# NEAR Lake Primitive Types

This article contains the primitive types used by theNEAR Lake Framework package . These types are used to define the data structures used by the framework as well as provide some popular helper functions.

## Block

Important Notes onBlock * All the entities located on different shards were merged into one single list without differentiation. * Block * is not the fairest name for this structure either. NEAR Protocol is a sharded blockchain, so its block is actually an ephemeral structure that represents a collection of real blocks called chunks in NEAR Protocol.

### Block

Structure Definition

TheBlock type is used to represent a block in the NEAR Lake Framework. It is comprised by the following structure:

export

class

Block

{ constructor ( readonly streamerMessage :

StreamerMessage , private executedReceipts :

Receipt [ ] , readonly postponedReceipts :

Receipt [ ] , readonly transactions :

Transaction [ ] , private _actions :

Map < string ,

Action

, private _events :

Map < string ,

Event [ ]

, private _stateChanges :

StateChange [ ] )

{

} ...

// helper methods and getters omitted for brevity }

**streamerMessage**

Low-level structure for backward compatibility. As implemented in previous versions ofhear-lake-framework .

**postponedReceipts**

Receipts included on the chain but not executed yet marked as “postponed”: they are represented by the same structureReceipt (see the corresponding section in this doc for more details).

**transactions**

List of includedTransactions , converted intoReceipts .

Heads up! Note: You might want to know aboutTransactions to know where the action chain has begun. Unlike Ethereum, where a Transaction contains everything you may want to know about a particular interaction on the Ethereum blockchain, Near Protocol because of its asynchronous nature converts aTransaction into aReceipt before executing it. Thus, On NEAR,Receipts are more important for figuring out what happened on-chain as a result of a Transaction signed by a user. Read more aboutTransactions on Near here.

## Block

Helper Methods

export

class

Block

{ ...

// constructor omitted for brevity get

blockHash ( ) :

string

{ } get

prevBlockHash ( ) :

string

{ } get

blockHeight ( ) :

number

{ }

header ( ) :

BlockHeader

{ } receipts ( ) :

Receipt [ ]

{ } actions ( ) :

Action [ ]

{ } events ( ) :

Event [ ]

{ } stateChanges ( ) :

StateChange [ ]

{ }

actionByReceiptId ( receipt_id :

string ) :

Action

|

undefined

{ } eventsByReceiptId ( receipt_id :

string ) :

Event [ ]

{ } eventsByAccountId ( account_id :

string ) :

Event [ ]

{ }

private

buildActionsHashmap ( )

{ } private

buildEventsHashmap ( ) :

Map < string ,

Event [ ]

{ }

static

fromStreamerMessage ( streamerMessage :

StreamerMessage ) :

Block

{ } }

**blockHash**

Returns the block hash. A shortcut to get the data from the block header.

**prevBlockHash**

Returns the previous block hash. A shortcut to get the data from the block header.

**blockHeight**

Returns the block height. A shortcut to get the data from the block header.

**header(): BlockHeader**

Returns aBlockHeader structure of the block

SeeBlockHeader structure sections for details.

**receipts(): Receipt[]**

Returns a slice ofReceipts executed in the block.

Basically is a getter for theexecutedReceipts field.

**actions(): Action[]**

Returns an Array ofActions executed in the block.

**events(): Event[]**

ReturnsEvents emitted in the block.

**stateChanges(): StateChange[]**

Returns an Array ofStateChange occurred in the block.

**actionByReceiptId(receipt_id: string): Action | undefined**

ReturnsAction s of the providedreceipt_id from the block if any. Returnsundefined if there is no correspondingAction .

This method uses the internalBlock action field which is empty by default and will be filled with the block's actions on the first call to optimize memory usage.

The result is eitherAction | undefined since there might be a request for anAction byreceipt_id from another block, in which case this method will be unable to find theAction in the current block. In the other case, the request might be for anAction for areceipt_id that belongs to aDataReceipt where an action does not exist.

**eventsByReceiptId(receipt_id: string): Event[]**

Returns an Array of Events emitted byExecutionOutcome for the givenreceipt_id . There might be more than oneEvent for theReceipt or there might be none of them. In the latter case, this method returns an empty Array.

**eventsByAccountId(account_id: string): Event[]**

Returns an Array of Events emitted byExecutionOutcome for the givenaccount_id . There might be more than oneEvent for theReceipt or there might be none of them. In the latter case, this method returns an empty Array.

# BlockHeader

Replacement forBlockHeaderView fromnear-primitives . Shrunken and simplified.

note The originalBlockHeaderView is still accessible via the.streamerMessage attribute.

## BlockHeader

Structure Definition

export

class

BlockHeader

{ constructor ( readonly height :

number , readonly hash :

string , readonly prevHash :

string , readonly author :

string , readonly timestampNanosec :

string , readonly epochId :

string , readonly nextEpochId :

string , readonly gasPrice :

string , readonly totalSupply :

string , readonly latestProtocolVersion :

number , readonly randomValue :

string , readonly chunksIncluded :

number , readonly validatorProposals :

ValidatorStakeView [ ] )

{ } ...

// helper method omitted for brevity }

# Receipt

This field is a simplified representation of theReceiptView structure fromnear-primitives .

### Receipt

Structure Definition

export

class

Receipt

implements

Events

{ constructor ( readonly receiptKind :

ReceiptKind , readonly receiptId :

string , readonly receiverId :

string , readonly predecessorId :

string , readonly status :

ExecutionStatus , readonly executionOutcomeId ? :

string

|

undefined , readonly logs :

string [ ]

=

[ ] )

{ } ...

// helper methods omitted for brevity }

## Receipt

Fields

**receiptKind**

Defined the type of theReceipt :Action orData representing theActionReceipt andDataReceipt .

**receiptId**

The ID of theReceipt of theCryptoHash type.

**receiverId**

The receiver account id of theReceipt .

**predecessorId**

The predecessor account id of theReceipt .

**status**

Represents the status ofExecutionOutcome of theReceipt .

See theExecutionStatus enum section for the details.

**executionOutcomeId**

The id of theExecutionOutcome for theReceipt . Returnsnull if theReceipt isn’t executed yet and has a postponed status.

**logs**

The original logs of the correspondingExecutionOutcome of theReceipt .

Note: not all of the logs might be parsed as JSON Events (Events ).

## Receipt

Helper Methods

export

class

Receipt

{ ...

// constructor omitted for brevity get

events ( ) :

Event [ ]

{ }

static

fromOutcomeWithReceipt ( outcomeWithReceipt :

OutcomeWithReceipt ) :

Receipt

{ } }

**Receipt.events(): Events[]**

Returns an Array ofEvents for theReceipt , if any. This might be empty if thelogs field is empty or doesn't contain JSON Events compatible log records.

# Event

This structure is an ephemeral entity to provide access to theEvents Standard structure and keep data about the relatedReceipt for convenience.

### Interface for Capturing Data About an Event inhandleStreamerMessage()

The interface to capture data about an event has the following arguments:

- standard
- : name of standard, e.g. nep171
- version
- : e.g. 1.0.0
- event
- : type of the event, e.g. nft_mint
- data
- : associate event data. Strictly typed for each set{standard, version, event}
- inside corresponding NEP

### Event

Structure Definition

export

class

Event

{ constructor ( readonly relatedReceiptId :

string , readonly rawEvent :

RawEvent )

{ } ...

// helper methods omitted for brevity }

### Event

Methods

export

class

Event

{ ...

// constructor omitted for brevity static

fromLog ( log :

string ) :

Event

{ } }

# Transaction

A representation of theIndexerTransactionWithOutcome fromnear-indexer-primitives which is an ephemeral structure combiningSignedTransactionView fromnear-primitives andIndexerExecutionOutcomeWithOptionalReceipt fromnear-indexer-primitives .

This structure is very similar toReceipt . UnlikeReceipt , aTransaction has a few additional fields likesignerId ,signature , andoperations .

## Transaction

Structure Definition

export

class

Transaction

{ constructor ( readonly transactionHash :

string , readonly signerId :

string , readonly signerPublicKey :

string , readonly signature :

string , readonly receiverId :

string , readonly status :

ExecutionStatus , readonly executionOutcomeId :

string , readonly operations :

Operation [ ] )

{ } }

**Transaction.transactionHash**

Returns the hash of theTransaction inCryptoHash .

**Transaction.signerId**

Returns the signer account id of theTransaction .

**Transaction.signerPublicKey**

Returns thePublicKey of the signer of theTransaction .

**Transaction.signature**

Returns theSignature theTransaction was signed with.

**Transaction.receiverId**

Returns the receiver account id of theTransaction .

**Transaction.status**

Returns the status of theTransaction asExecutionStatus .

**Transaction.executionOutcomeId**

Returns the id of theExecutionOutcome for theTransaction .

**Transaction.operations**

Returns an Array ofOperation for theTransaction .

# StateChange

This structure is almost an identical copy of theStateChangeWithCauseView fromnear-primitives with a propagated additional fieldaffectedAccountId .

## StateChange

Structure Definition

export

class

StateChange

{ constructor ( readonly cause :

StateChangeCause , readonly value :

StateChangeValue )

{ }

get

affectedAccountId ( ) :

string

{ }

static

fromStateChangeView ( stateChangeView :

StateChangeWithCauseView )

{ } }

**StateChange.cause**

Returns thecause of theStateChange .

**StateChange.value**

Returns thevalue of theStateChange .

**StateChange.affectedAccountId(): string**

Returns the account id of theStateChange .

**StateChange.fromStateChangeView(stateChangeView: StateChangeWithCauseView): StateChange**

Returns theStateChange from theStateChangeWithCauseView . Created for backward compatibility.Edit this page Last updatedonDec 9, 2023 bygagdiez Was this page helpful? Yes No

Previous Migrating to NEAR Lake framework Next JS basic tutorial