Block Spec for Bitcoin

Layout: Specification
Date: 2017-08-26
Activation: 1515888000
Version: 1.0

This section of the Bitcoin (BSV) specification ("spec") documents the **block data structure** for implementing a compatible BSV client, including the block header, block serialization, and coinbase transaction formats.

Block

A **block** is one of the two base primitives in the BSV system, the other being a **transaction**. Primitive in this context means it is one of the data structures for which the BSV software provides built-in support. Nodes collect new transactions into a **block**, hash them into a hash tree (**merkle root hash**), and scan through **nonce** values to make the block's hash satisfy proof-of-work requirements. When a miner solves the proof-of-work, it broadcasts the block to network nodes and if the block is valid it is added to the blockchain. The first transaction in the block is the **coinbase transaction** that creates a new coin owned by the creator of the block. An algorithm ensures that a new block is generated every 10 minutes (600 seconds) on average. The block validation rules described here ensure that BSV nodes stay in consensus with other nodes. There are several rules that must be respected for a block to be valid. A node is responsible for processing, validating, and relaying the block and its transactions. A node is distinct on the network from miners and wallets. A BSV node is a piece of software that connects to other nodes in a network and communicates via peer-to-peer messages. Nodes use the verack protocol to communicate and perform full validation checks, including:

1. Connecting to the network and peers.
2. Acquiring block headers.
3. Validating all blocks.
4. Validating all transactions.

Block Header

Block headers are serialized in the 80-byte format comprising six fields: version, previous block hash, merkle root hash, timestamp, difficulty target, and nonce. The block header is hashed as part of the proof-of-work algorithm, making the serialized header format part of the consensus rules. The hash of the block header is the unique signature of the block. The block header hash is included in the next block that is mined. The block header includes a pointer to the previous block that links them in the blockchain. The block header requires the following six fields. Note that the hashes are in internal byte order; all other values are in little-endian order.

Field	Size (bytes)	Data type	Description
nVersion	4	int32_t	The block version number indicates which set of block validation rules to follow.
hashPrevBlock	32	uint256	The SHA256(SHA256(Block_Header)) message digest of the previous block’s header.
hashMerkleRoot	32	uint256	The message digest of the Merkle root.
nTime	4	uint32_t	Current timestamp in seconds since 1970-01-01T00:00 UTC (Unix time).
nBits	4	uint32_t	Difficulty target for the proof-of-work for this block.
nNonce	4	uint32_t	32-bit number (starts at 0) used to generate this block (the "nonce").

Block Version

The block version number is a signed 4 byte integer (int32_t) that indicates which set of block validation rules to follow. BSV version >= 4 is valid.

Previous Block Hash

The SHA256(SHA256(Block_Header)) message digest (hash) of the previous block’s header in internal byte order. This ensures no previous block can be changed without also changing this block’s header.

Merkle Root Hash

The Merkle tree is data structure that provides a record of all transactions in the block. Each transaction in the block is a leaf in the Merkle tree and includes a hash of the previous transaction hash. The Merkle root is derived from the hashes of all transactions included in this block. The hash of the Merkle root ensures that no transaction can be modified without modifying the block header.

The Merkle root is constructed from the list of transaction IDs in the order the transactions appear in the block.

- The coinbase transaction TXID is always placed first.
- Any input within this block can spend an output which also appears in this block (assuming the spend is otherwise valid). However, the TXID corresponding to the output must be placed at some point before the TXID corresponding to the input. This ensures that any program parsing block chain transactions linearly will encounter each output before it is used as an input.

If a block only has a coinbase transaction, the coinbase TXID is used as the Merkle root hash.

If a block only has a coinbase transaction and one other transaction, the TXIDs of those two transactions are placed in order, concatenated as 64 raw bytes, and then SHA256(SHA256()) hashed together to form the Merkle root.

If a block has three or more transactions, intermediate Merkle tree rows are formed. The TXIDs are placed in order and paired, starting with the coinbase transaction's TXID. Each pair is concatenated together as 64 raw bytes and SHA256(SHA256()) hashed to form a second row of hashes. If there are an odd (non-even) number of TXIDs, the last TXID is concatenated with a copy of itself and hashed. If there are more than two hashes in the second row, the process is repeated to create a third row (and, if necessary, repeated further to create additional rows). Once a row is obtained with only two hashes, those hashes are concatenated and hashed to produce the Merkle root.

TXIDs and intermediate hashes are always in internal byte order when they're concatenated, and the resulting Merkle root is also in internal byte order when it's placed in the block header.

Note that the Merkle root makes it possible in the future to securely verify that a transaction has been accepted by the network using just the block header (which includes the Merkle tree), eliminating the current requirement to download the entire blockchain.

Block Timestamp

The block timestamp is Unix epoch time when the miner started hashing the header according to the miner's clock. The block timestamp must be greater than the median time of the previous 11 blocks. Note that when validating the first 11 blocks of the chain, you will need to know how to handle arrays of less than length 11 to get a median. A node will not accept a block with a timestamp more than 2 hours ahead of its view of network-adjusted time.

Difficulty Target

The difficulty target is a 256-bit unsigned integer which a header hash must be less than or

equal to for that header to be a valid part of the block chain. The header field *nBits* provides only 32 bits of space, so the target number uses a less precise format called "compact" which works like a base-256 version of scientific notation. As a base-256 number, nBits can be parsed as bytes the same way you might parse a decimal number in base-10 scientific notation.

Although the target threshold should be an unsigned integer, the class from which the original nBits implementation inherits properties from a signed data class, allowing the target threshold to be negative if the high bit of the significand is set.

- When parsing nBits, the system converts a negative target threshold into a target of zero, which the header hash can equal (in theory, at least).
- When creating a value for nBits, the system checks to see if it will produce an nBits which will be interpreted as negative; if so, it divides the significand by 256 and increases the exponent by 1 to produce the same number with a different encoding. Difficulty is a measure of how difficult it is to find a hash below a given target. The BSV network has a global block difficulty. Valid blocks must have a hash below the difficulty target calculated from the nBits value. The current difficulty target is available here: https://blockexplorer.com/api/status?q=getDifficulty.

Nonce

To be valid, a block include a **nonce** value that is the solution to the mining process. This proof-of-work is verified by other BSV nodes each time they receive a block.

NOTE: The original purpose of the nonce was to manipulate it to find a solution to the mining process. However, because mining devices now have hashrates in the terahash range, the nonce field is too small. In practice, most block headers do not include a solution to the mining process in the nonce. Instead, miners have try many different Merkle root hashes, which is typically done by adding transactions or changing the coinbase TX. A nonce value is nonetheless required.

The nonce is a 32-bit (4-byte) field whose value is arbitrarily set by miners to modify the header hash and produce a hash that is less than the difficulty target with the required number of leading zeros (currently 32) satisfies the proof-of-work.

An arbitrary number miners change to modify the header hash in order to produce a hash less than or equal to the target threshold. If all 32-bit values are tested, the time can be updated or the coinbase transaction can be changed and the Merkle root updated.

The nonce is an arbitrarily changed by miners to modify the header hash and produce a hash less than the difficulty target. If all 32-bit values are tested, the time can be updated or the coinbase transaction can be changed and the Merkle root updated.

Any change to the nonce will make the block header hash completely different. Since it is virtually impossible to predict which combination of bits will result in the right hash, many different nonce values are tried, and the hash is recomputed for each value until a hash containing the required number of zero bits as set by the difficulty target is found. The resulting hash has to be a value less than the current difficulty and so will have to have a certain number of leading zero bits to be less than that. As this iterative calculation requires time and resources, the presentation of the block with the correct nonce value constitutes

proof-of-work.

It is important to note that the proof-of-work can be verified by computing one hash with the proper content, and is therefore very cheap. The fact that the proof is cheap to verify is as important as the fact that it is expensive to compute.

Coinbase Transaction

The first transaction in the body of each block is a special transaction called the **coinbase transaction** which is used to pay miners of the block. The coinbase transaction is required, and must collect and spend any transaction fees paid by transactions included in the block.

A valid block is entitled to receive a block subsidy of newly created bitcoincash value, and it must also be spent in the coinbase transaction. Together, the transaction fees and block subsidy are called the **block reward**. A coinbase transaction is invalid if it tries to spend more value than is available from the block reward. The subsidy plus fees is the maximum coinbase payout, but note that it is valid for the coinbase to pay less.

The coinbase transaction must have one input spending from 000000000000000. The field used to provide the signature can contain arbitrary data up to 100 bytes. The coinbase transaction must start with the block height to ensure no two coinbase transactions have the same transaction id (TXID).

The coinbase transaction has the following format:

Bytes	Name	Data type	Description
32	hash (null)	char[32]	A 32-byte null, as a coinbase has no previous outpoint.
4	index (UINT32_MAX)	uint32_t	0xffffffff, as a coinbase has no previous outpoint.
Varies	script bytes	compactSize uint	The number of bytes in the coinbase script, up to a maximum of 100 bytes.
Varies (4)	height	script	The block height of this block. Required parameter. Uses the script language: starts with a data-pushing opcode that indicates how many bytes to push to the stack followed by the block height as a little-endian unsigned integer. This script must be as short as possible, otherwise it may be rejected. The data-pushing opcode is 0x03 and the total size is four bytes.
4	sequence	uint32_t	Sequence number.

Although the coinbase script is arbitrary data, if it includes the bytes used by any signature-checking operations such as OP_CHECKSIG, those signature checks will be counted as signature operations (sigops) towards the block's sigop limit. To avoid this, you

can prefix all data with the appropriate push operation. See Transaction format for details on opcodes.

Block Serialization

Blocks must be serialized in binary format for transport on the network. Under current BSV consensus rules, a BSV block is valid if its serialized size is not more than 32MB (32,000,000 bytes). All fields described below count towards the serialized size limit.

Bytes	Name	Data type	Description
80	block header	block_header	The block header in the proper format. See Block Header.
Varies	txn_count	compactSize uint	Total number of transactions in this block, including the coinbase transaction.
Varies	txns	raw transaction	Each transaction in this block in this block, one after another, in raw transaction format. Transactions must appear in the data stream in the same order their TXIDs appeared in the first row of the Merkle tree.

The serialized (raw) form of each block header is hashed as part of the proof-of-work, making the serialized block header part of the BSV consensus rules. As part of the mining process, the block header is hashed repeatedly to create proof-of-work. BSV uses SHA256(SHA256(Block_Header)) to hash the block header. You must ensure that the block header is in the proper byte-order before hashing. The following serialization rules apply to the block header:

- Both hash fields use double-hashing (SHA256(SHA256(DATA))) and are serialized in internal byte order, which means the standard order in which hash message digests are displayed as strings.
- The values for all other fields in the block header are serialized in little-endian order. Note that when displayed via a block browser or query, the ordering is big-endian.