MARGINCALL

Core - Currency eXchange (CoreCX)

Intro

CoreCX consists of two conceptual layers: Interface Layer and Engine Layer. The first layer supports the external client access to the second layer, which is responsible for the trading process itself.

Interface Layer is responsible for processing client connections via TCP. Native JSON protocol is used to communicate with core (acts alike JSON RPC). Technically, this layer awaits incoming TCP connections listening to default ports 1330, 1340, 1350.

Client-server interaction can be presented as a simple “client request – server response” model. Both request and response are in JSON format. Default port for this interaction is 1330.

Apart from this, core is supposed to be sending PUSH-messages with the use of default port 1350. There is a need for a daemon (with or without MQ system like RabbitMQ) to process these messages and act accordingly (e.g. persist market data with the use of PGSQL).

Core connection

TCP client connects to a running CoreCX instance via port 1330 (default). TcpClient repo contains everything needed to execute core functions remotely.

Communication

Client-core communication involves (by the client):

1. Establishing a TCP connection
2. Sending function calls with parameters
3. Receiving function call queueing status (success with FuncCallId or reject with an ErrorCode)

Both request and response have a minimized (“technical”) JSON format, each JSON object is appended with ‘\n’ newline symbol. This symbol is used to separate several messages received per one socket read operation.

Obviously, these JSON strings are transmitted as byte arrays (received from ASCI strings).

Sending a command into CoreCX is an attempt to call one of many core functions. Core responses with one of the following: Статусом постановки функции в очередь ядра на исполнение

1. Status indicating function queueing to execute
2. Status of function execution

Important: it is possible to use a single connection for a single command as well as a single connection for multiple commands.

Requests and responses

TCP client should form its request as follows: (minimized JSON):

{“0”: <core function code>, “1”: <argument 1>, “2”: < argument 2>, …}

Key “0” is core function code (which is integer), function arguments are provided afterwards (keys always increment by 1).

Attention! Every argument should be serialized to JSON in the same data type as a function expects it.

E.g. the following query

{“0”: 500, “1”: 1, “2”: “btc”, “3”: 300.515}

Calls core function #5 (deposit a trading account) with the following parameters: user_id equals 1, currency is “btc” and deposit amount is 300.515. Arguments has been passed in expected data types (Int32 user_id, String currency, decimal amount).

Another example – CreateAccount function:

{“0”: 100, “1”: 1}

It is a simple function which is responsible for the in-memory creation of user account with an ID given.

Upon receiving a command like {“0”: <core function code>, “1”: <arg 1>, “2”: < arg 2>, …} CoreCX tries to parse your function call, attach ID to it (func_call_id) and queue it for execution.

CoreCX **replies twice** in case of a successful function execution: first time with a func_call_id attached and second with an execution status. In case of an invalid function call CoreCX replies with an error code.

Response #1

Core responses instantly upon receiving a command. This response’s format is as follows:

{“0”: <status code>, “1”: <func_call_id [if status code = 0 (success)]>}

Key “0” always indicates a core queueing status.

Key “1” is set if key “0” equals 0 (success). In this case key “1” contains an integer function call ID value.

Possible responses:

{“0”: 0, “1”: 1653} //function call queued successfully, ID has been assigned: func_call_id = 1653

{“0”: 24} //invalid function arguments passed

{“0”: 25} //function with such ID not found

{“0”: 26} //invalid JSON input

{“0”: 40} //market closed

Response #2

CoreCX sends the second response upon function execution:

{“0”: <func_call_id>, “1”: <function execution status>, “2”: <return value 1>, “3”: <return value 2>, …}

Key “0” always indicates a long func_call_id value.

If there are no return values core will reply with just 2 figures:

{“0”: <func_call_id>, “1”: <function execution status>}

Core response sample

After sending function call to create an account (ID 100) a TCP client will receive 2 messages over TCP/IP:

{“0”: 0, “1”: 1653}\n

{“0”: 1653, “1”: 0}\n

//function queued successfully with assigned func_call_id = 1653 and successfully executed

Core function codes

The following function codes are necessary to call CoreCX functions:

CreateAccount = 100,

SuspendAccount = 200,

UnsuspendAccount = 300,

DeleteAccount = 400,

DepositFunds = 500,

WithdrawFunds = 600,

PlaceLimit = 700,

PlaceMarket = 800,

CancelOrder = 900,

SetAccountFee = 1000,

GetAccountBalance = 2400,

GetAccountParameters = 2500,

GetAccountFee = 2600,

GetOpenOrders = 2700,

GetOrderInfo = 2800,

CreateCurrencyPair = 5000,

GetCurrencyPairs = 5100,

GetDerivedCurrencies = 5200,

DeleteCurrencyPair = 5300,

GetTicker = 7000,

GetDepth = 7100,

CloseMarket = 8800,

OpenMarket = 8900,

BackupCore = 9000,

RestoreCore = 9100,

ResetFuncCallId = 9500

Status codes

CoreCX may return the following status codes:

Success = 0,

ErrorAccountAlreadyExists = 1,

ErrorAccountNotFound = 2,

ErrorAccountAlreadySuspended = 3,

ErrorAccountAlreadyUnsuspended = 4,

ErrorAccountSuspended = 5,

ErrorCrossUserAccessDenied = 6,

ErrorInsufficientFunds = 7,

ErrorIncorrectOrderKind = 8,

ErrorOrderNotFound = 9,

ErrorInsufficientMarketVolume = 10,

ErrorBorrowedFundsUse = 11,

ErrorNegativeOrZeroValue = 12,

ErrorNegativeOrZeroId = 13,

ErrorApiKeyNotPrivileged = 14,

ErrorIncorrectStopLossRate = 15,

ErrorIncorrectTakeProfitRate = 16,

ErrorIncorrectTrailingStopOffset = 17,

ErrorApiKeysLimitReached = 18,

ErrorApiKeyNotFound = 19,

ErrorSignatureDuplicate = 20,

ErrorNonceLessThanExpected = 21,

ErrorIncorrectSignature = 22,

ErrorNegativeOrZeroLimit = 23,

ErrorInvalidFunctionArguments = 24,

ErrorFunctionNotFound = 25,

ErrorInvalidJsonInput = 26,

ErrorNegativeOrZeroLeverage = 27,

ErrorIncorrectPercValue = 28,

ErrorFixAccountsLimitReached = 29,

ErrorFixRestartFailed = 30,

ErrorFixAccountAlreadyExists = 31,

ErrorFixAccountNotFound = 32,

ErrorFixSymbolNotFound = 33,

ErrorFixFieldsNotSet = 34,

ErrorFixInvalidClOrdID = 35,

ErrorFixUnknownOrderType = 36,

ErrorFixInvalidOrderId = 37,

ErrorSnapshotBackupFailed = 38,

ErrorSnapshotRestoreFailed = 39,

ErrorMarketClosed = 40,

ErrorMarketAlreadyClosed = 41,

ErrorMarketAlreadyOpened = 42,

ErrorMarketOpened = 43,

ErrorBackupRestoreInProc = 44,

ErrorIPDuplicate = 45,

ErrorInvalidCurrency = 46,

ErrorInvalidCurrencyPair = 47,

ErrorCurrencyNotFound = 48,

ErrorCurrencyPairNotFound = 49,

ErrorCurrencyPairAlreadyExists = 50,

ErrorStopLossUnavailable = 51,

ErrorTakeProfitUnavailable = 52,

ErrorTrailingStopUnavailable = 53,

ErrorIncorrectDelayValue = 54,

Unknown = 99