Append Store

AppendStore is meant to replicate the functionality of an append list in a cosmwasm efficient manner. The length of the list is stored and used to pop/push items to the list. It also has a method to create a read only iterator.

This storage object also has the methodremove to remove a stored object from an arbitrary position in the list, but this can be extremely inefficient.

! Removing a storage object further from the tail gets increasingly inefficient. We recommend you usepop andpush whenever possible.

The same conventions fromItem also apply here, that is:

1. AppendStore has to be told the type of the stored objects. And the serde optionally.
2. Every methods needs it's own reference todeps.storage
3. .
4.

Initialize

To import and initialize this storage object as a static constant instate.rs , do the following:

```

Copy use secret_toolkit::storage::{AppendStore}

```

```

Copy pub static COUNT_STORE: AppendStore= AppendStore::new(b"count");

```

! Initializing the object as const instead of static will also work but be less efficient since the variable won't be able to cache length data.

Often times we need these storage objects to be associated to a user address or some other key that is variable. In this case, you need not initialize a completely new AppendStore insidecontract.rs . Instead, you can create a new AppendStore by adding a suffix to an already existing AppendStore. This has the benefit of preventing you from having to rewrite the signature of the AppendStore. For example

```

Copy // The compiler knows that user_count_store is AppendStorelet user_count_store =
COUNT_STORE.add_suffix(info.sender.to_string().as_bytes());

```

Sometimes when iterating these objects, we may want to load the nextn objects at once. This may be prefered if the objects we are iterating over are cheap to store or if we know that multiple objects will need to be accessed back to back. In such cases we may want to change the internal indexing size (default of 1). We do this instate.rs :

```

Copy pub static COUNT_STORE: AppendStore= AppendStore::new_with_page_size(b"count", 5);

```

Read/Write

The main user facing methods to read/write to AppendStore arepop ,push ,get_len ,set_at (which replaces data at a position within the length bound),clear (which deletes all data in the storage),remove (which removes an item in an arbitrary position, this is very inefficient). An extensive list of examples of these being used can be found inside the unit tests of AppendStore found inappend_store.rs .

Iterator

AppendStore also implements a readonly iterator feature. This feature is also used to create a paging wrapper method calledpaging . The way you create the iterator is:

```

Copy let iter = user_count_store.iter(&deps.storage)?;

```

More examples can be found in the unit tests. And the paging wrapper is used in the following manner:

```

Copy let start_page: u32 = 0; let page_size: u32 = 5; // The compiler knows that values is Vec<u32> let values = user_count_store.paging(&deps.storage, start_page, page_size)?;

```

Last updated7 months ago On this page Was this helpful?Edit on GitHub Export as PDF