# FAQ for NEAR JavaScript API

A collection of Frequently Asked Questions by the community.

## General

### Can I usenear-api-js

on a static html page?

You can load the script from a CDN.

< script

# src

" https://cdn.jsdelivr.net/npm/near-api-js@0.45.1/dist/near-api-js.min.js "

</ script

note Make sure you load the latest version.

Versions list is onnpmjs.com Example Implementation

< html

# lang

" en "

< head

< meta

# charset

" UTF-8 "

< meta

# http-equiv

" X-UA-Compatible "

# content

" IE=edge "

< meta

# name

" viewport "

# content

" width=device-width, initial-scale=1.0 "

< title

Document </ title

</ head

< body

< ul

# id

" messages "

</ ul

< textarea

# id

" text "

# placeholder

" Add Message "

</ textarea

< button

# id

" add-text "

Add Text </ button

< script

# src

" https://cdn.jsdelivr.net/npm/near-api-js@0.45.1/dist/near-api-js.min.js "

</ script

< script

// connect to NEAR const near =

new

nearApi . Near ( { keyStore :

new

nearApi . keyStores . BrowserLocalStorageKeyStore ( ) , networkId :

'testnet' , nodeUrl :

'https://rpc.testnet.near.org' , walletUrl :

'https://testnet.mynearwallet.com/' } ) ;

// connect to the NEAR Wallet const wallet =

new

nearApi . WalletConnection ( near ,

'my-app' ) ;

// connect to a NEAR smart contract const contract =

new

nearApi . Contract ( wallet . account ( ) ,

'guest-book.testnet' ,

{ viewMethods :

[ 'getMessages' ] , changeMethods :

[ 'addMessage' ] } ) ;

const button =

document . getElementById ( 'add-text' ) ; if

( ! wallet . isSignedIn ( ) )

{ button . textContent

=

'SignIn with NEAR' }

// call the getMessages view method contract . getMessages ( ) . then ( messages

=>

{ const ul =

document . getElementById ( 'messages' ) ; messages . forEach ( message

=>

{ const li =

document . createElement ( 'li' ) ; li . textContent

=

{ message . sender } - { message . text } ; ul . appendChild ( li ) ; } ) } ) ;

// Either sign in or call the addMessage change method on button click document . getElementById ( 'add-text' ) .
addEventListener ( 'click' ,

( )

=>

{ if

( wallet . isSignedIn ( ) )

{ contract . addMessage ( { args :

{

text :

document . getElementById ( 'text' ) . value

} , amount : nearApi . utils . format . parseNearAmount ( '1' ) } ) }

else

{ wallet . requestSignIn ( { contractId :

'guest-book.testnet' , methodNames :

[ 'getMessages' ,

'addMessage' ] } ) ; } } ) ; </ script

</ body

</ html

### What front-end frameworks can I use the JavaScript API with?

The JavaScript API is framework-agnostic. You can include it in any front-end framework, such as React, Vue, Angular, and others.

You can usecreate-near-app to quickly bootstrap projects with different templates:

npx create-near-app

### Can I use the JavaScript API with mobile JavaScript frameworks such as React Native?

The JavaScript API can be used in most JavaScript runtimes, and under the hood, it's an abstraction over NEAR's RPC API . However, notice that the Wallet can't be used everywhere. For example, in React Native apps you'll be able to use the Wallet in web versions of the apps, but it won't work in the native app deployments.

You can use the Wallet inWebView components in iOS or Android, however be aware that it usesLocalStorage forKeyStore , and it's your responsibility to persist the storage when you manage loading ofWebView components.

## Transactions

### How to check the status of transaction

Please refer to examples about transactions in theCookbook .

### How transactions are signed and sent by near-api-js

There are a few steps involved before transaction data is communicated to the network and eventually included in a block. The following steps are taken when creating, signing and ultimately a transaction from a user's account:

1. The user creates a transaction object using theaccount.signAndSendTransaction method
2. . This method accepts an array of actions and returns an object for the outcome of the transaction.
3. The transaction is signed using theaccount.signTransaction method
4. . This method accepts an array of actions and returns a signed transaction object.
5. The signed transaction object is sent to the network using theaccount.connection.provider.sendTransaction method
6. . This method accepts a signed transaction object and returns a transaction hash. This stepperforms the borsh serialization of the transaction object
7. and calls thebroadcast_tx_commit JSON RPC method with the serialized transaction object encoded in base64
8. .

### How to send batch transactions

You may batch send transactions by using thesignAndSendTransaction({}) method fromaccount . This method takes an array of transaction actions, and if one fails, the entire operation will fail. Here's a simple example:

const

{ connect , transactions , keyStores }

=

require ( "near-api-js" ) ; const fs =

require ( "fs" ) ; const path =

require ( "path" ) ; const homedir =

require ( "os" ) . homedir ( ) ;

const

CREDENTIALS_DIR

=

".near-credentials" ; const

CONTRACT_NAME

=

"spf.idea404.testnet" ; const

WASM_PATH

= path . join ( __dirname ,

"../build/uninitialized_nft.wasm" ) ;

const credentialsPath = path . join ( homedir ,

CREDENTIALS_DIR ) ; const keyStore =

new

keyStores . UnencryptedFileSystemKeyStore ( credentialsPath ) ;

const config =

{ keyStore , networkId :

"testnet" , nodeUrl :

"https://rpc.testnet.near.org" , } ;

sendTransactions ( ) ;

async

function

sendTransactions ( )

{ const near =

await

connect ( {

... config , keyStore } ) ; const account =

await near . account ( CONTRACT_NAME ) ; const args =

{

some_field :

1 ,

another_field :

"hello"

} ;

const balanceBefore =

await account . getAccountBalance ( ) ; console . log ( "Balance before:" , balanceBefore ) ;

try

{ const result =

await account . signAndSendTransaction ( { receiverId :

CONTRACT_NAME , actions :

[ transactions . deployContract ( fs . readFileSync ( WASM_PATH ) ) ,

// Contract does not get deployed transactions . functionCall ( "new" ,

Buffer . from ( JSON . stringify ( args ) ) ,

10000000000000 ,

"0" ) ,

// this call fails transactions . transfer ( "1"

+

"0" . repeat ( 24 ) ) ,

// 1 NEAR is not transferred either ] , } ) ; console . log ( result ) ; }

catch

( e )

{ console . log ( "Error:" , e ) ; }

const balanceAfter =

await account . getAccountBalance ( ) ; console . log ( "Balance after:" , balanceAfter ) ; } Balance before: { total: '499878780549598382000000000', stateStaked: '455539000000000000000000', staked: '0', available: '454324880549598382000000000' } Receipts: 2PPueY6gnA4YmmQUzc8DytNBp4PUpgTDhmEjRSHHVHBd, 3isLCW9SBH1MrPjeEPAmG9saHLj9Z2g7HxzfBdHmaSaG Failure [spf.idea404.testnet]: Error: {"index":1,"kind":{"ExecutionError":"Smart contract panicked: panicked at 'Failed to deserialize input from JSON.: Error(\"missing field owner_id\", line: 1, column: 40)', nft/src/lib.rs:47:1"}} Error: ServerTransactionError: {"index":1,"kind":{"ExecutionError":"Smart contract panicked: panicked at 'Failed to deserialize input from JSON.: Error(\"missing field owner_id\", line: 1, column: 40)', nft/src/lib.rs:47:1"}} at parseResultError (/Users/dennis/Code/naj-test/node_modules/near-api-js/lib/utils/rpc_errors.js:31:29) at Account. (/Users/dennis/Code/naj-test/node_modules/near-api-js/lib/account.js:156:61) at Generator.next () at fulfilled (/Users/dennis/Code/naj-test/node_modules/near-api-js/lib/account.js:5:58) at processTicksAndRejections (node:internal/process/task_queues:96:5) { type: 'FunctionCallError', context: undefined, index: 1, kind: { ExecutionError: 'Smart contract panicked: panicked at \'Failed to deserialize input from JSON.: Error("missing field owner_id", line: 1, column: 40)\', nft/src/lib.rs:47:1' }, transaction_outcome: { block_hash: '5SUhYcXjXR1svCxL5BhCuw88XNdEjKXqWgA9X4XZW1dW', id: 'SKQqAgnSN27fyHpncaX3fCUxWknBrMtxxytWLRDQfT3', outcome: { executor_id: 'spf.idea404.testnet', gas_burnt: 4839199843770, logs: [], metadata: [Object], receipt_ids: [Array], status: [Object], tokens_burnt: '483919984377000000000' }, proof: [ [Object], [Object], [Object], [Object], [Object] ] } } Balance after: { total: '499851199593468682700000000', stateStaked: '455539000000000000000000', staked: '0', available: '454297299593468682700000000' } You may also find an example of batch transactions in theCookbook .

# Accounts

### What’s the difference betweenAccount

andConnectedWalletAccount ?

Interaction with the wallet is only possible in a web-browser environment because NEAR’s Wallet is web-based. The difference betweenAccount andConnectedWalletAccount is mostly about the way it signs transactions. TheConnectedWalletAccount uses the wallet to approve transactions. Under the hood theConnectedWalletAccount inherits and overrides some methods ofAccount .

### How to create implicit accounts?

You can read about it in the article aboutImplicit Accounts .

# Contracts

### How do I attach gas / a deposit?

Aftercontract is instantiated you can then call the contract and specify the amount of attached gas.

await contract . method_name ( { arg_name :

"value" ,

// argument name and value - pass empty object if no args required } , "300000000000000" ,

// attached GAS (optional) "100000000000000000000000"

// attached deposit in yoctoNEAR (optional) ) ;

# Common Errors

## RPC Errors

Refer to the exhaustivelist of error messages that RPC endpoints throws and JavaScript API propagates.

## Missing contract methods

When constructing aContract instance on the client-side, you need to specify the contract's methods. If you misspell, mismatch, or miss method names - you'll receive errors about missing methods.

There are a few cases of missing or wrong methods:

- When you call a method you didn't specify in the constructor.
- When you call a method that doesn't exist on the blockchain's contract (but you did specify it in the client-side constructor).
- When you mismatch betweenviewMethods
- andchangeMethods
- .

For example, let's look at the following contract code. It contains oneview and onecall method:

@ NearBindgen class

MyContract

extends

NearContract

{ constructor ( )

{

super ( ) ;

}

@view method_A_view ( ) : string { return

'Hi' ; }

@call method_B_call ( ) :

void

{ } }

### Client-side errors for missing methods

#### TypeError: contract.METHOD_NAME is not a function

The following contract constructor declares onlymethod_A_view , it doesn't declaremethod_B_call

const contract =

await

new

nearAPI . Contract ( walletConnection . account ( ) ,

'guest-book.testnet' , { viewMethods :

[ 'method_A_view' ] ,

// <=== Notice this changeMethods :

[ ] ,

// <=== Notice this sender : walletConnection . getAccountId ( ) , } ) ;

// This will work because we declared method_A_view in constructor await contract . method_A_view ( ) ;

// This will throw TypeError: contract.method_B_call is not a function // because we didn't declare method_B_call in constructor, // even if it exists in the real contract. await contract . method_B_call ( ) ;

// This will also throw TypeError: contract.method_C is not a function, // not because method_C doesn't exist on the contract, but because we didn't declare it // in the client-side constructor. await contract . method_C ( ) ;

**RPC errors for missing methods**

**wasm execution failed with error: FunctionCallError(MethodResolveError(MethodNotFound))**

In this example we specify and call a method, but this method doesn't exist on the blockchain:

const contract =

await

new

nearAPI . Contract ( // ... { viewMethods :

[ "method_C" ] ,

// <=== method_C doesn't exist on the contract above changeMethods :

[ ] , // ... } ) ; // We did specify method_C name in constructor, so this function exists on client-side contract instance, // but a method with this name does not exist on the actual contract on the blockchain. // This will return an error from RPC call FunctionCallError(MethodResolveError(MethodNotFound)) // and will throw it on the client-side await contract . method_C ( ) ;

// Notice: if we call method_A_view we get TypeError: contract.method_A_view is not a function. // Even though the method exists on the actual blockchain contract, // we didn't specify method_A_view in the contract's client-side constructor. await contract . method_A_view ( ) ;

**wasm execution failed with error: FunctionCallError(HostError(ProhibitedInView { method_name: "storage_write" }))**

Last case is when you mismatchviewMethods andchangeMethods .

In the contract above we declared:

- A@view
- method namedmethod_A_view
- A@call
- method namedmethod_B_call

In a client-side constructor, the contract's@view method names must be specified underviewMethods , and the contract's@call method names must be specified underchangeMethods . If we mismatch between the types we will receive errors.

For example:

const contract =

await

new

nearAPI . Contract ( // ... { viewMethods :

[ 'method_B_call' ] ,

// <=== here should be method_A_view changeMethods :

[ 'method_A_view' ] ,

// <=== and here should be method_B_call // ... } ) ;

// This will return an error from RPC call and throw: //wasm execution failed with error: FunctionCallError(HostError(ProhibitedInView { method_name: "storage_write" })) // This error indicates that we are trying to call a state-changing method but declare it as a read-only method in client-side. await contract . method_B_call ( ) ;

// The following behavior is undefined and might not work as expected. //method_A_veiw should be declared under viewMethods and in our example here we declare it under changeMethods. await contract . method_A_view ( ) ;

## Class{X}

is missing in schema: publicKey

There is currently a known issue with the JavaScript API library, when youimport it more than once it might cause a namespace collision.

A temporary workaround: make sure you don't re-import it, for example when running tests.

## regeneratorRuntime

is not defined

You are probably usingParcel like we do inother examples . Please make sure you have this line at the top of your main JS file. (Most likelyindex.js ):

import

"regenerator-runtime/runtime" ; * Also, ensure the dependencies for this are added to the project by checking the dev dependencies in yourpackage.json * . If not found you can install this by running the following in your terminal:

npm install regenerator-runtime --save-dev

## Window error usingNode.js

You're maybe using a KeyStore that's for the browser. Instead, use afilesystem key or private key string.

Browser KeyStore:

const

{ keyStores }

=

require ( "near-api-js" ) ; const keyStore =

new

keyStores . BrowserLocalStorageKeyStore ( ) ; FileSystem KeyStore:

const

{ keyStores }

=

require ( "near-api-js" ) ; const

KEY_PATH

=

"~./near-credentials/testnet/example-account.json" ; const keyStore =

new

keyStores . UnencryptedFileSystemKeyStore ( KEY_PATH ) ;Edit this page Last updatedonDec 9, 2023 bygagdiez Was this page helpful? Yes No

Previous Utilities Next NEAR SDKs