Transactions

NEAR is asynchronous by design. While it opens a wide range of possibilities for smart contracts implementations, it may also add to confusion among beginners and newcomers from other blockchain platforms. Transactions in NEAR may contain actions that do asynchronous work, in such cases keep in mind the possible outcomes of success or failure of the transaction. For example, if a transaction contains a cross-contract call, it may be marked as successful, but the other contract execution might fail. This article covers possible scenarios for this.

A transaction is the smallest unit of work that can be assigned to the network. "Work" in this case means compute (executing a function) or storage (reading/writing data). A transaction is composed of one or moreAction s. A transaction with more than one action is referred to as a "batch transaction". Since transactions are the smallest units of work, they are also atomic, but again, asynchronous actions do not necessarily cascade their success or failure the whole transaction.

There is also a concept ofReceipt , which is either "request to apply anAction " or "result of theAction ". All cross-contract communication is done through receipts. An action may result in one or more receipts. The Blockchain may be seen as a series of Transactions, but it's also a series of Receipts.

tip You can useNEAR Blockchain Explorer to inspect and see all actions and receipts related to a transaction. An in depth documentation about transactions can be found inNEAR Protocol Specifications (nomicon.io) . On this page we give an overview of the important aspects of transactions on NEAR.

Transaction

ATransaction is a collection ofActions that describe what should be done at the destination (thereceiver account).

EachTransaction is augmented with critical information about its:

- origin
- (cryptographically signed bysigner
-)
- destination
- or intention (sent or applied toreceiver
-)
- recency
- (block_hash
- from recent block within acceptable limits -1 epoch
-)
- uniqueness
- (nonce
- must be unique for a givensigner
- AccessKey
-)

Action

AnAction is a composable unit of operation that, together with zero or more otherActions , defines a sensibleTransaction . There are currently 8 supportedAction types:

- FunctionCall
- to invoke a method on a contract (and optionally attach a budget for compute and storage)
- Transfer
- to move tokens from between accounts
- DeployContract
- to deploy a contract
- CreateAccount
- to make a new account (for a person, contract, refrigerator, etc.)
- DeleteAccount
- to delete an account (and transfer the balance to a beneficiary account)
- AddKey
- to add a key to an account (eitherFullAccess
- orFunctionCall
- access)
- DeleteKey
- to delete an existing key from an account
- Stake
- to express interest in becoming a validator at the next available opportunity

You can find more about the technical details ofAction s in theNEAR nomicon .

Receipt

AReceipt is the only actionable object in the system. Therefore, when we talk about "processing a transaction" on the NEAR platform, this eventually means "applying receipts" at some point.

A good mental model is to think of aReceipt as a paid message to be executed at the destination (receiver). And aTransaction is an externally issued request to create theReceipt (there is a 1-to-1 relationship).

There are several ways of creatingReceipts :

- issuing aTransaction
- returning a promise (related to cross-contract calls)
- issuing a refund

You can find more about the technical details ofReceipts in theNEAR nomicon .

Transaction Atomicity

Since transactions are converted to receipts before they are applied, it suffices to talk about receipt atomicity. Receipt execution is atomic, meaning that either all the actions are successfully executed or none are. However, one caveat is that a function call transaction, unlike other transactions, can spawn an indefinite amount of receipts, and while each receipt is atomic, the success or failure of one receipt doesn't necessarily affect the status of other receipts spawned by the same transaction.

info When designing a smart contract, you should always consider the asynchronous nature of NEAR Protocol.

Transaction Status

You can query the status of a transaction throughRPC API orNEAR CLI . An example of the query result looks like this:

```
{ status :
{
SuccessValue :
''
} , transaction :
{ actions :
[
{
Transfer :
{
deposit :
'50000000000000000000000'
}
}
] , hash :
'54sZqhqvwynMmMEcN7LcNLxUjx2o5xyFn2FC4zkpNUas' , nonce :
64986174290372 , public_key :
'ed25519:EDPw6PkPetebJvrp1jtcvknCGeFguf7LrSGFCRqLrXks' , receiver_id :
'1167fc268181c9ee30e914016d2148b4b7fdc0dc2d70e2a29df9c65756b52116' , signature :
'ed25519:52mSUmSBCXe1fF2m6cWbhQPUFuKz965aWCP6Aa4Jaaf1Kr93wDfJ8DKwkCUhEdahqJuDNNcMqDP2qjX5Xb1XRvsf'
, signer_id :
'sweat_welcome.near' } , transaction_outcome :
{ block_hash :
'BS5ongkXQgcqFuH8xbJBfLVjF8fGhVip3wogi5f1SxpN' , id :
```

'54sZqhqvwynMmMEcN7LcNLxUjx2o5xyFn2FC4zkpNUas' , outcome :

{ executor_id :

'sweat_welcome.near' , gas_burnt :

4174947687500 , logs :

[ ] , metadata :

{

gas_profile :

null ,

version :

1

} , receipt_ids :

[

'6WGRhQyaxzKyMW1YaMiPpbH5u2QYM7hMFrkbW38guY9D'

] , status :

{ SuccessReceiptId :

'6WGRhQyaxzKyMW1YaMiPpbH5u2QYM7hMFrkbW38guY9D' } , tokens_burnt :

'417494768750000000000' } , "proof" :

[ ] } , receipts_outcome :

[{ block_hash :

'GGeKQ2GZoQffwef5oA4bRjYes7Cwp8fn3qiwo5ZpVKiN' , id :

'6WGRhQyaxzKyMW1YaMiPpbH5u2QYM7hMFrkbW38guY9D' , outcome :

{ executor_id :

'1167fc268181c9ee30e914016d2148b4b7fdc0dc2d70e2a29df9c65756b52116' , gas_burnt :

4174947687500 , logs :

[ ] , metadata :

{

gas_profile :

[ ] ,

version :

3

} , receipt_ids :

[

'5m6D2DxLX3A59cAMZJmd6iTkYqL3QEE3Cr2FnXwzzvSr'

] , status :

{

SuccessValue :

''

} , tokens_burnt :

'417494768750000000000' } , "proof" :

[ ] } , { block_hash :

'A9vaFWg9Dv9tSvtQxf8j2mna4hV3UUG6wzNqjVferp57' , id :

'5m6D2DxLX3A59cAMZJmd6iTkYqL3QEE3Cr2FnXwzzvSr' , outcome :

{ executor_id :

'sweat_welcome.near' , gas_burnt :

223182562500 , logs :

[ ] , metadata :

{

gas_profile :

[ ] ,

version :

3

} , receipt_ids :

[ ] , status :

{

SuccessValue :

''

} , tokens_burnt :

'0' } , "proof" :

[ ] }] } The query result displays:

- the overall status of the transaction,
- the outcomes of the transaction,
- and the outcome of the receipts generated by this transaction.

Thestatus field appears at:

- the top-level, where it indicates whether all actions in the transaction have been successfully executed,
- undertransaction_outcome
- , where it indicates whether the transaction has been successfully converted to a receipt,
- underreceipts_outcome
- for each receipt, where it indicates whether the receipt has been successfully executed.

Thestatus is an object with a single key, one of the following four:

- status: { SuccessValue: 'val or empty'}
-
    - the receipt or transaction has been successfully executed. If it's the result of a function call receipt, the value is the return value of the function, otherwise the value is empty.
- status: { SuccessReceiptId: 'id_of_generated_receipt' }
-
    - either a transaction has been successfully converted to a receipt, or a receipt is successfully processed and generated another receipt. The value of this key is the id of the newly generated receipt.
- status: { Failure: {} }'
-
    - transaction or receipt has failed during execution. The value will include error reason.
- status: { Unknown: '' }'
-
    - the transaction or receipt hasn't been processed yet.

note For receipts,SuccessValue andSuccessReceiptId come from the last action's execution. The results of other action executions in the same receipt are not returned. However, if any action fails, the receipt's execution stops, and the failure is returned, meaning thatstatus would beFailure . And if the last action is not a function call and it's successfully executed, the result will be an emptySuccessValue note For receipts, The last receipt in the list is therefund receipt. Refund receipts do not actually cost anygas , but they still count the gas towards the block gas. In this case, the refund receipt is5m6D2DxLX3A59cAMZJmd6iTkYqL3QEE3Cr2FnXwzzvSr . The top-levelstatus indicates whether all actions in the transaction have been successfully executed. However, one caveat is that the successful execution of the function call does not necessarily mean that the receipts spawned from the function call are all successfully executed.

For example:

pub

fn

transfer (receiver_id :

String)

{ Promise :: new (receiver_id) . transfer (10) ; } This function schedules a promise, but its return value is unrelated to that promise. So even if the promise fails, potentially becausereceiver_id does not exist, a transaction that calls this function will still haveSuccessValue in the overallstatus . You can check the status of each of the receipts generated by going throughreceipt_outcomes in the same query result.

Finality

Transaction finality is closely tied to the finality of the block in which the transaction is included. However, they are not necessarily the same because often, one is concerned with whether the receipts, not the transaction itself, are final since receipt execution is where most of the work is done. Therefore, to verify the finality of a transaction, you can query the transaction and check if all the block hashes of the transactions and receipts generated from the transaction are final.

Previous Creating Accounts Next Gas