QuickBlocks Documentation

Release 0.2.0 beta

QuickBlocks <info@quickblocks.io>

Oct 16, 2017

CONTENTS:

CHAPTER

ONE

INTRODUCTION

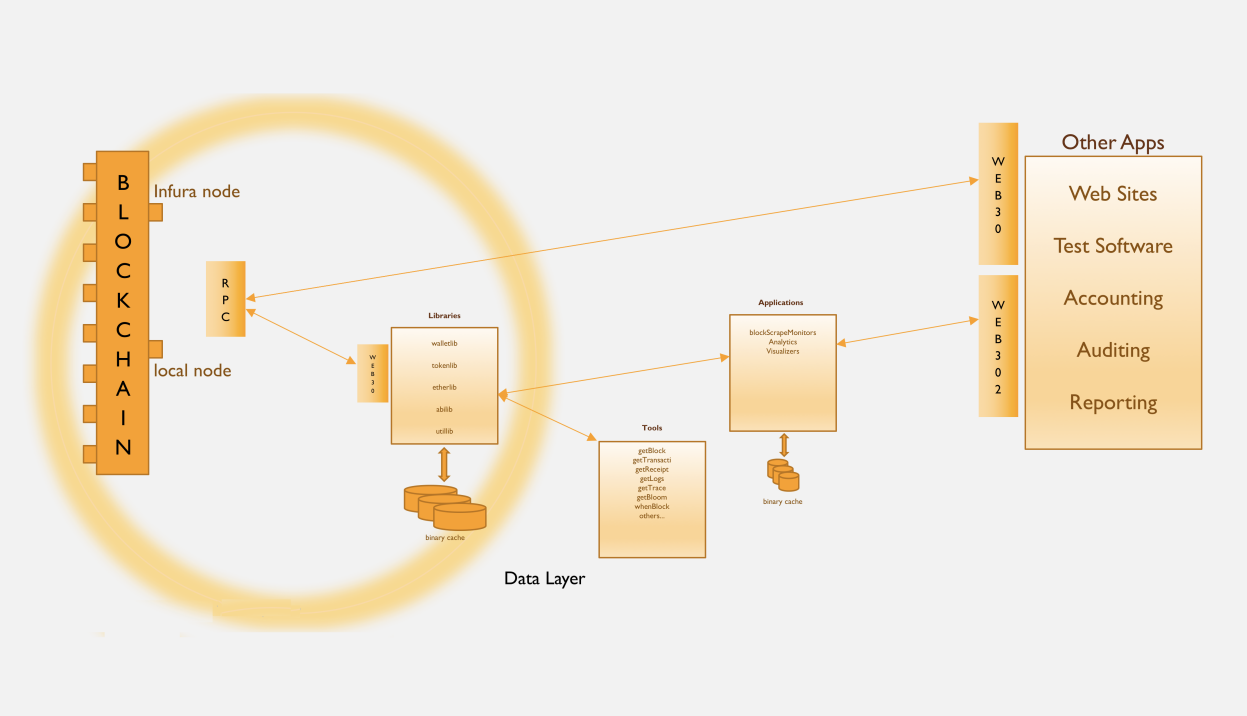
QuickBlocks is a collection of software libraries, applications, and command-line tools designed to give you quick access to the data provided by an Ethereum node.

1.1 Overview

After the infamous DAO hack and subsequent fork in the summer of 2016, the WhiteHats needed an accurate list of token account holders so they could be reimbursed. The ICO had taken place a month earlier, and during that sale, after an initial fifteen day period where the token cost was at a ratio of 1:100, the ratio rose in a daily step function until the end of the sale. As token holders, we were very interested, of course, in the process. After a few days the Python code that was used to generate that off-chain list of token holders was presented to the world for its review. That code took over 20 hours to generate a list of transactions on the DAO during the last fifteen days of the sale. *This bears repeating: it took 20 hours to accumulate 15 days worth of data!*. The word "Quick" in the name of our project is in direct response to this shortcoming in the ecosystem.

1.2 Architecture

In a manner very similar to the way web3.js works, QuickBlocks sits between a locally running Ethereum node (or any node, local or remote, for that matter), and delivers the Ethereum data to your application. There are two significant improvements to the web3.js path however. First, QuickBlocks caches the data locally which means that the 20 hours mentioned above is reduced to mere minutes. Secondly, if you provide QuickBlocks with the ABI to your smart contract, it can deliver what we call articulated data. By this we mean that instead of returning data in the language of the Ethereum node (blocks and transactions and receipts and logs), we return data in the language of your smart contract (transfers and votes and proposals and expenditures). This eases the burden on your dApp developers.



Whereas the web3.js library delivers nearly identical data as is retrieved from the RPC interface, making it difficult for any but the most well versed in the data to easily use it, QuickBlocks stands between the node and your application improving the data significantly in two ways: (1) it's way faster, and (2) it's translated into the language of the smart contract. See this FAQ item, in response the nagging question in your head about the quality and accuracy of the data.

1.2.1 Terminology

- **cache** - This refers to QuickBlocks' database cache which you may store on any hard drive (an SSD drive is much preferred). The QuickBlocks cache allows us to return data between 50 and 100 times faster than one can retrieve the same data through web3.js and the RPC alone.
- **articulated data** - QuickBlocks is able to parse and "articulate" the data returned by the RPC, so that you get votes and proposals and onTokenPurchase events rather than the current hexadecimal mess returned by web3.js and the RPC (oh, and did we mention, it's a lot faster!)
- **block_list** - Many of our tools take a *block_list* as a command line parameter. A block_list may be one or more valid block numbers (hex or integer), a range of valid block lists, one of a list of customizable *special* blocks such as 'byzantium' or 'doafork', or any combination of the above.
- **trans_list** - a *trans_list*, like a block_list, comes in many forms and is used by various of our command line tools. A trans_list may be one or more transaction hashes, block numbers followed by a transaction index in the block (blockNum.transID), a block hash followed by a transaction ID (blockHash.transID), or any combination. *trans_list* is used in many tools.
- **address_list** - *address_list* items are lists of one or more Ethereum addresses which are 42 character hexadecimal strings starting with '0x'. In the near future, we hope to add 'named accounts' which will be Ethereum addresses as picked up from the ENS smart contract; however, this feature is still in the works. Stay tuned.

1.2.2 Methodology

QuickBlocks is old school. We abhor JSON data, which in our estimation, is literally the worst way to deliver data imaginable (although one could add random strings of arbitrary meaningless data just to make it more terrible–why does every record carry every field name–JSON data–the format endorsed by the Department of Redundancy Department!). JSON is wonderful and useful *if you're transferring data across a wire and if the receiver of the data doesn't know the data's format and if the each record may follow a different format*, but none of these things is true for the Ethereum data. In the case where the data is remote (for example if you're using Infura), then you have to receive the data in a wire-neutral format, but there's no reason to keep it that way (thus QuickBlocks caching mechanism). Here's some example code for visiting every block on the chain and printing its hash:

```
/*-------------------------------------------------------------------------
 * Name:    Simple
 * Purpose: To provide the easiest introduction to the QuickBlocks library.
 *          Simply request a block from Infura and print it to the screen.
 *-------------------------------------------------------------------------*/
#include "etherlib.h"
int main(int argc, const char *argv[]) {

    // Initialize the system and tell it where to find the blockchain data.
    etherlib_init("infura");

    // Request the 3,500,000th block from Infura. Store it in 'block'
    CBlock block;
    getBlock(block, 3500000);

    // Print the block to the screen
    cout << block << "\n";

    return 0;
}
```

1.3 Resources

Title	Link
Website	http://quickblocks.io
GitHub	http://github.com/Great-Hill-Corporation/quickBlocks
Twitter	http://twitter.com/@quickBlocks
Medium	http://medium.com/@tjayrush

CHAPTER

TWO

QUICKSTART

Start with the installation instructions (which are slightly different per operating system). Here's the instructions for Ubuntu:

```
(sudo) apt-get update
(sudo) apt-get upgrade
(sudo) apt-get install build-essential
(sudo) apt-get install libcurl3-dev
(sudo) apt-get install cmake
(sudo) apt-get install python python-dev
```

Next compile, build, and install the executable files:

```
git clone https://github.com/Great-Hill-Corporation/quickBlocks.git .
cd quickBlocks
mkdir build
cd build
cmake ../src
make
(sudo) make install
```

Finally, play with some sample code. Here's some sample code that uses one of QuickBlocks' *forEvery* functions to visit each block in the chain and print out its logsBloom. We present it with no explanation.

```
#include "etherlib.h"
#include "options.h"

//--------------------------------------------------------------
int main(int argc, const char *argv[]) {

    etherlib_init("");

    COptions opt;
    opt.parseArguments(argc, argv);

    forEveryBlock ( displayBloom, &opt, opt.start, opt.nBlocks, opt.skip );

    return 0;
}

//--------------------------------------------------------------
bool displayBloom(CBlock& block, void *data)
{
    cout << block.blockNumber << fromBloom(block.logsBloom) << "\n";
```

```
    return true;
}
```

CHAPTER

THREE

USE CASES

Access to faster, more informative data ("articulated data") from the Ethereum blockchain opens up the possibility of a richer collection of applications than has been previously possible. Here we provide a list of potential applications that may be built on top of the QuickBlocks libraries.

3.1 Accounting / Auditing Functions

A Better RPC Interface (Web3.2)

- Improves on the experience of smart contract developers and aids regulators, auditors and accountants by providing improved access to the Ethereum data. May run either remotely (AWS, Digital Ocean) or locally against your own local-running Ethereum node.

  End users Smart contract developers, auditors/regulators, accountants

  Notes When running locally, 100s times faster than existing methods; significantly improved (articulated) data fed directly into traditional databases; programmable SDK; example source code; ability to automatically generate code; code does not need to be maintained; built in support for ERC20 tokens and wallet contracts

Smart Contract Monitoring:

- Actively monitor one (or more) Ethereum smart contracts and user accounts (or any combination) watching for odd or 'known-dangerous' transactional patterns. Report to anomalies to a list of email, SMS, web site, or individuals whenever something of interest happens.

  End Users Smart contract developers, smart contract participants (i.e. token holders)

  Notes 'Weird' things include recursive attacks, violations of invariants (token balances to ether balance), largest purchases, most active trader accounts, etc.; Could potentially spawn an "insured" smart contract industry expectation.

Smart Contract Reporting:

- Instantaneous "Quarterly" reports available every second. On demand reports generated for cap tables (report on token holders), individual ether holdings and transaction histories (i.e. bank statements) on a per-account, per-contract-group, by industry, or system wide.

  End Users Smart contract developers, smart contract participants (i.e. token holders), economists, regulators

  Notes Allows for self-reporting on business processes, expenditures, and revenue from outside an organization–no need to wait for company reports; marketing efforts might engender an expectation that every smart contract's accounting is fully transparent.

Automated Tax Returns:

- Automated tax reporting for any jurisdiction showing dates and cost of acquisitions, cost basis, holding period, dates and revenues on sales, and any tax liabilities.

  End Users Individual users, purveyors of smart contract systems, accountants, auditors

  Notes Historical spot prices from agreed-upon source for each currency and/or token can be shared across the planet. With the addition of APIs from popular crypto exchanges (i.e. Kraken, CoinBase) could also report on exchange-held accounts

Auditing Support:

Provide data and transactional information to third parties not associated with the development team of a smart contract system. Interesting to potential investors, industry analysts, auditors and/or regulators.

End Users regulators, auditors, potential investors

Note Fully parsed data makes for much easier auditing of smart contracts, could expose non-delivery of promised behavior (i.e. are "provably true" gambling sites actually paying out at the rate they claim? Gambler Watch™).

3.2 Developer Ecosystem

Testing / Debugging Aids:

- Record and play back "real world" interactions with a already deployed smart contracts. Being programmable, test engineers may use QuickBlocks to build test cases ad-hoc (fuzzing) or modify previously recorded live playbacks.

  End Users Smart contract testing engineers, smart contract developers

  Notes Could be used against proposed new versions of the smart contract (with programmatic modification of the data to meet the new contract's interface); could be used to aid in gas optimization

Account 'Clustering':

Using computational geometry code already written, visualize 'relatedness' of accounts, relationship between transactions, usage trends, upward or downward movement of token transfer activity, and myriad other things.

End Users Economists, data scientists interest in the system-wide behavior

Notes The strength of link could be related to the number of interactions between two account or the total value transferred; computational geometry (graph-based data structures) might be useful in sharding the chain (under the assumption that one would want interrelated accounts to belong to the same shard and that in most cases accounts are tightly coupled).

3.3 Consulting Opportunities

Gas Optimization

- As Ethereum progresses and it becomes possible for smart contracts to pay for their end users' gas usage, it will become increasingly important to optimize for gas usage in a smart contract. Existing tools report only on expected gas usage based on estimated behavior. QuickBlocks can report on actually "live" data captured in the wild. A consulting service or product ideas could be built on top of this capability.

Smart Contract Auditing

- Some smart contracts (SingularDTV as an example) do not have adequate logging / event generation built into them. This makes it nearly impossible to properly account for their behavior. Certain activity in a smart contract can be "protected" with semaphore events that can surround value transfers, for example. These 'semaphores'

if not properly closed would indicate a recursive attack. There is an opportunity for consulting related to full instrumenting smart contracts with "active events" that aid in the monitoring function.

CHAPTER

FOUR

TECHNICAL REFERENCE

QuickBlocks is a collection of software libraries, command-line tools, and applications. The command-line tools, while very useful in their own right, are primarily intended to serve as instructional material helping you understand how to use the libraries to accomplish various tasks. The applications are more full-featured and provide useful functions such as providing a fully-decentralized block chain scraper (blockScrape), automatic generation of source code that comprises the libraries, and the libraries themselves which, while largely auto-generated provide most of the function. Each of these apsects of QuickBlocks is described below. We intend to make the libraries and tools open source. The applications will be commercial products.

4.1 Command-Line Tools

The command-line tools present direct access to the Ethereum data structures while providing many options and export formats. They can be broken down into the following major groups:

4.1.1 Block-related tools

Four block-related tools deal primarily with the Ethereum block data structure. Each of these tools takes a **block_list** which may be one or more block numbers (as hex or integer), ranges of block numbers, special named blocks (see whenBlock), or any combination. Each of these tools is open source. The block-related tools are:

getBlock, getBloom, whenBlock, whereBlock

4.1.2 Transaction-related tools

There are four transaction-related tools that deliver Ethereum transaction data structures. Each of these tools takes a **trans_list** which may be one or more transaction hashes, blocknumber / transID pairs, blockHash / transID pairs or any combination. Each of these tools is open source. The transaction-related tools are:

getTrans, getTrace, getLogs, getReceipt

4.1.3 Account-related tools

Five account-related tools return data particular to accounts such as an accounts Ether or ERC20 token balance, an account's name, or whether or not it's a smart contract. These tools are also open source. The account-related tools are:

ethName, getAccounts, getBalance, getTokenBal, isContract

4.2 Applications

QuickBlocks tools are intended primarily sample source code to help you understand how to program using the QuickBlocks libraries. QuickBlocks applications are more full-featured applications that accomplish specific tasks: scraping the chain, building cached databases for quick retrieval of the data, etc. The other thing that distinguishes applications from tools is that while the tools and the QuickBlocks libraries are open source, the applications are sold through our website.

4.2.1 blockScraper application

The **blockScrape** application scrapes the Ethereum blockchain creating three different databases that help the other tools and applications quickly access and deliver the Ethereum shared ledger to your applications. This process is fully described in our three white papers,. This tool is not open source. It is available for purchase from our website.

4.2.2 grabABI application

The **grabABI** program pulls your smart contract's ABI file from either Etherscan or (in the future) the ENS smart contract. Many things are possible with your smart contract's ABI file including automatic generation of source code needed to build a fully functional smart contract monitor / debugger. grabABI also is able to encode and decode the functional and event signatures needed to fully parse the blockchain data returned through the RPC. This tool is not open source. It is available for purchase from our website.

4.2.3 makeClass application

The **makeClass** application takes a class definition file and generates a fully detailed, self describing, easily serializable C++ class that carries all the information necessary to store the entire Ethereum blockchain. Surprisingly, nearly 75% of all of the code that makes up the QuickBlocks libraries is generated automatically through the use of the **makeClass** application. In conjunction with the **grabABI** application an **makeClass** all of the code in both the token library and the wallet library was generated automatically with from simple configuration files. An important thing to note is that **grabABI** and **makeClass** can be used to create a full parsing / monitoring library for your smart contracts. With a simple command line pointing to your ABI file, a full feature smart contract monitor / transaction debugger may be created. Here is an example of a **makeClass** class definition file:

```
[settings]
class      = CBlock
fields     = gas gasLimit|gas gasUsed|hash hash|bloom logsBloom|uint32␣
↪blockNumber|hash parentHash|timestamp timestamp|CTransactionArray transactions
includes   = ethtypes.h|abilib.h|transaction.h
c_includes = etherlib.h
scope      = extern
serialize  = true
```

4.2.4 Smart contract monitors / transaction debuggers

While not yet fully ready, QuickBlocks provides a very powerful technology that we call smart contract monitors or transactional debuggers. After recording the full history of every internal and external transactions on your smart contracts (or collection of smart contracts and other addresses), a QuickBlocks monitor is able to replay the transactions step by step. At each block, the monitor may do a full "bank

reconciliation" asking the node for account balances and double checking those balances against the transaction history of those accounts. At any block, if the account balances don't match, then the most likely cause is a bug in our code–in fact, finding these kind of mismatches allows us to improve our code. We highly doubt we will find a bug in the Ethereum node (but, boy, wouldn't that be amazing?). We do however think that there are bugs to be found in the various smart contracts. We believe QuickBlocks monitors can find and warn users about various types of smart contract bugs by doing a token reconciliation at the end of each block. The recursive DAO bug exhibited an incorrect relationship on the first block of the hack. Humans didn't notice it for six hours. QuickBlocks could have identified it on the first block.

4.3 Libraries

4.3.1 utillib

The most basic functionality is contained in a library called **utillib**. This library consists of software code for carrying out common functions such as string and time manipulation; concurrency-protected data access; container classes such as lists, arrays and maps; and other utility functionality.

4.3.2 etherlib

The second library component is called etherlib. This library allows for reading, writing, and manipulation of application binary interface (ABI) files. This library also mirrors the blocks, transactions, receipts, traces, and accounts found in the blockchain data. It is in the etherlib that we interact directly with the blockchain via RPC. We do this in order to collect raw blockchain data, which is then enhanced so as to provide more useful data to higher-level components such as the tokenlib; the customized, per-smart-contract libraries; and the various applications. It is the job of etherlib to provide faster access to the data, and many speed optimizations, in addition to a collection of easy-to-use interfaces for traversing the blocks, transactions, and accounts, are contained in this library.

4.3.3 tokenlib / walletlib

The final two pre-compiled libraries are called tokenlib and walletlib, which implement support for the Ethereum ERC 20 token standard and popular multi-sig wallets.

CHAPTER

FIVE

FAQ

5.1 How is QuickBlocks different than web3.js?

Two ways: (1) faster data, (2) better data.

While QuickBlocks uses the same technique to acquire the data from the Ethereum node (RPC interface), it does so with two significant improvements. First, QuickBlocks caches the data client side. This means that subsequent retrievals are significantly faster. The second improvement is that QuickBlocks can translate the data back into the language of your smart contract. Instead of **0xc9d27afe** which is what the RPC returns, QuickBlocks returns **vote** which is what that hex string means. The RPC returns blockchain-centric data. It cannot do anything else. It only has blockchain-centric data. By virtue of the ABI definition file, QuickBlocks is able to return data that looks like your smart contract. Additional benefits include QuickBlocks' ability to export its data to any format including comma-separated values (.csv), tab delimited text (.txt), or even database commands. Or, you can avoid the export function entirely and just wire up the code to feed a database directly. With a traditional web scale databases of every transactions on your smart contract (including those that ended in error), you can build a much richer, much more interactive website for your end users than people are building with web3.js.

5.2 Can I trust the data QuickBlocks produces?

QuickBlocks retrieves its data using the RPC, which is identical to the method that web3.js uses, so if you trust web3.js, you should trust QuickBlocks. The results are identical. We hear your objection: everyone uses web3.js – it's battle tested. This is why QuickBlocks includes, with each tool, a command line option called **–raw**. When asked for **–raw** output, QuickBlocks simply returns exactly the data returned by the RPC. Many QuickBlocks tools have an additional option called **–check**. **Check** takes the **–raw** data and compares it, data-field-by-data-field, with the QuickBlocks data. Finally, the portions of QuickBlocks that accumulate and produce the data are open source. If you don't believe us that it works as advertised, step through the code and convince yourself.

5.3 What is the distinction between a tool and an application?

The code for the tools is intended to serve as easy-to-grasp examples and use cases for the QuickBlocks libraries. The tools are open source and therefore, can be reviewed to see how they work and what they do. Most tools are written with only two files: a file named the same as the tool (getBlock.cpp for getBlock tool for example) and a file called options.cpp. Options.cpp handles all the command line parsing, so you can concentrate in each case on the other file. Most of them are very short. For example, here's the entirety of the getAccounts.cpp code:

```
#include "etherlib.h"
//--------------------------------------------------------------------------
int main(int argc, const char *argv[]) {

```

```
    // Tell the system where the blocks are and which version to use
    etherlib_init("fastest");

    SFAddressArray addrs;
    getAccounts(addrs);
    for (uint32_t i = 0 ; i < addrs.getCount() ; i++)
        cout << addrs[i] << "\n";
    return 0;
}
```

5.4 Are there any tutorials or samples?

Please see our Quick Start Guide for information on getting started.

5.5 More Questions. . .

Q: Do I need any API keys to run QuickBlocks?

QuickBlocks itself currently does not require any API keys (it runs locally after all), but one tool, called *ethslurp* does. (The reasons for this are long and varied, but let's just say "legacy".) You may get the API key you need for *ethslurp* at the website http://etherscan.io/api. Please follow the instructions there. Once you get a key, open a command line and type *ethslurp <address>* (replace <address> with any valid Ethereum address). The program will ask you for your key.

Q: May I use QuickBlocks on 32-bits machines?

We know of no reason why not, but we've done exactly zero testing on 32-bit machines. If you find it works or you fix anything related to this, please post a PR.

Q: What are the hardware requirements to run quickBlocks using a local node?

As of November, 2017 we are able to run both a full archive Parity node (started with *parity –tracing on –pruning archive*) on an Apple Macintosh laptop with a 1 TB internal SSD (where we store the entire Ethereum blockchain data) and a 1 TB Samsung external SSD (where we store QuickBlocks' caches). The blockchain data takes up about 400 GB, with occasional spikes up to around 900 GB. The QuickBlocks cache currently takes up about 500 GB with no spiking, but in the future, when we implement a future feature we call minimal impact, it will take up magnificently (three orders of magnitude?) less space.

Q: What operating systems are supported?

Mac OSX, Ubuntu, Fedora, Debian, CentOS, FreeBSD. Please see the installation instructions for more information. We currently do not have a Windows version, but if you want to make a pull request, we're all ears.

Q: Is the product open source or must I buy a license?

A little bit of both. The SDK libraries and the command-line tools are free and open source. You may use them as you wish. Other parts of QuickBlocks (the applications) must be licensed as either installed software (if you wish to run it locally) or via an annual subscription (where we deliver the data to your application via a web API). The first scenario imposes the burden on you to run a node but is more secure. The second scenario takes that burden off your hands, but is less secure (because what happens your business if we disappear?) More information will be available on our website shortly.

Q: How may I contact you?

Please visit our website for contact information.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- search