Genesis Upgrade Specification

Version: 2019-12-17
Status: Draft
Authors: nChain Ltd

Genesis Upgrade Specification

Version: 2019-12-17
Status: Draft
Authors: nChain Ltd

Introduction **2**

Definitions **2**

Upgrade Activation Mechanism **3**

UTXO Height Rule Determination **3**

Changes to the Bitcoin Specification **4**

Block Consensus Rules 4

Block Size Consensus Rule 4

Number of CheckSig Operations per MB of Block Space 4

Transaction Consensus Rules 4

Introduction

This is the specification of the Genesis Upgrade. It defines the upgrade activation mechanism, the changes to the Bitcoin Specification and the Consensus Rules, as well as the Standard Policies that are recommended for client implementations.

This document is a specification. It is not a guide and it does not describe implementation specifics or emergent behaviour of the system.

Definitions

The **Bitcoin Rules** are the precise rules which define Bitcoin. These include rules such as: the sum of the value of the inputs of a transaction must be greater than or equal to the sum of the values of the outputs, and the block subsidy schedule. These are the foundational rules of Bitcoin, irrespective of implementation.

The **Consensus Rules** are additional rules that have been reached by general agreement and are necessary to implement the Bitcoin Specification in software. They are needed because software has limitations and agreements are needed in order to facilitate integration of the software. Some Consensus Rules are configurable in the software, some are not. An example of a non-configurable Consensus Rule is that the version of a transaction must fit in a signed 32-bit integer. An example of a configurable Consensus Rule, after Genesis activation, is the maximum accepted block size.

Local Policies are additional controls that are implemented in software and may place additional restrictions on the transactions and blocks that the software will propagate to other systems and, for transactions, confirm in blocks. Policies are "local", they apply to the instance of software that is running, they do not apply to the validation of blocks, or the transactions within a block. A block accepted from another miner may contain transactions that do not conform to local policy.

Standard Policies are common local policies that are used by client software. They are defined as a Standard to facilitate common application across independent software implementations but it is important to note that it is not required that software implement or adhere to these policies.

Throughout this specification the following words are used with specific meanings:

- **valid** - a transaction or block is valid if it follows the Bitcoin and Consensus rules.
- **invalid** - a transaction or block is invalid if it does not follow either the Bitcoin or Consensus rules.
- **rejected** - a transaction or block may be rejected by an implementation due to a Policy. The implementation may not have determined whether the transaction or block was valid or invalid.

Throughout this specification the metric system is used. One KB is one kilobyte which is 1,000 bytes.

Upgrade Activation Mechanism

The Genesis Upgrade will activate at the block heights defined in the table below.

Mainnet	620,538
Testnet	1,344,302
Scaling Test Network	14,896
Regtest	10,000

The upgraded Bitcoin Rules, Consensus Rules, and Policies become active in the block that is mined at this height.

UTXO Height Rule Determination

The Genesis Upgrade introduces the “UTXO Height” mechanism for determining the Bitcoin Rules, Consensus Rules, and Policies that are to be applied to a transaction. The purpose of this mechanism is to maintain compatibility for UTXO’s that were created under the previous version of the Bitcoin Rules and Consensus Rules.

Every output of a transaction is evaluated under the rules that are in effect for the block in which the transaction is being confirmed. If a transaction is confirmed in a block that has a height which is greater or equal to the Genesis activation height, then the outputs of that transaction must comply with the Genesis Upgrade. If a transaction is confirmed in a block that has a height which is less than the Genesis activation height, then the outputs of that transaction must comply with the rules prior to the Genesis Upgrade.

Every input of a transaction is evaluated under either the rules prior to the Genesis Upgrade, or the Genesis Upgrade rules, depending on whether the UTXO that is being spent by the input was, or will be, confirmed prior to the Genesis activation height or after the Genesis activation height. If the transaction which contains the UTXO that is being spent was, or will be, confirmed in a block before the Genesis activation height then the input script and the output script for the UTXO being spent by that input are evaluated according to rules prior to the Genesis Upgrade. If the transaction which contains the UTXO that is being spent was, or will be, confirmed in a block with a height greater than or equal to the Genesis activation height, then the input script and the output script for the UTXO being spent by that input are evaluated according to the Genesis Upgrade.

Other characteristics of the transaction, such as the maximum number of sigops, are evaluated according to the rules that are in effect for the block in which the transaction is being confirmed.

It is worth noting that, after Genesis activation, a single transaction may spend inputs from before Genesis activation and after Genesis activation, in which case each input is evaluated under the rules that are applicable to that input based upon the block height that the transaction referenced by that input was confirmed.

Changes to the Bitcoin Specification

Block Consensus Rules

These consensus rules apply to blocks that are produced after Genesis activation.

Block Size Consensus Rule

The size of a block is the size in bytes of the serialized form of the block, including the block header and all of the transactions confirmed by the block[1].

[1] Note that the size of a block does not include the size of any messaging envelope that may encapsulate the block, such as the message header in the P2P networking standard.

The consensus rule that restricts the maximum size of a block to a specific number of bytes has been converted into a configurable consensus rule. The default value of the maximum acceptable size of a block is unlimited.

Miners are expected to reach consensus on this value and configure it manually.

Number of CheckSig Operations per MB of Block Space

The consensus rule that limits the number of checksig operations per megabyte of block space has been removed.

Transaction Consensus Rules

These consensus rules apply to transactions that are confirmed in blocks after Genesis activation.

Maximum Transaction Size

The size of a transaction is the size in bytes of the serialized form of the transaction[2].

The maximum size of a transaction is 1GB (1,000,000,000 bytes). This limitation is expected to be lifted in the future.

Maximum Number of CheckSig Operations per Transaction

The consensus rule that limits the number of checksig operations per transaction has been removed.

Signature Hashing Algorithm

The signature hashing algorithm for UTXO's created after the Genesis Upgrade remains the same as it was prior to the Genesis Upgrade. This is the signature hashing algorithm that was introduced on the Bitcoin Cash blockchain during the BTC/BCH split on the 1st August 2017. The signature hashing algorithm is described in requirement 6-2 of the specification.

After the Genesis activation, the original signature hashing algorithm, which is still in use on the BTC blockchain, is valid for outputs created before the Genesis activation.

nLockTime & nSequence

After Genesis activation, the functionality of the nLockTime field of a transaction and the nSequence fields of transaction inputs revert to their original purpose. The rules defined here only apply to transactions that are confirmed after Genesis activation.

The nSequence fields of every transaction input and the nLockTime field of the transaction collectively determine the "finality" of a transaction. If a transaction is "non-final" then it can not be valid but it can become "final" at a later time. If a transaction is "final" then it can be valid.

- If the value of nSequence of a transaction input is 0xFFFFFFFF then that input is a "final input".

[2] Footnote 1 also applies to transactions.

- If the value of nSequence of a transaction input is not 0xFFFFFFFF then that input is a “non-final input”.
- If all of the inputs of a transaction are “final inputs” then the transaction is “final”, irrespective of the value of the nLockTime field.
- If one or more of the inputs of a transaction are “non-final inputs” then:
    - If the value of the transactions nLockTime field is less than 500,000,000 then the field represents a block height.
        - If the height of the block in which the transaction is being confirmed is greater or equal to the value of this field, then the transaction is “final”.
        - Otherwise the transaction is “non-final”
    - If the value of the transactions nLockTime field is greater or equal to 500,000,000 then the field represents a UNIX epoch timestamp.
        - If the MTP of the last 11 blocks is greater or equal to the value of this field, then the transaction is “final”.
        - Otherwise, the transaction is “non-final”.

A “final” transaction may be confirmed in a block following the “first-seen” rule.

If a new transaction is detected which has the same inputs as a “non-final” transaction, and the sequence numbers of all inputs are greater or equal to the corresponding sequence numbers of the prior transaction, then this new transaction must replace the prior transaction.

If a new transaction is detected which has the same inputs as a “non-final” transaction, and the sequence numbers of the inputs are not all greater or equal to the corresponding sequence numbers of the prior transaction, then this new transaction must be rejected and the prior transaction kept.

If a new transaction is detected which has inputs that conflict with the inputs of a “non-final” transaction, but which are not identical to the inputs of the “non-final” transaction, then the “non-final” transaction is the “first seen” transaction and takes priority over the new transaction.

Script Language Rules

Bitcoin Script is the programming language that is used to lock and unlock transaction outputs. This section contains changes to the Script Language Rules.

The rules defined here apply to locking and unlocking scripts for transactions outputs that are created after the Genesis activation. The previous rules apply to transaction outputs created prior to Genesis activation.

Data Types

All data items in Bitcoin Script are a byte sequence. Some operations interpret their parameters as numeric or boolean values and require the item to fulfill the requirements of those types. Some operations produce items on the stack which are valid numeric or boolean values.

A byte sequence has a length and a value. The length of the byte sequence must be an integer greater or equal to zero and less than or equal to 2^32-1 (MAX_UINT32).

The byte sequence of length zero is called the “null value”.

Any data item can be interpreted as a boolean value. If the data item consists entirely of bytes with value zero, or the data item is the null value, then the boolean value of the item is **false**. Otherwise, the boolean value of the item is **true**.

A data item can be interpreted as a numeric value. The numeric value is encoded in a byte sequence using little-endian notation. The byte sequence of length zero (the null value) is the standard representation for the numeric value zero. There is a Consensus Rule on the maximum size of a numeric value that is defined below.

Formal Grammar for Bitcoin Script

A formal grammar has been defined for Bitcoin Script. This section of the document defines that grammar but it does not describe how this definition is applied. The application of the grammar definition is described in further parts of this document.

This grammar is fully described by the following BNF grammar:

```
<script> ::= <unlocking-script> <locking-script>

<unlocking-script> ::= <constants> | <empty>

<locking-script> ::= <script-block> [“OP_RETURN” <non-script-data>]

<script-block> ::= <function> [<script-block>] |

                <push-data> [<script-block>] |

                <branch> [<script-block>] |

                “OP_RETURN” [<script-block>] |

                <empty>

<branch> ::= <op-if> <script-block> [“OP_ELSE” <script-block>]

                “OP_ENDIF”

<op-if> ::= “OP_IF” | “OP_NOTIF”

<constants> ::= <constant> [<constants>]

<constant> ::= “OP_PUSHDATA1” <count-1> “bytes” |

                     “OP_PUSHDATA2” <count-2> “bytes” |

                     “OP_PUSHDATA4” <count-4> “bytes” |

                     <op-push-xx> “bytes” | <op-false> |

                     <op-true> | “OP_2” | “OP_3” | “OP_4” |

                     “OP_5” | “OP_6” | “OP_7” | “OP_8” | “OP_9” |

                     “OP_10” | “OP_11” | “OP_12” | “OP_13” |
                    "OP_14" | "OP_15" | "OP_16"

<count-1> ::= a one byte unsigned integer (0-255)

<count-2> ::= a two byte unsigned integer (0-65535)

<count-4> ::= a four byte unsigned integer (0-4294967295)

<op-push-xx> ::= hex codes 0x01 to 0x4b

<op-false> ::= "OP_0" | "OP_FALSE"

<op-true> ::= "OP_1" | "OP_TRUE"

<function> ::= "OP_NOP" | "OP_VERIFY" |

           "OP_TOALTSTACK" | "OP_FROMALTSTACK" | "OF_IFDUP" |

           "OP_DEPTH" | "OP_DROP" | "OP_DUP" | "OP_NIP" | "OP_OVER" |

           "OP_PICK" | "OP_ROLL" | "OP_ROT" | "OP_SWAP" | "OP_TUCK" |

           "OP_2DROP" | "OP_2DUP" | "OP_3DUP" | "OP_2OVER" |

           "OP_2ROT" | "OP_2SWAP" | "OP_SIZE" | "OP_EQUAL" |

           "OP_EQUALVERIFY" | "OP_1ADD" | "OP_1SUB" | "OP_NEGATE" |

           "OP_ABS" | "OP_NOT" | "OP_0NOTEQUAL" | "OP_ADD" |

           "OP_SUB" | "OP_BOOLAND" | "OP_BOOLOR" | "OP_NUMEQUAL" |

           "OP_NUMEQUALVERIFY" | "OP_NUMNOTEQUAL" | "OP_LESSTHAN" |

           "OP_GREATERTHAN" | "OP_LESSTHANOREQUAL" |

           "OP_GREATERTHANOREQUAL" | "OP_MIN" | "OP_MAX" |

           "OP_WITHIN" | "OP_RIPEMD160" | "OP_SHA1" | "OP_SHA256" |

           "OP_HASH160" | "OP_HASH256" | "OP_CODESEPARATOR" |

           "OP_CHECKSIG" | "OP_CHECKSIGVERIFY" | "OP_CHECKMULTISIG" |

           "OP_CHECKMULTISIGVERIFY" | "OP_CAT" | "OP_SPLIT" |

           "OP_AND" | "OP_OR" | "OP_XOR" | "OP_DIV" | "OP_MOD" |

           "OP_NUM2BIN" | "OP_BIN2NUM" | "OP_NOP1" | "OP_NOP2" |

           "OP_NOP3" | "OP_NOP4" | "OP_NOP5" | "OP_NOP6" |

           "OP_NOP7" | "OP_NOP8" | "OP_NOP9" | "OP_NOP10" |

           "OP_MUL" | "OP_LSHIFT" | "OP_RSHIFT" | "OP_INVERT" |
          "OP_2MUL" | "OP_2DIV" | "OP_VERIF" | "OP_VERNOTIF"

<non-script-data> ::= any sequence of bytes
```

It’s worth highlighting the following features of this formal grammar:

- The complete script consists of two sections, the unlocking script and the locking script. The locking script is included in the transaction output that is being spent, the unlocking script is included in the transaction input that is spending the output.
- The unlocking script can only contain constants. This requirement is part of Validity of Script Consensus Rule, defined later.
- A branching operator (OP_IF or OP_NOTIF) must have a matching OP_ENDIF.
- An OP_ELSE can only be included between a branching operator and OP_ENDIF pair. There can only be at most one OP_ELSE between a branching operator and an OP_ENDIF.
- OP_RETURN may appear at any location in a valid script. The functionality of OP_RETURN has been restored and is defined later in the section OP_RETURN Functionality. Grammatically, any bytes after an OP_RETURN that is not in a branch block are not evaluated and there are no grammatical requirements for those bytes.
- Note that disabled operations are included in this grammar. A disabled operation is grammatically correct but will produce a failure if executed.

Validity of Script Consensus Rule

For transactions that are created after Genesis activation, the transaction can only be valid if all of the locking scripts that it contains are grammatically valid, as defined by the formal grammar above.

For transactions that are created after Genesis activation and for each input that is spending an output that was confirmed after Genesis activation, the unlocking script must be grammatically valid, as defined by the formal grammar above.

Note that these constraints, including the formal syntax definition, do not apply to the scripts that are spending outputs created prior to Genesis activation.

Disabled Operations Consensus Rule

The following operations are disabled: OP_2MUL, OP_2DIV, OP_VERIF, OP_VERNOTIF.

Although these operations are grammatically correct and can appear in valid Script, their execution will cause script failure and therefore transaction invalidity. If they are present in un-executed script, the script execution may succeed.

It is very strongly recommended that these operations not be used at all until they are activated.

Script Element Size Consensus Rule

The consensus rule that limits the maximum size of a script element has been removed.

The functionality previously provided by this consensus rule is now covered by the Stack Memory Usage Consensus Rule.

Stack Size Consensus Rule

The consensus rule that limits the combined number of elements that can be placed on the stack, and the altstack, has been removed.

The functionality previously provided by this consensus rule is now covered by the Stack Memory Usage Consensus Rule.

Script Size Consensus Rule

The consensus rule that limits the maximum size of a script has been removed.

The functionality previously provided by this consensus rule is now covered by the Maximum Transaction Size Consensus Rule and the Stack Memory Usage Consensus Rule.

Numeric Value Size Consensus Rule

For a byte sequence to validly represent a numeric value, the length of the byte sequence must be less than or equal to 750,000 bytes. A byte sequence that is larger than this is a valid byte sequence but is not a valid numeric value.

Note that while some operations require parameters to be valid numeric values, they may produce byte sequences which are not valid numeric values (for example, OP_MUL may produce a byte sequence which is too large to validly represent a numeric value).

Number of Public Keys per Multisig Consensus Rule

The consensus rule that limits the number of public keys per multisig has been changed to be 2^32-1 (MAX_UINT32).

Number of Non-Push Operations Per Script Consensus Rule

The consensus rule that limits the number of non-push operations per script has been removed.

The functionality previously provided by this consensus rule is now covered by the Stack Memory Usage Consensus Rule while other capabilities manage the cost of execution.

Stack Memory Usage Consensus Rule

The stack memory usage consensus rule limits the amount of memory that can be used on the stacks. This rule is evaluated against the sum of the memory used by the stack and the memory used by the altstack.

A transaction which uses more stack memory than defined in this rule will be invalid.

The memory usage of a stack is calculated using a specific formula, so that it can be coordinated across software implementations. The formula for calculating the memory usage of a stack is:

```
usage = sum of (for each element: 32 + size in bytes of element)
```

where “`size in bytes of element`” is the size in bytes of the element when serialized in the Bitcoin Serialization Format.

This is a configurable consensus rule, with a default value that is formally unlimited but will in practice depend on the capabilities of the system that is evaluating the script.

Miners are expected to reach consensus on this value and configure it manually.

Sunset P2SH

Pay to script hash (P2SH) is a capability that was added to Bitcoin in 2012 by BIP13. It assigns a special meaning to a specific output script template. If the specific script template is present in the script of an output that is being spent by an input then it is treated in a different way, rather than being executed normally.

This feature is being removed by the Genesis Upgrade. The P2SH script template will not be treated “specially” for outputs created after Genesis activation but will be evaluated normally.

The P2SH feature will remain valid for outputs created prior to Genesis activation.

OP_RETURN Functionality

The functionality of the OP_RETURN operation is being restored for outputs created after Genesis activation.

For outputs created prior to Genesis activation, OP_RETURN causes immediate failure of the script.

For outputs created after Genesis activation, OP_RETURN will cause termination of the script and the validity of the script is determined by the value of the top item on the stack.

Sunset OP_CHECKLOCKTIMEVERIFY & OP_CHECKSEQUENCEVERIFY

OP_CHECKLOCKTIMEVERIFY and OP_CHECKSEQUENCEVERIFY are operations that were introduced by BIP0065 and BIP0112.

These operations remain effective for outputs created prior to Genesis activation.

For outputs created after Genesis activation, these operations revert to NOP’s, which have no effect, according to the consensus rules.

However, most implementations have a local policy which rejects scripts that contain “upgradeable NOPs”, including these, and will reject transactions containing these operations.

Standard Policies

Policies are settings that are configured by software operators. These settings are generally required by software implementations. It is expected that as implementations of the software mature that these settings will be adjusted.

Policies control which transactions the software will propagate across the P2P network or include in a block. However, policies are not Bitcoin Rules or Consensus Rules and are not used to determine the validity of blocks or the transactions confirmed by a block.

Transaction Policies

Maximum Acceptable Transaction Size Policy

The maximum acceptable transaction size is a standard policy that configures the largest transactions that the software will propagate across the P2P network or include in a block.

The value of this configuration item must be less than or equal to the value of the Maximum Transaction Size Consensus Rule or it will have no effect

The default value for this policy setting is 10MB (10,000,000 bytes).

Transaction Evaluation Timeout

The transaction evaluation timeout is a standard policy that defines the maximum amount of time that the software will allow for the evaluation of a transaction before terminating the evaluation and rejecting the transaction. This setting is always defined with a time unit and the default value is 1 second.

This policy applies to transactions that are received by the software and its processes for determining whether the transaction should be propagated across the P2P network or included in a block. The policy does not apply to the evaluation of transactions in a confirmed block.

Note that:

- if the evaluation of a transaction is terminated due to exceeding the timeout limit then the software has not determined whether the transaction is valid or invalid, it has merely decided not to make that determination;
- the time taken to evaluate a transaction by a particular system will vary based on the software used, the resources available, and other dynamic factors such as the load on the system. It is not an exact measurement.

Script Language Policies

Script Language Policies apply to the execution of script.

It must be noted that there are existing policies regarding the representation of numeric values in byte sequences, in particular the minimal encoding policy. These policies have not been changed as part of the Genesis Upgrade and are not documented here.

Numeric Value Length

The length of numeric value policy defines the maximum length of a byte sequence to be considered a valid numeric value. The default value for this policy is 250,000 bytes.

Transactions which contain script that consume numeric values that are larger than this policy setting will be rejected; they will not be propagated across the P2P network or included in a block.

Stack Memory Usage Policy

The stack memory usage policy limits the amount of memory that can be used on the stacks. This policy is evaluated against the sum of the memory used by the stack and the memory used by the altstack.

A transaction which uses more stack memory than defined in this policy will be rejected.

The memory usage of a stack is calculated using the same formula described in the Stack Memory Usage Consensus Rule (above).

The default value for this policy is 100MB (100,000,000 bytes). The value of this policy must be kept less than or equal to the value of the Stack Memory Usage Consensus Rule.