Callbacks

NEAR Protocol is a sharded, proof-of-stake blockchain that behaves differently than proof-of-work blockchains. When interacting with a native Rust (compiled to Wasm) smart contract, cross-contract calls are asynchronous. Callbacks are used to either get the result of a cross-contract call or tell if a cross-contract call has succeeded or failed.

Calculator Example

A callback method can be declared in your contract class as a regular method decorated with thecall({}) decorator. Be sure to pass in theprivateFunction: true option to the decorator. This will ensure that the method is only callable by the contract itself.

For example, let's assume the calculator is deployed oncalc.near , we can use the following:

@ NearBindgen ({ }) export

class

CalculatorCallerContract

{ @ call ({ }) sum_a_b ({ a , b })

{ let calculatorAccountId =

"calc.near" ; // Call the methodsum on the calculator contract. // Any unused GAS will be attached since the default GAS weight is 1. // Attached deposit is defaulted to 0. return

NearPromise . new (calculatorAccountId) . functionCall ("sum" ,

{ a , b } ,

BigInt (0) ,

BigInt (100000000000000)) ; }

@ call ({

privateFunction :

true

}) sum ({ a , b })

{ return a + b ; } }

Allowlist Example

Next we'll look at a simple cross-contract call that is made to an allowlist smart contract, returning whether an account is in the list or not.

The common pattern with cross-contract calls is to call a method on an external smart contract, use.then syntax to specify a callback, and then retrieve the result or status of the promise. The callback will typically live inside the same, calling smart contract. There's a special decorator parameter used for protecting the callback function, which isprivateFunction: true . We'll see this pattern in the example below.

The following example demonstrates two common approaches to callbacks using the high-level cross-contract approach withNearPromise .

@ NearBindgen ({ }) export

class

ExtAllowlist

{ // ...

@ call ({ }) is_allowlisted ({ staking_pool_account_id })

{ return

this . allowlist . get (staking_pool_account_id)

!=

null ; } ; } After creating the class, we'll show a simple flow that will make a cross-contract call to the allowlist smart contract, asking if the accountidea404.testnet is allowlisted.

@ NearBindgen ({ }) export

class

Contract

{ @ call ({ }) xcc_query_allowlist ( )

{ // Call the method is_allowlisted on the allowlisted contract. Static GAS is only attached to the callback. // Any unused GAS will be split between the function call and the callback since both have a default unused GAS weight of 1 // Attached deposit is defaulted to 0 for both the function call and the callback. return

NearPromise . new ("allowlist.near") . functionCall ("is_allowlisted" ,

{

staking_pool_account_id :

"idea404.testnet"

} ,

BigInt (0) ,

BigInt (100000000000000)) . then ("internalCallbackMethod" ,

{ } ,

BigInt (0) ,

BigInt (100000000000000)) ; }

@ call ({

privateFunction :

true

}) internalCallbackMethod ( )

{ assert (near . promiseResultsCount ( )

===

BigInt (1) ,

"Error: expected 1 promise result") ; let result =

JSON . parse (near . promiseResult (0)) ; return result ; } The syntax begins withNearPromise.new()(which initializes the async call to the designated . Subsequent calls to this program in this account are invoked using.functionCall() . The.functionCall() method takes in the following parameters:

- functionName
- : the name of the method to call on the contract
- args
- : the arguments to pass to the method
- amount
- : the amount of Ⓝ to attach to the call
- gas
- : the amount of GAS units to attach to the call

There are a couple things to note when doing these function calls:

1. You can attach a deposit of Ⓝ, in yoctoⓃ to the call by specifying theamount
2. parameter. This value is defaulted to 0 (1 Ⓝ = 1000000000000000000000000 yoctoⓃ, or 1^24 yoctoⓃ).
3. You can attach an amount of GAS units by specifying thegas

4. method. This value is defaulted to 0.Edit this page Last updatedonJan 20, 2023 byDennis Was this page helpful? Yes No

Previous Payable Methods Next Promises: Introduction