ORIGINAL BITCOIN 0.1.3 ALPHA

Reverse engineering the original C++ source code of Bitcoin Core 0.1.3 alpha release to understand how Bitcoin and the Blockchain works.

ABSTRACT

This document review the C++ source code of the original Bitcoin Core software.

Author:Jean Cavallera

How to understand Bitcoin Core?

You need to go back in time, and learn the technology like if you were back in the days (2009, 2010, 2011, ...). Imagine that you grow up as the years pass and the technology evolves.

Read the mail exchanges from the cryptography mailing list, and highlight with colours (implies to have the emails in PDF).

Read on the Bitcoin Talk forum the early posts, starting from the opening of the forum. Use the advanced search feature if there is one.

You can also look at the posts in the p2p foundation, where Bitcoin was marketed.

Based on original C++ source code available at: https://github.com/trottier/original-bitcoin/tree/master/src

Introduction

Bitcoin does not use encryption.

This is the content included in the README file below.

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)

Compilers Supported

MinGW GCC (v3.4.5)
Microsoft Visual C++ 6.0 SP6

Dependencies

```
              default path   download
wxWidgets      \wxWidgets     http://www.wxwidgets.org/downloads/
OpenSSL        \OpenSSL       http://www.openssl.org/source/
Berkeley DB    \DB            http://www.oracle.com/technology/software/products/berkeley-
db/index.html
Boost          \Boost         http://www.boost.org/users/download/
```

OpenSSL

According to the original README file, Bitcoin does not use any encryption. If you want to do a no-everything build of OpenSSL to exclude routines, a few patch are required (OpenSSL v0.9.8h)

```
Edit engines\e_gmp.c and put this #ifndef around #include <openssl/rsa.h>
  #ifndef OPENSSL_NO_RSA
  #include <openssl/rsa.h>
  #endif

Add this to crypto\err\err_all.c before the ERR_load_crypto_strings line:
  void ERR_load_RSA_strings(void) { }

Edit ms\mingw32.bat and replace the Configure line's parameters with this
no-everything list.  You have to put this in the batch file because batch
files can't handle more than 9 parameters.
  perl Configure mingw threads no-rc2 no-rc4 no-rc5 no-idea no-des no-bf no-cast no-aes no-
camellia no-seed no-rsa no-dh
```

```
Also REM out the following line in ms\mingw32.bat.  The build fails after it's
already finished building libeay32, which is all we care about, but the
failure aborts the script before it runs dllwrap to generate libeay32.dll.
  REM  if errorlevel 1 goto end

Build
  ms\mingw32.bat

If you want to use it with MSVC, generate the .lib file
  lib /machine:i386 /def:ms\libeay32.def /out:out\libeay32.lib
```

Software Directory Structure

The next pages go over the directory structure of Bitcoin Core.

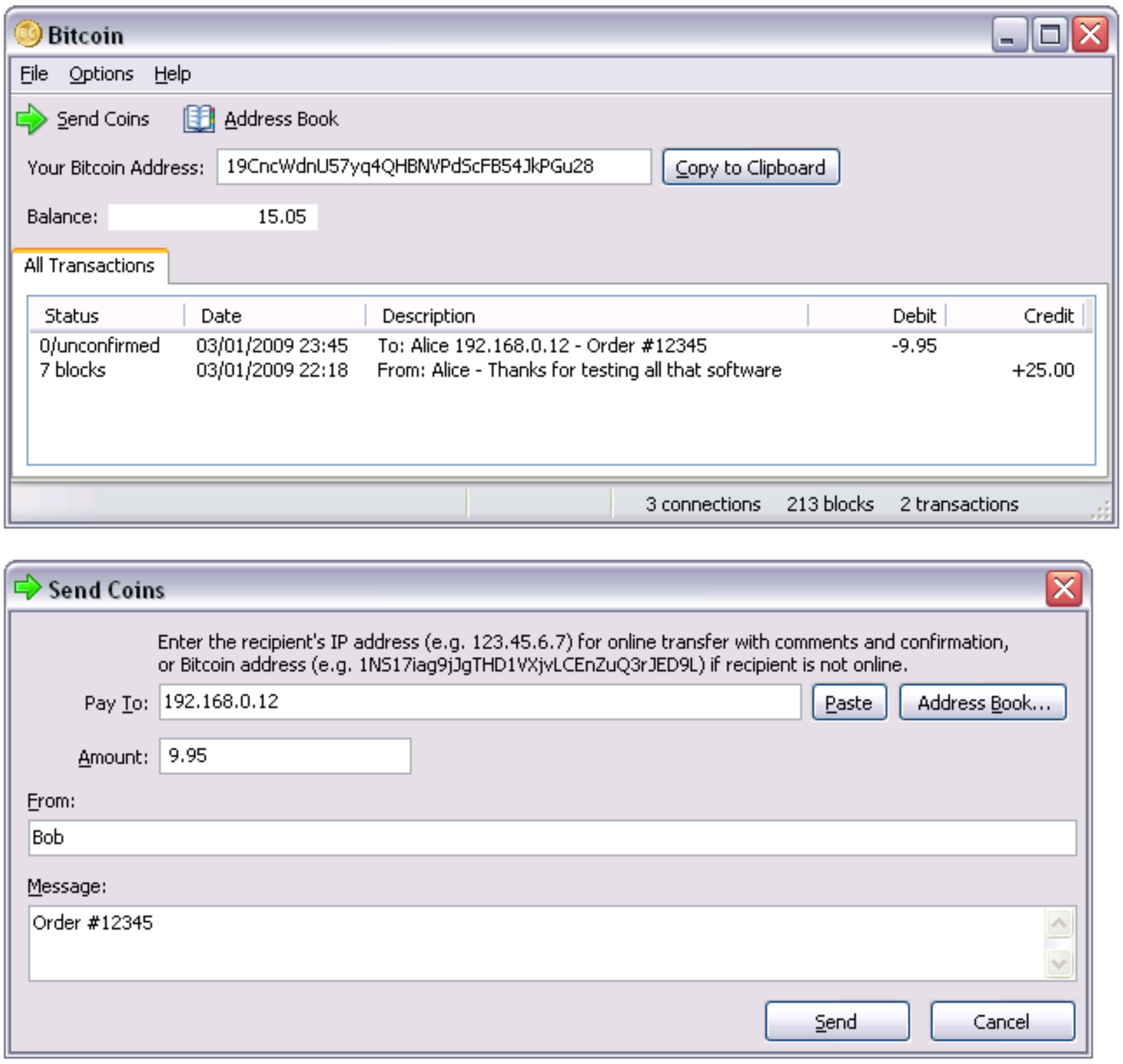
/src	
|--- /rc	
|--- addressbook16.bmp	
|--- addressbook16mask.bmp	
|--- addressbook20.bmp	
|--- addressbook20mask.bmp	
|--- bitcoin.ico	
|--- check.ico	
|--- send16.bmp	
|--- send16mask.bmp	
|--- send16masknoshadow.bmp	
|--- send20.bmp	
|--- send20mask.bmp	
|--- base58.h	
|--- bignum.h	
|--- db.cpp	
|--- db.h	
|--- headers.h	
|--- irc.cpp	
|--- irc.h	
|--- key.h	
|--- license.txt	
|--- main.cpp	

|--- main.h	
|--- makefile	
|--- makefile.vs	
|---market.cpp	
|--- market.h	
|--- net.cpp	
|--- net.h	
|--- readme.txt	
|--- script.cpp	
|--- script.h	
|--- serialize.h	
|--- sha.cpp	
|--- sha.h	
|--- ui.cpp	
|--- ui.h	
|--- ui.rc	
|--- uibase.cpp	
|--- uibase.h	
|--- uint256.h	
|--- uiproject.fbp	
|--- util.cpp	
|--- util.h	

.gitignore	
README.md	
Bitcoin.exe	Executable file. When you run it, it automatically connects to other nodes.
Libeay32.dll	
License.txt	
Mingwm10.dll	
Readme.txt	

The /rc folder

This folder only contains icons pictures in .ico or .bmp (bitmap) formats. These icons form part of the User Interface (UI) of the software, for instance the Bitcoin logo, the green arrow and the address book. You can see the UI of the previous software below:



Your Bitcoin Address: 19CncWdnU57yq4QHBNVPdScFB54JkPGu28

Balance: 15.05

Status	Date	Description	Debit	Credit
0/unconfirmed	03/01/2009 23:45	To: Alice 192.168.0.12 - Order #12345	-9.95	
7 blocks	03/01/2009 22:18	From: Alice - Thanks for testing all that software		+25.00

3 connections 213 blocks 2 transactions

Enter the recipient's IP address (e.g. 123.45.6.7) for online transfer with comments and confirmation, or Bitcoin address (e.g. 1NS17iag9jJgTHD1VXjvLCEnZuQ3rJED9L) if recipient is not online.

Pay To: 192.168.0.12

Amount: 9.95

From: Bob

Message: Order #12345

Figure 1: Original UI of the Bitcoin Core 0.1.3 alpha (retrieved from https://web.archive.org/web/20090303195936/http://bitcoin.org/)

The /src folder

This is where everything happens ☺ We will analyse file by file this folder. We can summarize by saying that there are C++ header files and standard C++ files. They are summarized below:

What are header files in C++ ?

C++ classes (and often function prototypes) are normally split up into two files. The **header file** has the extension of .h and contains class definitions and functions (just their declarations).. The implementation of the class goes into the .cpp file.

By doing this, if your class implementation doesn't change, then it won't need to be recompiled.

	Header file	Standard file	
	Base58.h		
done	Db.h	Db.cpp	
	Bignum.h		
	Headers.h		
done	Irc.h	Irc.cpp	
	Key.h		
Biggest file	Main.h	Main.cpp	
done	Market.h	Market.cpp	
done	Net.h	Net.cpp	
done	Script.h	Script.cpp	
	Serialize.h		
	Sha.h	Sha.cpp	
	Ui.h	Ui.cpp	
	Uibase.h	Uibase.cpp	
	Uint256.h		
	Util.h	Util.cpp	

The colours on the table describe the following:

- Standard files in yellow are the files that include the headers.h
- Standard files in orange are the files that include the sha.h

Database

This section corresponds to the files db.cpp and db.h. It contains the properties and methods mentioned in the table below.

DB.h

See the table in the next page.

Line number	**Class**	**properties**	**methods**	**Line number**	**Additional infos**
	CTransactions				
	CTxIndex				
	CDiskBlockIndex				
	CDiskTxPos				
	COutPoint				
	CUser				
	CReview				CReviewDB
	CAddress				CAddres
	CWalletTx				CWalletTx
	CDB		Read		
			Rite		
			Erase		
			Exit		
			GetCursor		
			ReadCursor		
			GetTxn		
			TxnBegin		
			TxnCommit		
			TxnAbort		
			ReadVersion		
			WriteVersion		
	CTxDB		Constructor		
			ReadTxIndex		
			UpdateTxIndex		
			AddTxIndex		
			EraseTxIndex		
			ContainsTx		
			ReadOwnerTxes		
			ReadDiskTx		Declared 4 times
			WriteBlokIndex		
			EraseBlockIndex		
			ReadHashBestChain		
			WriteHashBestChain		

			LoadBlockIndex		
Line 280	CReviewDB				
			Constructor		Public + private
			ReadUser		
			WriteUser		
			ReadReview		
			WriteReview		
	CMarketDB				Inherits CDB
			constructor		Public + private
Line 306	CAddrDB				Inherits CDB
			constructor		Public + private
			WriteAddress		
			LoadAddress		
337	CWalletDB				Inherits CDB
			Constructor		
			ReadName		
			WriteName		
			EraseName		
			ReadTx		
			WriteTx		
			EraseTx		
			ReadKey		
			WriteKey		
			ReadDefaultKey		
			WriteDefaultKey		
			ReadSetting		

			WriteSetting		
			LoadWallet		

There also contains a function DBFlush, that takes a bool parameter fshutdown.
There is also a function called SetAddressBookName

The database files contain **flags** : DB_SET, DB_SET_RANGE, DB_GET_BOTH, DB_GET_BOTH_RANGE, DB_DBT_MALLOC

DB.cpp

See the table in the next page.

Line number	**Class**	**properties**	**methods**	**Line number**	**Additional infos**
21	CDBInit				
39			CDB::CDB(...)		Implementation of the header file, line 33
114			CDB::Close()		Implementation of header file line 36 Returns void
132			DBFlush		Implementation of header file line 21 Returns void
170	CTxDB				Implementation of the header file lines 259 – 273
210			ReadOwnerTxes		Implements protocol rules below
322			LoadBlockIndex		protocol rules below Interesting parts line 352 - 367
306	CAddrDB				Implementations header line 319
			LoadAddresses		Most interesting, uses CRITICAL_BLOCK
475	CReviewDB				Header implementation line 480

480			ReadReviews		Comment "msvc workaround, just need to do anything with vReviews)
496	CWalletDB				Header implementation line 337
			LoadWallet		Only this method is implemented.

The file contains pszfile (database file name) and pszMode (can be created on read only).
The fiel contains CRITICAL_BLOCKS { … }

Also interesting values : DB_CREATE, DB_INIT_LOCK, DB_INIT_LOG, DB_INIT_MPOOL DB_INIT_TXN, DB_THREAD, DB_PRIVATE, DB_RECOVER

There is also a new function in this file: InsertBlockIndex() (line 302). It returns an existing block or create a new block in the database (ne CBlockIndex) (in which file this CBlockIndex is located?)

Protocol: Several functions from the CTxDB follow a specific set of protocol rules:

1. Getcursor()
2. Loop by reading next record (readAtCursor())
3. Unserizlize
4. Depending on the file:
    a. Read transaction (readFromDisk()) (for ReadOwnerTx())
    b. Construct block index object (for LoadBlockIndex()) (line 352 – 367) by using the InsertBlockIndex() function to add current, previous and next block index.

Line 366 -367 checks for genesis block and best block (not sure to understand exactly this line)

LoadAddresses method loades user provided addresses via fopen("addr.tx") and implements the protocol above

The strange part in the CWalletDB class is the fact that only the method LoadWallet is implemented. Again, this class use CRITICAL_BLOCK to then in step 4 print three things: PGenerateBitcoin, nTransactionFee, addrIncoming. The step 3 in the protocol 1) check if the strTyp is name or tx or key or setting then 2) debug print comment block (line 544 – 560)

For some reasons (and I don't know why) the LoadWallet method is also mentioned by itself on line 584 (as a function, not a method of the class CWalletDB). This fuction either setKeyUser or create a new keyUser and set as default key

Interesting comments

Line 458 – 460: Nakamoto mentions to not delete line 460, because there is a bug that manifests in the function mapAddresses.count in irc.cpp

Line 503 : /// todo: shouldn't we catch exceptions and try to recover and continue?

Line 524 – 252: taking advantage of the fact that pair serialization is just the two items serialized one after another. (**really interesting**).

Internet Relay Chat (IRC)

Bitcoin implements IRC in two files irc.h and irc.cpp

Irc.h

The file contains 3 internal linkages for non-constant globals.

```
extern bool RecvLine(SOCKEThSocket, string& strLine);
extern void ThreadIRCSeed(void* parg);
extern bool fRestartIRCSeed;
```

Irc.cpp

Include a #pragma pack(1) at the beginning **(what does that do?)**
The file also contains a struct that represents the irc address.

```
struct ircaddr
{
    int ip;
    short port;
};
```

The rest of the file simply contain functions described below

line	Function	
17	**EncodeAddr**	
27	**DecodeAddr**	
47	**Send**	
63	**RecvLine**	
99	**RecvLineIRC**	
122	**RecvUntil**	
139	**Wait**	Wait n seconds to reconnect
155	**ThreadIRCSeed**	

Moreover, the **main** function of this file uses wsdata, WSAStartup and WSACleanup. (what are these?)

Market

Market.h

The file contains the following 3 functions: AdvertInsert(), AdvertErase() and AddAtomsAndPropagate()

The classes below are defined. See table in the next page.

Line number	**Class**	**properties**	methods	Line number	Additional infos
	CUser				
		vAtomsIn			vector
		vAtomsNew			vector
		vAtomsOut			vector
		vLinksOut			vector
			SetNull		
			GetHash		
			GetAtomCount		
			AddAtom		
	CReview				
		nVersion			
		hashTo			
		mapValue			
		vchPubKeyFrom			
		vchSig			
		nTimes			Memory only
		nAtoms			Memory only
			GetHash		
			GetSigHash		
			GetUserHash		
			AcceptReview		
	CProduct				
		nVersion			
		Addr			
		mapValue			
		mapDetails			
		vOrderForm			
		nSequence			
		vchPubKeyFrom			
		vchSig			

		nAtoms			disk only
		setSources			Memory only
			Gethash		
			GetSigHash		
			GetUserHash		
			CheckSignature		
			CheckProduct		

This file contains a lot of comments mentioning 'memory only', what do they mean? There is also IMPLEMENT_SERIALIZE { ... } that I do not understand.

The file also contains 3 non constant variables at the end:

```
extern map<uint256, CProduct> mapProducts;
extern CCriticalSection cs_mapProducts;
extern map<uint256, CProduct> mapMyProducts;
```

Market.cpp

Implementation of AdvertInsert line 29

1. Insert or find existing product
2. Update If newer

Implementation of AdvertErase line 58

What does the **Union** function do ? (line 85)

Implementation of CUser:::AddAtom (line 109)

Implementation of AddAtomsAndPropagate (line 143)

Need to finish this file

Interesting comments in market.cpp file

Line 21: //// later figure out how these are persisted
```
map<uint256, CProduct> mapMyProducts;
```

Line 111 : // ignore duplicates

Line 116 - 119:

> //// instead of zero atom, should change to free atom that propagates, limited to lower than a certain value like 5, so conflicts quickly
>
> // the zero atom never propagate, new atom always propagate through the user that created them

Line 133: // Select Atom to flow through to vAtomsOut

Line 136: // Merge vAtomsNews in vAtomsIn

The Network

This file is probably the most interesting as it seems to relate to the p2p network.

Net.h

Line number	**Class**	**properties**	methods	Line number	Additional infos
	CMessageHeader				
		COMMAND_SIZE = 12			
		pchMessageStart			
		pchCommand			
		nMessageSize			
			GetCommand		
			IsValid		
	CAddress				
		nServices			
		pchReserved[12]			
		Ip			
		Port			
		nTime			
		nLastFailed			
			GetKey		
			GetSockAddr		
			IsIPv4		
			IsRoutable		
			GetByte		
			ToStringIPPort		
			ToStringIP		
			ToString		
			print		
	CInv				
		type			
		hash			
			IsKnownType		
			GetCommand		

			ToString		
			print		
392	CRequestTracker				
		Param1			
			Constructor		
			IsNull		
	CNode				
			constructor		
			ReadyToDisconnect		
			GetRefCount		
			AddRef		
			Release		
			AddInventoryKnown		
			PushInventory		
			AskFor		
			BeginMessage		
			AbortMessage		
			EndMessage		
			EndMessageAbortIfEmpty		
			GetMessageCommand		
			PushMessage		+ 4 times the same function with template
			PushRequest		+ 2 times the same function with template

			IsSubscribed		Not implemented
			Subscribe		Not implemented
			CancelSubscribe		Not implemented
			Disconnect		Not implemented

5 functions are defined in this file:

- ConnectSocket
- GetMyExternalIP
- AddAddress
- FindNode
- ConnectNode
- AbandonRequests
- AnySubscribed
- ThreadBitcoinMiner
- StartNode
- StopNode
- CheckForShutdown

The file also include other functions

- RelayMessage
- RelayInventory (put on the list to offer to other nodes)

The default port is set as **8333.** See line 13:

```
static const unsigned short DEFAULT_PORT = htons(8333);
```

Th GetSockAddr function is probably one of the most interesting one, because of the struct that it returns

```
 struct sockaddr_in GetSockAddr() const
    {
        struct sockaddr_in sockaddr;
        sockaddr.sin_family = AF_INET;
        sockaddr.sin_addr.s_addr = ip;
        sockaddr.sin_port = port;
        return sockaddr;
    }
```

The different messages types are defined as **enums** in this file, lines 301 – 308

```
enum
{
    MSG_TX = 1,
    MSG_BLOCK,
    MSG_REVIEW,
    MSG_PRODUCT,
    MSG_TABLE,
};
```

The file also contains a large set of **extern** declarations, from line 414 to 428

```
extern bool fClient;
extern uint64 nLocalServices;
extern CAddress addrLocalHost;
extern CNode* pnodeLocalHost;
extern bool fShutdown;
extern array<bool, 10> vfThreadRunning;
extern vector<CNode*> vNodes;
extern CCriticalSection cs_vNodes;
extern map<vector<unsigned char>, CAddress> mapAddresses;
extern CCriticalSection cs_mapAddresses;
extern map<CInv, CDataStream> mapRelay;
extern deque<pair<int64, CInv> > vRelayExpiration;
extern CCriticalSection cs_mapRelay;
extern map<CInv, int64> mapAlreadyAskedFor;
extern CAddress addrProxy;
```

The CNode class is the most important to understand. It contains a large set of properties, classified (via the comment) by *socket, flood, inventory based relay* and *publis and subscription.* The table below list them. Only nRefCounts is a protect method

// socket	// inventory based relay	// flood	// publish and subscription

uint64 nServices;
SOCKET hSocket;
CDataStream vSend;
CDataStream vRecv;
CCriticalSection cs_vSend;
	set<CInv> setInventoryKnown;
	set<CInv> setInventoryKnown2;
		vector<CAddress> vAddrToSend;
		set<CAddress> setAddrKnown;
			vector<char> vfSubscribe;

`CCriticalSection cs_vRecv;` `unsigned int nPushPos;` `CAddress addr;` `int nVersion;` `bool fClient;` `bool fInbound;` `bool fNetworkNode;` `bool fDisconnect;` `int64 nReleaseTime;` `map<uint256, CRequestTracker> mapRequests;` `CCriticalSection cs_mapRequests;`	`vector<CInv> vInventoryToSend;` `CCriticalSection cs_inventory;` `multimap<int64, CInv> mapAskFor;`		

The file also include a list of templates for the Publish and Subscription system. These are:

- AdvertStartPublish
- AdvertStopPublish
- AdvertRemoveSource

Interesting comments:

Line 51 – 53 : // the message start string is designed to be unlikely to occur in normal data. The characters are rarely used upper ascii, not valid as UTF-8, and produce a large 4 byte int at any alignment.

Line ? :

// we are using mapAskFor as apriority queue
// the key is the earliest time the request can be sent
// make sure to not reuse time indexes to keep things in the same order
// Each retry is 2min after the last

Line 784 to 797:

// Expire old relay messages
// save original serialize in newer versions are preserved

Net.cpp

This include the following

```
#include <winsock2.h>
```

The file contains the following list of global state variables

```
bool fClient = false;
uint64 nLocalServices = (fClient ? 0 : NODE_NETWORK);
CAddress addrLocalHost(0, DEFAULT_PORT, nLocalServices);
CNode nodeLocalHost(INVALID_SOCKET, CAddress("127.0.0.1", nLocalServices));
CNode* pnodeLocalHost = &nodeLocalHost;
bool fShutdown = false;
array<bool, 10> vfThreadRunning;
vector<CNode*> vNodes;
CCriticalSection cs_vNodes;
map<vector<unsigned char>, CAddress> mapAddresses;
CCriticalSection cs_mapAddresses;
map<CInv, CDataStream> mapRelay;
deque<pair<int64, CInv> > vRelayExpiration;
CCriticalSection cs_mapRelay;
map<CInv, int64> mapAlreadyAskedFor;
```

Here are the list of functions

ThreadMessageHandler2	
ThreadSocketHandler2	
ThreadOpenConnections2	
ConnectSocket	
GetMyExternalIP2	
GetMyExternalIP	
AddAddress	
AbandonRequests	
// Subscription methods for the broadcast and subscription system	
AnySubscribed	
CNode::IsSubscribed	
CNode::Subscribe	
CNode::CancelSubscribe	
FindNode	Mentioned x2
ConnectNode	
CNode::Disconnect	
ThreadSocketHandler	
ThreadSocketHandler2	Very long functions (lines 458 to 676)
ThreadOpenConnections	
ThreadOpenConnections2	(line 748)
ThreadMessageHandler	
ThreadMessageHandler2	
ThreadBitcoinMiner	
StartNode	
StopNode	
CheckForShutdown	

Interesting comments

Just above the list of subscriptions methods for the broadcast and subscription system, Nakamoto explains the following:

> // The subscription system uses a meet-in-the-middle strategy
> // with 100,000 nodes, if senders broadcast to 1,000 random nodes and reeivers subscribe to 1,000, 99.995 % (1 – 0.99 ^1000) of messages will get through

Line 302: For the CancelSubscribe method,

> // Prevent from relaying cancel if wasn't subscribed
> // relay subscription cancel
> // clear memory, no longer subscribed

Line 415: For the CNode::Disconnect method,

> // All of nodes broadcast and suscriptions are automatically torn down when it goes down
> // So a node has to stay up to keep its broadcast going.

Line 747 – 755: The function ThreadOpenConnections2 contains the following long comment:

```
//
// The IP selection process is designed to limit vulnerability to address flooding.
// Any class C (a.b.c.?) has an equal chance of being chosen, then an IP is
// chosen within the class C.  An attacker may be able to allocate many IPs, but
// they would normally be concentrated in blocks of class C's.  They can hog the
// attention within their class C, but not the whole IP address space overall.
// A lone node in a class C will get as much attention as someone holding all 255
// IPs in another class C.
//
```

The “Script” language in Bitcoin

Script.h

This file contains a class CTransaction

It also includes a first enum (line 7 – 13)...

```
enum
{
    SIGHASH_ALL = 1,
    SIGHASH_NONE = 2,
    SIGHASH_SINGLE = 3,
    SIGHASH_ANYONECANPAY = 0x80,
};
```

And antoher enum that contain all the opcode types (line 71 to 151). The **script opcodes** can be divided in the following categories:

- Push value
- Control
- Stack operations
- Splice operations
- Bit logic
- Numeric
- Crypto
- Multi-byte opcode

Template matching params

- Invalid opcode

It also contains the following functions

- GetOpName
- ValueString
- StackString

The class **CScript** contains the following functions:

- Push_int64
- Push_uint64
- Getop (defined x2)
- FindAndDelete
- PrintHex
- ToString
- Print

We then have 6 functions declarations with no implementations

- EvalScript
- SignatureHash
- IsMine
- ExtractPubKey
- ExtractHash160
- SignSignature
- VerifySignature

Script.cpp

The first function declared (but not implemented) is CheckSig.

Lines 12 to 18 defined a list of static constants

```
typedef vector<unsigned char> valtype;
static const valtype vchFalse(0);
static const valtype vchZero(0);
static const valtype vchTrue(1, 1);
static const CBigNum bnZero(0);
static const CBigNum bnOne(1);
static const CBigNum bnFalse(0);
static const CBigNum bnTrue(1);
```

We then have two functions : CastToBool and MakeSameSize.

Line 41 – 42 are interesting, but I don't understand these statements tho

```
#define stacktop(i)  (stack.at(stack.size()+(i)))
#define altstacktop(i)  (altstack.at(altstack.size()+(i)))
```

The rest of the file implements the functions from script (located at the bottom of the header file).

Additional functions are also included (see table below). These are interesting to read, as you can see through the comments the details of how it works.

Line number	**Function**	**Comment**
881	CheckSig	// Hash type is one byte tacked on to the end of the signature
913	Solver	
919		// Standard tx, sender provides pubkey, receiver adds signature
922		// short account number tx, sender provides hash of the pub key, receiver provides pub key and signature
975	Solver	
983		// compile solution
990		// sign
1005		// Sign and give pubkey

The functions SignSignature and EvalScript have several interesting comments (see below).

The SignatureHash function (line 818) Is very interesting and worth to understand correctly (although the function's body is really long and spans 58 lines).

Interesting comments

Line 38 – 39 :

// Script is a stack machine (like Forth) that evaluates a predicate returning a bool indicating valid or not. There are no loops.

Line 1097 – 1098 (for the function SignSignature)

```
  // Leave out the signature from the hash, since a signature can't sign itself.
      // The checksig op will also drop the signatures from its hash.
```

Big function EvalScript :

Line 494 – 498

```
  // OP_NOTEQUAL is disabled because it would be too easy to say
  // something like n != 1 and have some wiseguy pass in 1 with extra
  // zero bytes after it (numerically, 0x01 == 0x0001 == 0x000001)
  // if (opcode == OP_NOTEQUAL)
  //    fEqual = !fEqual;
```

Line 687 :

```
// Hash starts after the code separator
```

Line 706 and 752

```
// Subset of script starting at the most recent codeseparator
```

Line 709 and 755

```
// Drop the signature, since there's no way for a signature to sign itself
```

Line 777 – 778

```
  // If there are more signatures left than keys left,
  // then too many signatures have failed
```

The Blockchain

Below are the functions / methods that make up the main blockchain and design of the software. These are from the file *main.cpp*

The functions can be categorised as follow, based on the which dimension of the software they relate to:

- Keys & Wallet
- Transactions (including orphan transactions)
- Blocks and Chain logic
- Messages
- Mining
- Actions (get balance, etc...)

Keys & Wallet

Method	**Description**
AddKey	
GenerateNewKey	
AddToWallet	
AddToWalletIfMine	
EraseFromWallet	

Transactions

This include handling of orphans transactions

Method	**Description**
AddOrphanTx	
EraseOrphanTx	
IsMine	
GetDebit	
GetTxTime	
SetMerkleBranch	
AddSupportingTransaction	
AcceptTransaction	
AddToMemoryPool	
RemoveFromMemoryPool	
GetDepthInMainChain	
GetBlocksToMaturity	
AcceptTransaction	
AcceptWalletTransaction	
ReacceptWalletTransaction	
RelayWalletTransaction	
RelayWalletTransaction**s**	

Blocks & Chain logic

Method	**Description**
ReadFromDisk	
GetOrphanChain	
GetBlockValue	
GetNextWorkRequired	
DisconnectInputs	
ConnectInputs	
ClientConnectInput	
ClientConnectInputs	
DisconnectBlock	
ConnectBlock	
Reorganize	
AddToBlockIndex	
CheckBlock	
AcceptBlock	
ProcessBlock	
ScanMessageStart	
GetAppDir	
OpenBlockFile	*FILE*
AppendBlockFile	*FILE*
LoadBlockIndex	What are *scriptSig? ScriptPubKey? nValue? nBits?*
PrintBlockTree	*Try to run this function*

Messages

Method	**Description**
AlreadyHave	
ProcessMessage**s**	
ProcessMessage	
SendMessages	

Mining

Method	Description
FormatHashBlocks	
BlockSHA256	
BitcoinMiner	

Actions

Method	Description
GetBalance	
SelectCoins	
CreateTransaction	
CommitTransactionSpent	
SendMoney	

Current version of Bitcoin

In the latest version of Bitcoin Core (0.18.1), the only folder that contain the source code is /src (starting from the root folder).

We could also look at the other folders containing the files to build the binary (like Make files) for narrow specifications. But for understanding how the protocol has evolved, what is more important is the /src folder.

https://github.com/bitcoin/bitcoin

The genesis block is contained in the chainparams.cpp

References

These references are useful to understand Bitcoin. Below some archives of emails exchanged between Satoshi Nakamoto and other subscribers of the cryptography mailing list:

- https://pastebin.com/syrmi3ET