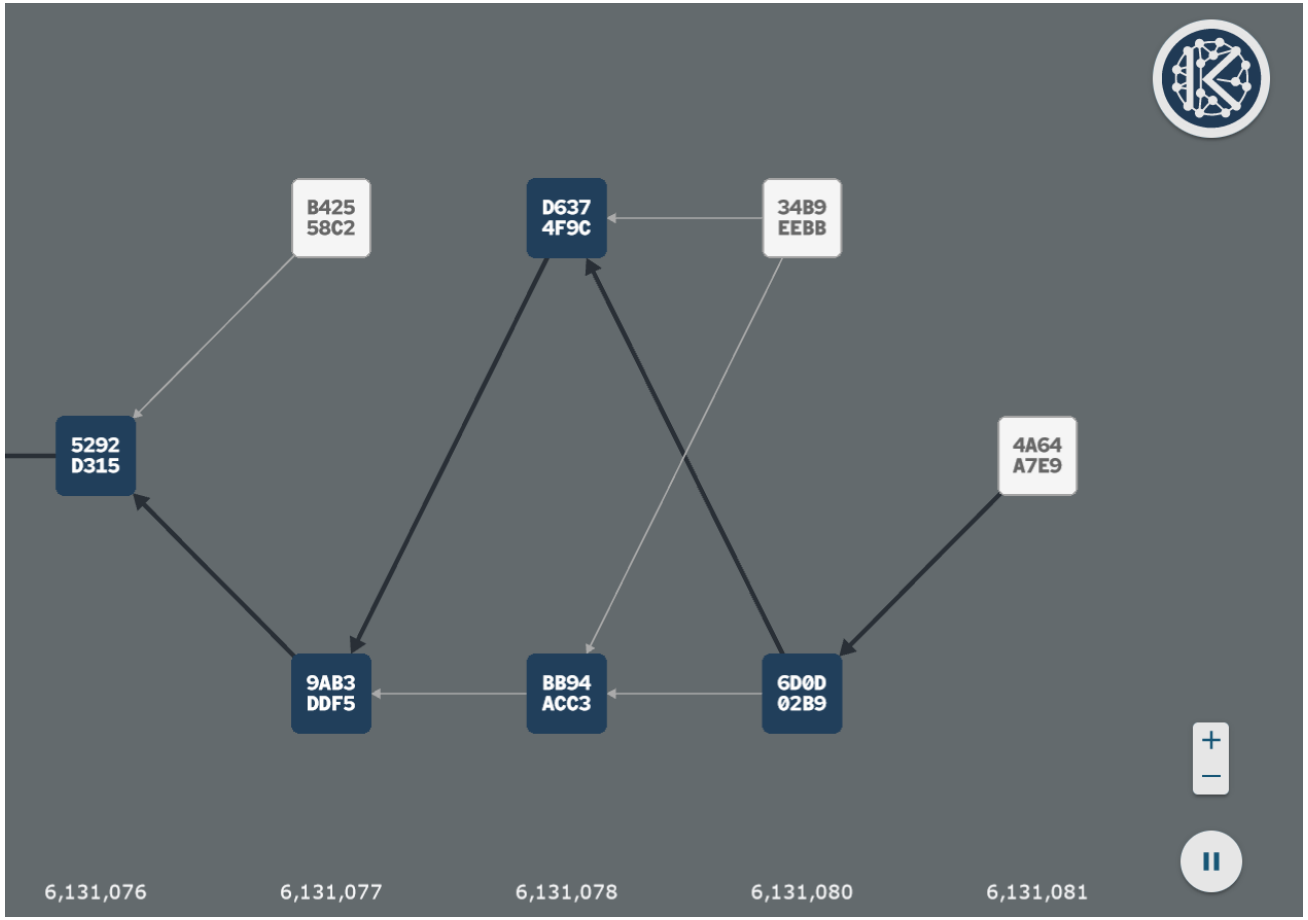
Integration of Karlsen BlockDAG

In the following slides you will find the most important information needed in order to integrate Karlsen into your application.

How does the BlockDAG look like

- In the BlockDAG blocks can be created in parallel.
- Every block tries to merge other already known block-tips which don‘t have children yet.
- The numbers on the x-axis are the daa_scores.
- Daa_score = Total number of merged blocks from the point of view of blocks in this column.



Karlsend – The Karlsen Node

and the REST-API

Karlsend is the Karlsen Node

- Source Code:
  - https://github.com/karlsen-network/karlsend
- Karlsen Node API:
  - gRPC + Protocol buffers spec
- gRPC Documentation:
  - https://github.com/Karlsen-network/karlsend/blob/master/infrastructure/network/netadapter/server/grpcserver/protowire/README.md
- Protobuf specification:
  - https://github.com/karlsen-network/karlsend/blob/master/infrastructure/network/netadapter/server/grpcserver/protowire/messages.proto
  - https://github.com/karlsen-network/karlsend/blob/master/infrastructure/network/netadapter/server/grpcserver/protowire/rpc.proto

Karlsen REST-API

For even simpler access to the Karlsen Node the Karlsen developers created a REST-API, which provides the most important commands.

- SWAGGER Docs: https://api.karlsencoin.com
- API-endpoint is free to use for everyone.

You can also create your own REST-API instance using the open source:

- https://github.com/karlsen-network/karlsen-rest-server

or the docker container

- https://hub.docker.com/r/karlsennetwork/karlsen-rest-server

Use a Karlsen Wallet

Wallet application

- We suggest to use:
  - Karlsenwallet: A wallet written in go, which is provided with the karlsend node binary package. There are executables for Windows, Linux, macOS and Arm.
  - When you create such a wallet application, it reads UTXOs from the node via GetUtxosByAddresses and submits signed transactions with SubmitTransaction.

Karlsenwallet 1/2

- The wallet application needs a connection to a Karlsen node. This is solved by a dedicated wallet daemon (acts as a client of karlsend and as a server for wallet commands).
- For the communication with the daemon you can use either the executable or another wallet-specific gRPC interface exposed by the wallet daemon.
- First you need to create a new wallet with the following instruction
  - karlsenwallet create
- The wallet application needs a connection to a Karlsen node. This is solved with a wallet daemon client, which should be running.
  - karlsenwallet start-daemon /s <karlsen node IP>:42110
- Note: You can use a local or a remote node

Karlsenwallet 2/2

- You can also use the wallet gRPC interface. The protocol buffers spec for this API can be found here:
  - https://github.com/karlsen-network/karlsend/blob/master/cmd/karlsenwallet/daemon/pb/kaspawalletd.proto
- When the daemon is running you can check the balance and send transactions:
  - karlsenwallet balance

    Total balance, KLS 2.55324509
  - karlsenwallet send /t karlsen:qqe3p64wpjf5y27kxppxrgks298ge6lhu6ws7ndx4tswzj7c84qkjlrspcuxw /v 0.1

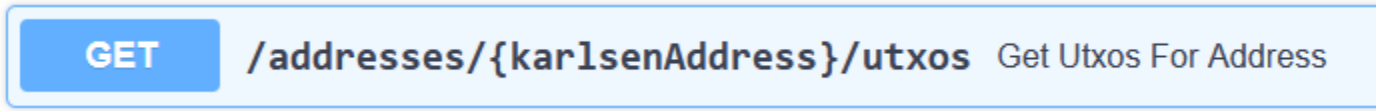
    Transactions were sent successfully

    Transaction ID(s):

    845bb303acc717a4988de1e19b1cbbc80652490af9414893ecfd5dabdb00f45d
- Using gRPC the commands are: GetBalance(), Send()
- Note: For SendRequest the wallet-password is needed, so the use of a secured connection is recommended
- Hint: When sending requests, you can use useExistingChangeAddress to force the HDWallet sending the change value to a specified address

Creating an own wallet

If you want to use your own wallet implementation, you can use the Node's gRPC API or the Karlsen REST-API for fetching UTXOs:

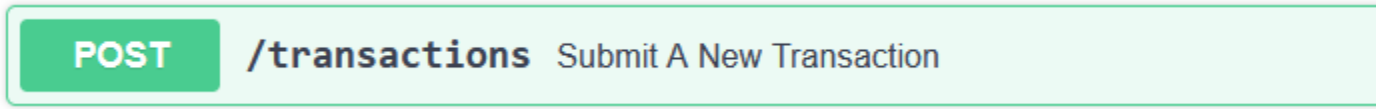


GetUtxosByAddressesRequestMessage
GetUtxosByAddressesRequestMessage requests all current UTXOs for the given karlsend addresses

This call is only available when this karlsend was started with `--utxoindex`

Field	Type	Label	Description
addresses	string	repeated	

And sending transactions



SubmitTransactionRequestMessage
SubmitTransactionRequestMessage submits a transaction to the mempool

Field	Type	Label	Description
transaction	RpcTransaction		
allowOrphan	bool		

The default signing algorithm is Schnorr. You can also use ECDSA. See [karlsenwallet signing algorithm]

Check Transactions

When is a transaction confirmed?

How to index all blocks and transactions

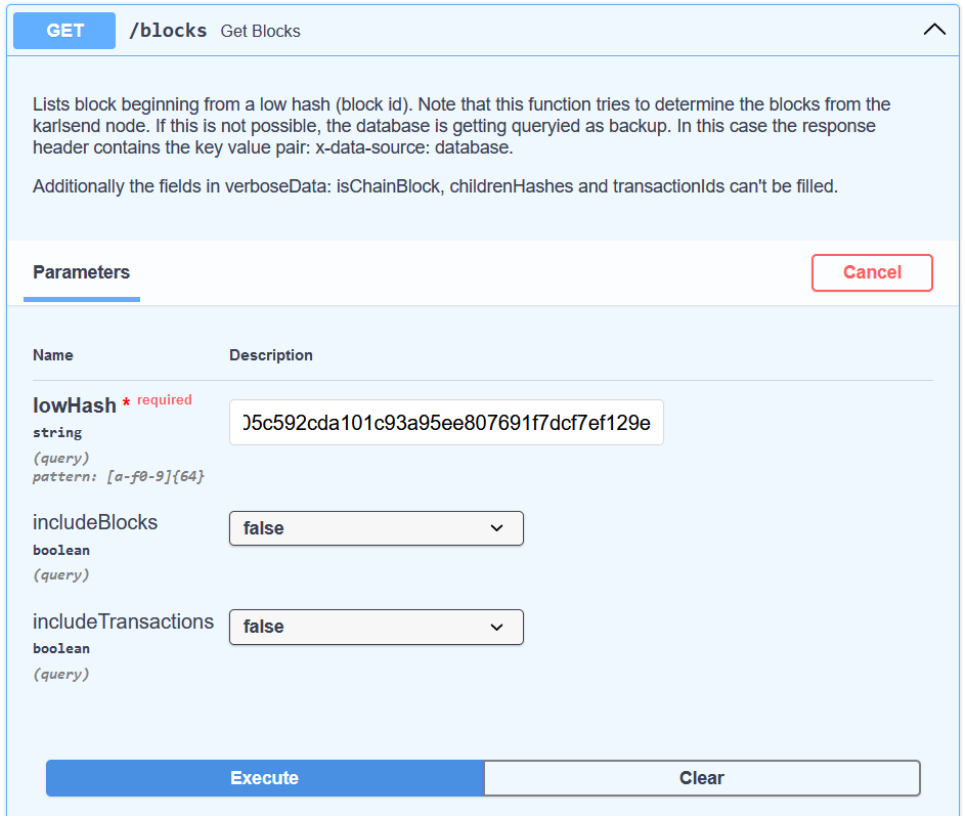
1. Define a lowHash being either the last one indexed or pruningPointHash returned by calling GetBlockDagInfo
2. Call GetBlocks with the lowHash, including blocks and transactions
3. Cache/Save the blocks and their transactions. The mapping between a transaction and its block is important and should be recorded.
4. Go to 2 and use the last blocks hash as the new lowHash

Hint: With this procedure you won‘t miss any block and transactions.

GetBlocksRequestMessage
GetBlocksRequestMessage requests blocks between a certain block lowHash up to this karlsend’s current virtual.

Field	Type	Label	Description
lowHash	string		
includeBlocks	bool		
includeTransactions	bool		

See the BlockProcessor.py for a battle-tested example.



Simplify the BlockDAG with VirtualSelectedParentChain (VSPC)

In the BlockDAG picture (see KGI) you see the **thick** black arrows.

These arrows are pointing at blocks, which are selected as a “chain_block”. In every daa_score there can be exactly **one** chain_block.

Important Rules

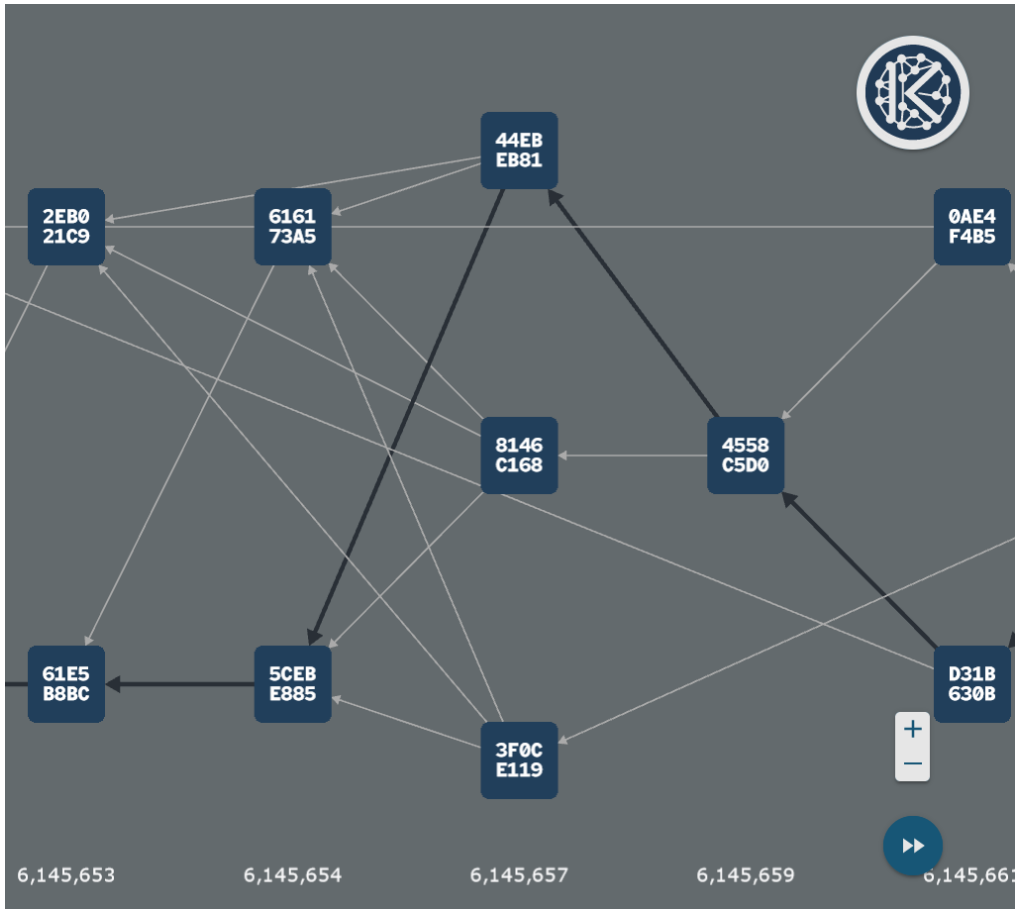
1. A single Transaction can appear in multiple blocks.
2. The chain block dictates which of the transactions in the merged blocks are accepted.

Reorg - Very important to know:

The virtual parent chain can have a so-called **Reorg**. That means, that blocks, which are chain blocks in the present, can loose this state and another block in the same daa_score can start being the chain_block instead.

Consequence

It's possible that transactions, which were declared as accepted, are not accepted anymore. This happens only in the most recent blocks around the DAG tips.



An accepted transaction has one accepting block

For checking whether a transaction has been accepted, an additional variable is required in the cache/database per transaction. We call this variable acceptingBlockHash. It will be used on the next slide, but first a few words:

- The acceptingBlockHash describes which chain_block has accepted this transaction (note that accepted does not mean the tx is *included* in the block, acceptance is a *logical* concept)
- Since reorgs can occur in the VSPC, it is possible that the acceptingBlockHash changes.
- A transaction can only have one accepting block at a given moment.

What happens in a reorg?

When a reorg happens, a former chain_block looses its state and a new chain_block is added instead. (see previous slide)

The effect is, that all transactions which have acceptingBlockHash = former chain_block are not considered as accepted anymore. The new chain_blocks define the accepted transactions instead.

The acceptingBlockHash has to be updated in this step to the new chain_block.

Requesting VirtualSelectedParentChain (VSPC) via gRPC

On the next slide the following two gRPC commands will be used.

Request:

GetVirtualSelectedParentChainFromBlockRequestMessage

GetVirtualSelectedParentChainFromBlockRequestMessage requests the virtual selected parent chain from startHash to this karlsend's current virtual.

Field	Type	Label	Description
startHash	string		
includeAcceptedTransactionIds	bool		

Response:

GetVirtualSelectedParentChainFromBlockResponseMessage

Field	Type	Label	Description
removedChainBlockHashes	string	repeated	The chain blocks that were removed, in high-to-low order
addedChainBlockHashes	string	repeated	The chain blocks that were added, in low-to-high order
acceptedTransactionIds	AcceptedTransactionIds	repeated	The transactions accepted by each block in addedChainBlockHashes. Will be filled only if includeAcceptedTransactionIds = true in the request
error	RPCError		

How to check which transactions are accepted

Go through the VirtualSelectedParentChain (VSPC).
The VSPC contains the information about which transactions are accepted by which chain blocks.
This information is only saved in the VirtualSelectedParentChain.

1. Request the VSPC with GetVirtualSelectedParentChainFromBlock, beginning from a start block hash with set **includeAcceptedTxIds = True**.

2. The response contains, beginning from start block, the following information:
   addedChainBlockHashes: Which blocks are added to the VSPC
   removedChainBlockHashes: Which blocks are removed from VSPC due to **reorgs**
   acceptedTransactionIds: Which transactions got accepted by the VSPC

3. For each transaction which acceptingBlockHash is in removedChainBlockHashes, set
   accepted = **false**

4. Go through AcceptedTransactionIds, and set in your cache/database for these transactions:
   accepted = **true**
   acceptingBlockHash = acceptingBlockHash from response
   // note that each entry in the list has an acceptingBlockHash and a sub-list of tx ids

   ⚠ Requesting the VirtualSelectedParentChain might return more blocks than you know / have indexed.
   If you haven't indexed the acceptingBlockHash yet, then jump to 1) and use the last known acceptingBlockHash as the startHash for the next call (this loop should operate with a timer every ~1 seconds)

Hint: It fequently may happen, that a transaction gets unaccepted by removedChainBlockHashes and immediately gets accepted with acceptedTransactionIds again.

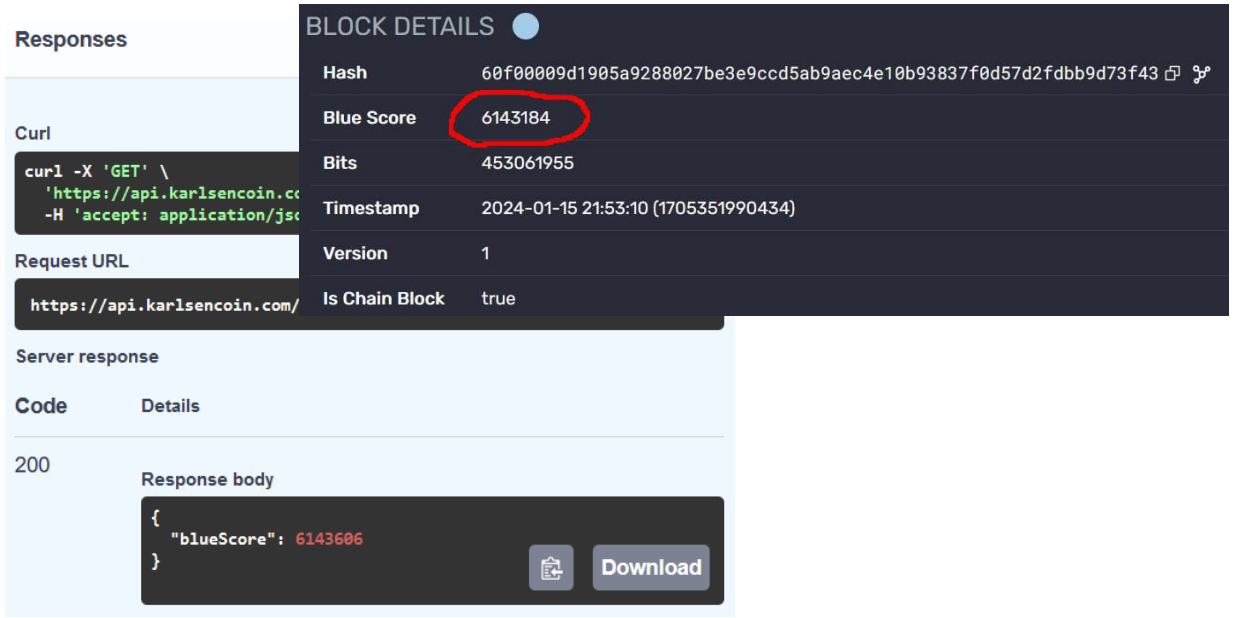
See the VirtualChainProcessor.py for a battle-tested example.

Confirmation of blocks and its transactions via BlueScore

For checking the confirmations, you need the BlueScore, which is the total sum of blue blocks in the BlockDAG

1. Get the current bluescore of the VSPC with GetVirtualSelectedParentBlueScoreRequest
2. Get the acceptingBlockHash's blueScore for your TxId to be checked and substract if from the **current bluescore**.
   (currentVspcBluescore) – (acceptingBlockHash's blueScore) = confirmations

In the following example we have 422 confirmations.



gRPC Request and Response:

GetVirtualSelectedParentBlueScoreRequestMessage

GetVirtualSelectedParentBlueScoreRequestMessage requests the blue score of the current selected parent of the virtual block.

GetVirtualSelectedParentBlueScoreResponseMessage

Field	Type	Label	Description
blueScore	uint64		
error	RPCError		

Examples written in Python

Mini example

This is a minimalistic example to show how to fetch blocks, TXs and checking, if TXs accepted. If you understand this already, you can proceed with the database filler example.

https://github.com/karlsen-network/karlsen-check-txs-example/blob/main/main.py

Database filler example

This is the code which is used to fill our database with all blocks, transactions and their accepted-state. Reading blocks/tx and checking the VSPC are running in parallel.

https://github.com/karlsen-network/karlsen-db-filler/blob/main/BlocksProcessor.py

https://github.com/karlsen-network/karlsen-db-filler/blob/main/VirtualChainProcessor.py