\x00\x00\x16'

nonce = b'\x00\x00\x00\r'

notification_list = b'\x00\x00\x00;'

on_multinodes = b'\x00\x00\x00\x18'

outer_domain_id = b'\x00\x00\x00x'

port_number = b'\x00\x00\x003'

query_id = b'\x00\x00\x00\x0b'

random = b'\x00\x00\x00\x12'

reason = b'\x00\x00\x00\x01'

ref_index = b'\x00\x00\x00s'

result = b'\x00\x00\x00\x02'

retry_timer = b'\x00\x00\x00\x13'

signature = b'\x00\x00\x00v'

source_domain_id = b'\x00\x00\x00y'

source_node_id = b'\x00\x00\x00U'

source_user_id = b'\x00\x00\x00Q'

static_entry = b'\x00\x00\x000'

stats = b'\x00\x00\x00\x0f'

status = b'\x00\x00\x00\x00'

transaction_data = b'\x00\x00\x00p'

transaction_data_format = b'\x00\x00\x00|'

transaction_id = b'\x00\x00\x00a'

transaction_id_list = b'\x00\x00\x00b'

transaction_tree = b'\x00\x00\x00r'

transactions = b'\x00\x00\x00q'

txid_having_cross_ref = b'\x00\x00\x00z'

user_id = b'\x00\x00\x00`'

user_list = b'\x00\x00\x009'

class bbc1.core.message_key_types.**Message**

Bases: `object`

Message parser

HEADER_LEN = 8

parse()
: Parse the message in the buffer

recv(*dat*)
: Append message to the buffer

class bbc1.core.message_key_types.**PayloadType**
: Bases: object

Type_any = 1

Type_binary = 0

Type_encrypted_msgpack = 3

Type_msgpack = 2

bbc1.core.message_key_types.**convert_from_binary**(*data_type*, *dat*)
: Deserialization from simple serialization

bbc1.core.message_key_types.**derive_shared_key**(*private_key*, *serialized_pubkey*, *shared_info*)
: Utility for deriving shared key in ECDH procedure

bbc1.core.message_key_types.**deserialize_data**(*payload_type*, *dat*)
: Utility for deserializing the received message

bbc1.core.message_key_types.**get_ECDH_parameters**()
: Utility for initialization of ECDH parameters

bbc1.core.message_key_types.**make_TLV_formatted_message**(*msg*)
: Utility for simple serialization function

bbc1.core.message_key_types.**make_binary**(*dat*)
: Simple serialize function

Basically, Type-Length-Value format is created for each item.

bbc1.core.message_key_types.**make_dictionary_from_TLV_format**(*dat*)
: Utility for simple deserialization function

bbc1.core.message_key_types.**make_message**(*payload_type*, *msg*, *payload_version=0*, *key_name=None*)
: Utility for making serialized message data

bbc1.core.message_key_types.**set_cipher**(*shared_key*, *nonce*, *key_name*, *hint*)
: Set shared key to the encryptor and decryptor

Encryptor and Decryptor are created for each inter-node connection

bbc1.core.message_key_types.**to_2byte**(*val*, *offset=0*)

bbc1.core.message_key_types.**to_4byte**(*val*, *offset=0*)

bbc1.core.message_key_types.**unset_cipher**(*key_name*)

bbc1.core.query_management module

class `bbc1.core.query_management.`**`QueryEntry`**(*expire_after=30*, *callback_expire=None*, *callback=None*, *callback_error=None*, *interval=0*, *data={}*, *retry_count=-1*)

Bases: `object`

Callback manager

`callback`()
Call a callback function for successful case

`callback_error`()
Call a callback function for failure case

`deactivate`()
Deactivate the entry

`update`(*fire_after=None*, *expire_after=None*, *callback=None*, *callback_error=None*, *init=False*)
Update the entry information

Parameters

- **`fire_after`** (*`float`*) – set callback (periodical) to fire after given time (in second)
- **`expire_after`** (*`float`*) – set expiration timer to given time (in second)
- **`callback`** (*`obj`*) – callback method that will be called periodically
- **`callback_error`** (*`obj`*) – callback method that will be called when error happens
- **`init`** (*`bool`*) – If True, the scheduler is sorted again

`update_expiration_time`(*expire_after*)
Update the expire timer

Parameters **`expire_after`** (*`float`*) – new expiration time in second

class `bbc1.core.query_management.`**`Ticker`**(*tick_interval=0.049*)

Bases: `object`

Clock ticker for query timers

`del_entry`(*nonce*)
Delete an entry from the scheduler identified by nonce

`get_entry`(*nonce*)
Get an entry identified by nonce

`bbc1.core.query_management.`**`get_ticker`**(*tick_interval=0.049*)

bbc1.core.repair_manager module

class `bbc1.core.repair_manager.`**`RepairManager`**(*network=None*, *domain_id=None*, *workingdir='.'*, *loglevel='all'*, *logname=None*)

Bases: `object`

Data repair manager for forged transaction/asset

`REQUEST_REPAIR_ASSET_FILE = 1`

`REQUEST_REPAIR_TRANSACTION = 0`

`REQUEST_TO_SEND_ASSET_FILE = 4`

`REQUEST_TO_SEND_TRANSACTION_DATA = 2`

`RESPONSE_ASSET_FILE = 5`

`RESPONSE_TRANSACTION_DATA = 3`

`exit_loop`()

Exit the manager loop

`put_message`(*msg=None*)

append a message to the queue

bbc1.core.topology_manager module

class `bbc1.core.topology_manager.`**`TopologyManagerBase`**(*network=None*, *config=None*, *domain_id=None*, *node_id=None*, *loglevel='all'*, *logname=None*)

Bases: `object`

Network topology management for a domain

This class defines how to create topology, meaning that who should be neighbors and provides very simple topology management, that is full mesh topology. If P2P routing algorithm is needed, you should override this class to upgrade functions. This class does not manage the neighbor list itself (It's in BBcNetwork)

`NEIGHBOR_LIST_REFRESH_INTERVAL = 300`

`NOTIFY_NEIGHBOR_LIST = b'\x00\x00'`

`make_neighbor_list`()

make nodelist binary for advertising

`notify_neighbor_update`(*node_id*, *is_new=True*)

Update expiration timer for the notified node_id

Parameters

- **node_id** (*bytes*) – target node_id
- **is_new** (*bool*) – If True, this node is a new comer node

process_message (*msg*)
Process received message

Parameters **msg** (*dict*) – received message

stop_all_timers ()
Invalidate all running timers

update_refresh_timer_entry (*new_entry=True*, *force_refresh_time=None*)
Update expiration timer

bbc1.core.user_message_routing module

class bbc1.core.user_message_routing.**UserMessageRouting** (*networking*, *domain_id*, *loglevel='all'*, *logname=None*)

Bases: object

Handle message for clients

CROSS_REF_ASSIGNMENT = b'\x00\x05'

JOIN_MULTICAST_RECEIVER = b'\x00\x03'

LEAVE_MULTICAST_RECEIVER = b'\x00\x04'

MAX_CROSS_REF_STOCK = 10

REFRESH_FORWARDING_LIST_INTERVAL = 300

RESOLVE_TIMEOUT = 5

RESOLVE_USER_LOCATION = b'\x00\x00'

RESPONSE_NO_SUCH_USER = b'\x00\x02'

RESPONSE_USER_LOCATION = b'\x00\x01'

process_message (*msg*)
Process received message

Parameters **msg** (*dict*) – received message

register_user (*user_id*, *socket*, *on_multiple_nodes=False*)
Register user to forward message

Parameters

- **user_id** (*bytes*) – user_id of the client
- **socket** (*Socket*) – socket for the client

- **on_multiple_nodes** (*bool*) – If True, the user_id is also registered in other nodes, meaning multicasting.

send_message_to_user (*msg*, *direct_only=False*)
Forward message to connecting user

Parameters

- **msg** (*dict*) – message to send
- **direct_only** (*bool*) – If True, _forward_message_to_another_node is not called.

send_multicast_join (*user_id*, *permanent=False*)
Broadcast JOIN_MULTICAST_RECEIVER

send_multicast_leave (*user_id*)
Broadcast LEAVE_MULTICAST_RECEIVER

set_aes_name (*socket*, *name*)
Set name for specifying AES key for message encryption

Parameters

- **socket** (*Socket*) – socket for the client
- **name** (*bytes*) – name of the client (4-byte random value generated in message_key_types.get_ECDH_parameters)

stop_all_timers ()
Cancel all running timers

unregister_user (*user_id*, *socket*)
Unregister user from the list and delete AES key if exists

Parameters

- **user_id** (*bytes*) – user_id of the client
- **socket** (*Socket*) – socket for the client

class bbc1.core.user_message_routing.**UserMessageRoutingDummy** (*networking*, *domain_id*, *loglevel='all'*, *logname=None*)

Bases: *bbc1.core.user_message_routing.UserMessageRouting*

Dummy class for bbc_core.py

process_message (*msg*)
Process received message

Parameters msg (*dict*) – received message

register_user (*user_id*, *socket*, *on_multiple_nodes=False*)
Register user to forward message

Parameters

- **user_id** (*bytes*) – user_id of the client
- **socket** (*Socket*) – socket for the client
- **on_multiple_nodes** (*bool*) – If True, the user_id is also registered in other nodes, meaning multicasting.

send_message_to_user (*msg*, *direct_only=False*)
Forward message to connecting user

Parameters

- **msg** (*dict*) – message to send
- **direct_only** (*bool*) – If True, _forward_message_to_another_node is not called.

send_multicast_join(*user_id*, *permanent=False*)
Broadcast JOIN_MULTICAST_RECEIVER

stop_all_timers()
Cancel all running timers

unregister_user(*user_id*, *socket=None*)
Unregister user from the list and delete AES key if exists

Parameters

- **user_id** (*bytes*) – user_id of the client
- **socket** (*Socket*) – socket for the client

bbc1.core.user_message_routing.**direct_send_to_user**(*sock*, *msg*, *name=None*)

1.1.1.3 Module contents

1.2 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index