

# State & Data Structures

Each contract has its own state (storage), which only they can modify but [anyone can see](#) .

A contract stores all its data in a key-value storage. This however is abstracted from you by the SDK through [serialization](#) .

info Contracts [pay for their storage](#) by locking part of their balance. Currently it costs ~1 Ⓝ to store 100KB

## Defining the State

The contract's state is defined by the [main class attributes](#) , and accessed through them.

In the state you can store constants, native types, and complex objects. When in doubt, prefer to use [SDK collections](#) over native ones, because they are optimized for the [serialized key-value storage](#) .

- JavaScript
- Rust

storage-js/src/index.ts loading ... [See full example on GitHub](#) storage-rs/contract/src/lib.rs loading ... [See full example on GitHub](#)

## Data Structures

The NEAR SDK exposes a series of structures ([Vectors](#) , [Sets](#) , [Maps](#) and [Trees](#) ) to simplify storing data in an efficient way.

Instantiation All structures need to be initialized using a unique prefix , which will be used to identify the structure's keys in the [serialized state](#)

- JavaScript
- Rust

storage-js/src/index.ts loading ... [See full example on GitHub](#) storage-rs/contract/src/lib.rs loading ... [See full example on GitHub](#)

### Vector

Implements a [vector/array](#) which persists in the contract's storage. Please refer to the Rust and AS SDK's for a full reference on their interfaces.

- JavaScript
- Rust

storage-js/src/index.ts loading ... [See full example on GitHub](#) \* vector.rs \* lib.rs

storage-rs/contract/src/vector.rs loading ... [See full example on GitHub](#) storage-rs/contract/src/lib.rs loading ... [See full example on GitHub](#)

### Map

Implements a [map/dictionary](#) which persists in the contract's storage. Please refer to the Rust and AS SDK's for a full reference on their interfaces.

- JavaScript
- Rust

storage-js/src/index.ts loading ... [See full example on GitHub](#) \* map.rs \* lib.rs

storage-rs/contract/src/map.rs loading ... [See full example on GitHub](#) storage-rs/contract/src/lib.rs loading ... [See full example on GitHub](#) Nesting of Objects - Temporary Solution In the JS SDK, you can store and retrieve elements from a nested map or object, but first you need to construct or deconstruct the structure from state. This is a temporary solution until the improvements have been implemented to the SDK. Here is an example of how to do this:

```
import
```

```
{
```

```
NearBindgen , call , view , near ,
```

```
UnorderedMap
```

```

}

from
"near-sdk-js" ;

@ NearBindgen ( { } ) class
StatusMessage
{ records :
  UnorderedMap ; constructor ( )
  { this . records
    =
    new
    UnorderedMap ( "a" ) ; }
  @ call ( { } ) set_status ( { message , prefix } :
  { message :
    string ; prefix :
    string
  } )
  { let account_id = near . signerAccountId ( ) ;
    const inner :
    any
    =
    this . records . get ( "b"
    + prefix ) ; const inner_map :
    UnorderedMap
    = inner ?
    UnorderedMap . deserialize ( inner ) :
    new
    UnorderedMap ( "b"
    + prefix ) ;
    inner_map . set ( account_id , message ) ;
    this . records . set ( "b"
    + prefix , inner_map ) ; }
  @ view ( { } ) get_status ( { account_id , prefix } :
  { account_id :
    string ; prefix :
    string
  } )
  { const inner :

```

```

any
=
this . records . get ( "b"
+ prefix ) ; const inner_map :
UnorderedMap
= inner ?
UnorderedMap . deserialize ( inner ) :
new
UnorderedMap ( "b"
+ prefix ) ; return inner_map . get ( account_id ) ; } }

```

## Set

Implements [aset](#) which persists in the contract's storage. Please refer to the Rust and AS SDK's for a full reference on their interfaces.

- JavaScript
- Rust

storage-js/src/index.ts loading ... [See full example on GitHub](#) \* set.rs \* lib.rs

storage-rs/contract/src/set.rs loading ... [See full example on GitHub](#) storage-rs/contract/src/lib.rs loading ... [See full example on GitHub](#)

## Tree

An ordered equivalent of Map. The underlying implementation is based on an [AVL](#) . You should use this structure when you need to: have a consistent order, or access the min/max keys.

- Rust
- tree.rs
- lib.rs

storage-rs/contract/src/tree.rs loading ... [See full example on GitHub](#) storage-rs/contract/src/lib.rs loading ... [See full example on GitHub](#)

## Storage Cost

Your contract needs to lock a portion of their balance proportional to the amount of data they stored in the blockchain. This means that:

- If more data is added and the storage increases ↑
- , then your contract's balance decreases ↓
- .
- If data is deleted and the storage decreases ↓
- , then your contract's balance increases ↑
- .

Currently, it cost approximately 1 Ⓝ to store 100kb of data.

info You can save on smart contract storage if using NEAR Account IDs by encoding them using base32. Since they consist of [a-z.-\_] characters with a maximum length of 64 characters, they can be encoded using 5 bits per character, with terminal\0 . Going to a size of  $65 * 5 = 325$  bits from the original  $(64 + 4) * 8 = 544$  bits. This is a 40% reduction in storage costs. caution An error will raise if your contract tries to increase its state while not having NEAR to cover for storage. warning Be mindful of potential [small deposit attacks](#) [Edit this page](#) Last updated on Jan 31, 2024 by gagdiez Was this page helpful? Yes No

[Previous Environment](#) [Next Transfers & Actions](#)