Prefixed Storage

Why Is Prefixed Storage Needed?

Before we understand what the PrefixedStorage struct is, we should first understand the problem they attempt to solve. We may be tempted to think that the key-value model as described inStorage solves all our storage needs. And we'd technically be correct in thinking that. It is possible to efficiently store vast amounts of user data by just using the methods described in that page. However, we will see that doing this would be cumbersome for the developer in many cases. That's why there are many additional storage structures built on top of the methods described inStorage . Prefixed storage is an additional structure built on top of these methods for the convenience of the developer.

The Problem

We will go over a storage problem and discuss how we might tackle it with the methods we learnt inStorage to get to the root of what prefixed storage actually is. Suppose that we have contract that stores a different password (String) for each wallet address that interacts with it (so that the wallets can later give that password when querying the contract). How would we want to store all these passwords so that they can be checked when it is time for queries?

The Naive Approach

Our first approach might be to use a hashmap that stores all the passwords, and save it as binary using the methods we already know. So we would add the following lines to the init

```

Copy letpasswords:HashMap<&[u8],String>=HashMap::new();

save(&mutdeps.storage, PASSWORDS_KEY,&passwords)?;

```

And then we can proceed to load, and add each users' password whenever a user sends thecreate_password HandleMsg. Note that I'm assuming we would generate the&[u8] key for the hashmap from the user's wallet address. This approach has a huge problem! That is whenever we want to add another wallet's password to the hashmap, we must load the entire hashmap with all the passwords stored inside it. This will increase gas costs as we gain thousands of users. Moreover, loading the hashmap for queries will strain the node.

The Reasonable Approach

We now realize that we can actually generate&[u8] storage keys for each user wallet and save the password directly todeps.storage as described inStorage . This way, we can usemay_load to check if a user has a password, and if he does, learn what it is without loading all the other passwords. Then we would use the following lines of code to save a password

```

Copy letkey:&[u8]=env.message.sender.to_string().as_bytes(); save(&mutdeps.storage, key,&msg.password)?;

```

This method does work! However, what if we want to save additional things that is associated to each user, such as token balances. Then using save on the same key would overwrite our previous data. The solution is to add a prefix to the storage key so that we know it belongs to the password property. One way to implement this is the following:

```

Copy letprefix:String="passwords".to_string(); letkey:&[u8]=( prefix+env.message.sender.to_string()).as_bytes(); save(&mutdeps.storage, key,&msg.password)?;

```

This works but can be cumbersome and ugly, especially if you need to build more functionalities around this idea.This is exactly what prefixed storage does in the background! Prefixed Storage is built to solve this problem while hiding away all the ugliness.

What Is Prefixed Storage?

PrefixedStorage is a struct that keeps track of all the storage keys that are used to store various data under a shared namespace similar to describedabove . Prefixed storage is often used to keep track of storage keys ofdeps.storage , but it can in fact be used to store the keys of other storage structures.

The compiler only allows one mutable reference to the underlying storage to be valid at one point. Thus, you cannot have more than one mutable Prefixed Storage objects at the same time.

Example

Let's solvethe password problem that was mentioned above with prefixed storage. The solution to this problem resemblesviewing keys for permissioned viewing . You will learn more about this in the coming sections. We first define a namespace (prefix) instate.rs .

```

Copy pubconstPREFIX_PASSWORDS:&[u8]=b"passwords";

```

The idea then is that all the wrapper functions that we've learnt inStorage also work on prefixed storage. We will only demonstratesave andmay_load by example.

Saving To Prefixed Storage

We would instantiate a mutable PrefixedStorage object using the following lines of code to save a new password

```

Copy letmutpassword_store=PrefixedStorage::new(PREFIX_PASSWORDS,&mutdeps.storage); letkey:& [u8]=env.message.sender.to_string().as_bytes(); save(&mutpassword_store, key,&msg.password)?;

```

Notice that we are using the same storage wrapper functions fromStorage on prefixed storage. The reason why we can use the same wrapper functions is because Prefixed Storage implements the Storage trait, even though most of what it's doing is to add a prefix on top of the keys we provide to it.

Loading From Prefixed Storage

We may use bothload andmay_load We may load data from a mutable Prefixed Storage instance with the following lines of code

```

Copy letmutpassword_store=PrefixedStorage::new(PREFIX_PASSWORDS,&mutdeps.storage); letkey:& [u8]=env.message.sender.to_string().as_bytes(); // Throws error if there is no password saved before letpassword:String=load(&password_store, key)?;

```

In queries, we don't want to use PrefixedStorage struct. Instead we use ReadonlyPrefixedStorage struct. It works in the exact same way, except that you cannot save to it.

```

Copy letpassword_store=ReadonlyPrefixedStorage::new(PREFIX_PASSWORDS,&deps.storage); letkey=address.to_string().as_bytes(); letmay_password:Option=may_load(&password_store, key)?;

```

You can have as many ReadonlyPrefixedStorage objects as you want at the same time, unlike PrefixedStorage.

Removing From Prefixed Storage

We can use theremove wrapper function for this.

```

Copy letmutpassword_store=PrefixedStorage::new(PREFIX_PASSWORDS,&mutdeps.storage); letkey:& [u8]=env.message.sender.to_string().as_bytes(); remove(&mutpassword_store, key);

```

Last updated7 months ago On this page *Why Is Prefixed Storage Needed? * The Problem * What Is Prefixed Storage? * Example

Was this helpful? Edit on GitHub Export as PDF