Fabric's block consists of three segments which are Header, Data and Metadata. In part 1, we dived into the content of Header and Metadata segment of a block. In part 2, we look at the content of the main components, i.e., Data.

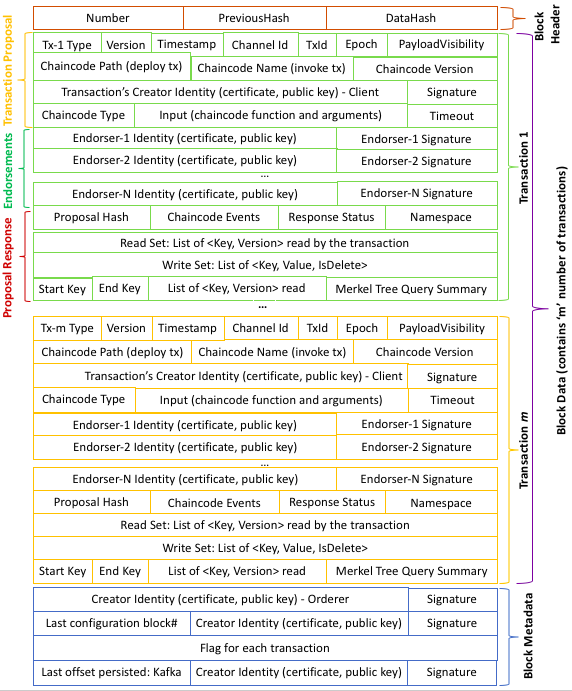
**type** **Block** struct {

**Header** \***BlockHeader**

**Data** \***BlockData**

**Metadata** \***BlockMetadata**

}

[](file:///C:\Workspace\document\fabric\Under%20Construction_%20Hyperledger%20Fabric%20V1.0_%20Block%20Structure%20(Part%202)%20_%20Senthilnathan's%20Blockchain_s%20Blog_files\block-structure.png)

## Block Header

The header of each block consists of three items which are

1. Number -- the unique block number.
2. PreviousHash -- hash of the previous block's header.
3. DataHash -- hash of the data segment of the current block.

The block number of each block is unique and is assigned sequentially starting from zero. The first block in a chain is special and is called genesis block which gets zero as its Number. The PreviousHash of the genesis blocks is set to nil, whereas the PreviousHash of the next block holds SHA256 hash of BlockHeader of the previous block. The DataHash holds SHA256 hash of BlockData of the current block.

**type** **BlockHeader** struct {

**Number** uint64

**PreviousHash** **[]**byte

**DataHash** **[]**byte

}

SizeOf(BlockHeader) = 8 bytes for Number + 32 bytes for PreviousHash + 32 bytes for  DataHash = 72 bytes

## Block Data

Each transaction data is encoded as byte array and stored in the Data field. The length of the Data field is equal to the number of transactions encapsulated in this Block.

**type** **BlockData** struct {

**Data** **[][]**byte

}

A transaction data is represented as an envelope which consists of *transaction payload* and a *signature*as shown in below structure (which is stored in Data field). The  *transaction payload*stores the Payload struct  which contains the actual transaction data and its metadata.

**type** **Envelope** struct {

**Payload** **[]**byte

**Signature** **[]**byte

}

### Payload Header

The Payload struct contains ChannelHeader and SignatureHeader in Header field. We have already described the SignatureHeader in part 1 and hence, we skip this field in part 2.  First, we describe Header field. Then, we move on to Data field

**type** **Payload** struct {

**Header** \***Header**

**Data** **[]**byte

}

**type** **Header** struct {

**ChannelHeader** **[]**byte

**SignatureHeader** **[]**byte

}

The ChannelHeader consists of eight fields which describes certain things about the transaction.

**type** **ChannelHeader** struct {

**Type** int32

**Version** int32

**Timestamp** \*google\_protobuf**.Timestamp**

**ChannelId** string

**TxId** string

**Epoch** uint64

**Extension** **[]**byte

}

The Type field denotes the transaction type such as endorser transaction, configuration transaction, etc.. and it can be one of the following.

HeaderType\_MESSAGE HeaderType **=** 0

HeaderType\_CONFIG HeaderType **=** 1

HeaderType\_CONFIG\_UPDATE HeaderType **=** 2

HeaderType\_ENDORSER\_TRANSACTION HeaderType **=** 3

HeaderType\_ORDERER\_TRANSACTION HeaderType **=** 4

HeaderType\_DELIVER\_SEEK\_INFO HeaderType **=** 5

HeaderType\_CHAINCODE\_PACKAGE HeaderType **=** 6

The Version field denotes protobuf version used for serialization and Epoch field is currently not used and set to zero. The identifier of the channel and transaction is stored in ChannelID and TxId, respectively. The time at which the transaction proposal message was created is stored in Timestamp field. In general, transaction is executed by a chaincode and hence, we need to be able to associate a given transaction with a chaincode. For this purpose, we have Extension field which hold the ChaincodeHeaderExtension struct which is given below.

**type** **ChaincodeHeaderExtension** struct {

**PayloadVisibility** **[]**byte

**ChaincodeId** \***ChaincodeID**

}

**type** **ChaincodeID** struct {

**Path** string

**Name** string

**Version** string

}

The PayloadVisibility field denotes the visibility of chaincode payload such as chaincode function to invoke, arguments, etc... Currently, this field is set to nil to enable full visibility of the chaincode payload. In future, it is expected to support *only hash* and *nothing* visibility options. The ChaincodeID field holds either a Path to the chaincode (for deploy transaction) or a Name of the chaincode to be invoked.

### Payload Data

The Data field in  Payload struct hold multiple actions per transaction. An action is a chaincode invoke. Currently, Fabric supports only one transaction action per transaction. Each

TransactionAction field holds a signature header and a payload. We have already seen that the content of a signature header is a creator identity and a nonce. Here, the header holds the identify of the client which submitted this transaction.  The  Payload field contains ChaincodeActionPayload struct which stores a proposal payload and a chaincode endorsed action. The proposal payload, i.e., ChaincodeProposalPayload struct, contains chaincode information, the input given to the chaincode function, etc... The response payload, i.e., ChaincodeEndorsedAction struct, contains a list of endorsers who endorsed this transaction and read/write set, etc...

The Input field of ChaincodeProposalPayload struct stores ChaincodeInvocationSpec which contains chaincode type, chaincode name and version (for invoke transaction) or chaincode path (for deploy transaction), arguments passed to the chaincode function, and timeout for response from the peer. Some of the secret information such as encryption key, etc can be passed to the chaincode in the TransientMap field but it does not get stored in the block for privacy/confidentiality/securiy reasons.

**type** **TransactionAction** struct {

**Header** **[]**byte

**Payload** **[]**byte

}

**type** **ChaincodeActionPayload** struct {

**ChaincodeProposalPayload** **[]**byte

**Action** \***ChaincodeEndorsedAction**

}

**type** **ChaincodeProposalPayload** struct {

**Input** **[]**byte

**TransientMap** **map**[string]**[]**byte

}

**type** **ChaincodeInvocationSpec** struct {

**ChaincodeSpec** \***ChaincodeSpec**

**IdGenerationAlg** string

}

**type** **ChaincodeSpec** struct {

**Type** **ChaincodeSpec\_Type**

**ChaincodeId** \***ChaincodeID**

**Input** \***ChaincodeInput**

**Timeout** int32

}

**type** **ChaincodeInput** struct {

**Args** **[][]**byte

}

**type** **ChaincodeEndorsedAction** struct {

**ProposalResponsePayload** **[]**byte

**Endorsements** **[]**\***Endorsement**

}

**type** **Endorsement** struct {

**Endorser** **[]**byte

**Signature** **[]**byte

}

**type** **ProposalResponsePayload** struct {

**ProposalHash** **[]**byte

**Extension** **[]**byte

}

**type** **ChaincodeAction** struct {

**Results** **[]**byte

**Events** **[]**byte

**Response** \***Response**

}

**type** **TxReadWriteSet** struct {

**DataModel** **TxReadWriteSet\_DataModel**

**NsRwset** **[]**\***NsReadWriteSet**

}

**type** **NsReadWriteSet** struct {

**Namespace** string

**Rwset** **[]**byte

}

**type** **KVRWSet** struct {

**Reads** **[]**\***KVRead**

**RangeQueriesInfo** **[]**\***RangeQueryInfo**

**Writes** **[]**\***KVWrite**

}

**type** **KVRead** struct {

**Key** string

**Version** \***Version**

}

**type** **Version** struct {

**BlockNum** uint64

**TxNum** uint64

}

**type** **RangeQueryInfo** struct {

**StartKey** string

**EndKey** string

**ItrExhausted** bool

**ReadsInfo** isRangeQueryInfo\_ReadsInfo

}

**type** **KVWrite** struct {

**Key** string

**IsDelete** bool

**Value** **[]**byte

}

## Block Metadata

The BlockMetadata stores four metadata, each as a byte array in Metadata field.

**type** **BlockMetadata** struct {

**Metadata** **[][]**byte

}

The four metadata stored in a block are listed below along with its index.

* SIGNATURES: signature on the block creation.  (index: 0)
* LAST\_CONFIG: reference to the last configuration block.  (index: 1)
* TRANSACTIONS\_FILTER: valid and invalid transactions in a block.  (index: 2)
* ORDERER: last offset persisted (kafka ordering metadata).  (index: 3)

The SIGNATURE, LAST\_CONFIG, and ORDERER metadata are added by the ordering service, whereas the TRANSACTION\_FILTER is added by the committer after validating the transaction based on the endorsement policy, verification of read-write set, etc... The TRANSACTION\_FILTER is a byte array of size equal to the number of transactions in the block. For each transaction, committer sets *validation code* in the byte array appropriately to represent the validation result. A set of *validation code*is given below.

TxValidationCode\_VALID TxValidationCode **=** 0

TxValidationCode\_NIL\_ENVELOPE TxValidationCode **=** 1

TxValidationCode\_BAD\_PAYLOAD TxValidationCode **=** 2

TxValidationCode\_BAD\_COMMON\_HEADER TxValidationCode **=** 3

TxValidationCode\_BAD\_CREATOR\_SIGNATURE TxValidationCode **=** 4

TxValidationCode\_INVALID\_ENDORSER\_TRANSACTION TxValidationCode **=** 5

TxValidationCode\_INVALID\_CONFIG\_TRANSACTION TxValidationCode **=** 6

TxValidationCode\_UNSUPPORTED\_TX\_PAYLOAD TxValidationCode **=** 7

TxValidationCode\_BAD\_PROPOSAL\_TXID TxValidationCode **=** 8

TxValidationCode\_DUPLICATE\_TXID TxValidationCode **=** 9

TxValidationCode\_ENDORSEMENT\_POLICY\_FAILURE TxValidationCode **=** 10

TxValidationCode\_MVCC\_READ\_CONFLICT TxValidationCode **=** 11

TxValidationCode\_PHANTOM\_READ\_CONFLICT TxValidationCode **=** 12

TxValidationCode\_UNKNOWN\_TX\_TYPE TxValidationCode **=** 13

TxValidationCode\_TARGET\_CHAIN\_NOT\_FOUND TxValidationCode **=** 14

TxValidationCode\_MARSHAL\_TX\_ERROR TxValidationCode **=** 15

TxValidationCode\_NIL\_TXACTION TxValidationCode **=** 16

TxValidationCode\_INVALID\_OTHER\_REASON TxValidationCode **=** 255

Other three metadata added by the orderer service, such as SIGNATURE, LAST\_CONFIG, and ORDERER, stores the metadata information along with a signature. For LAST\_CONFIG and ORDERER metadata, the *last configuration block number* and *last offset persisted* is set to Value field in Metadata struct given below.

**type** **Metadata** struct {

**Value** **[]**byte

**Signatures** **[]**\***MetadataSignature**

}

The Signature field holds the signature of the orderer as well as respective headers using MetadataSignature struct.

**type** **MetadataSignature** struct {

**SignatureHeader** **[]**byte

**Signature** **[]**byte

}

The Creator field holds x.509 certificate, publick key and membership service provided (MSP) who issued these identity to the client. The Nounce field contains some random bytes.

**type** **SignatureHeader** struct {

**Creator** **[]**byte

**Nonce** **[]**byte

}

The SIGNATURE metadata (i.e., index 0) uses the above three structures but set the Value field in Metadata struct to nil as it wants to add just the signature on block creation.