Running the Application

Learn how to use SecretCLI to handle messages to query and modify contract state.

Contract Messages

The way you interact with contracts on a blockchain is by sendingcontract messages . A message contains the JSON description of a specific action that should be taken on behalf of the sender, and in most Rust smart contracts they are defined in themsg.rs file.

In ourmsg.rs file, there are two enums:ExecuteMsg , andQueryMsg . They are enums because each variant of them represents a different message which can be sent. For example, theExecuteMsg::Increment corresponds to thetry_increment message in ourcontract.rs file.

In the previous section, we compiled, uploaded and instantiated our counter smart contract. Now we are going to query the contract and also execute messages to update the contract state. Let's start by querying the counter contract we instantiated.

Query Message

AQuery Message is used to request information from a contract; unlikeexecute messages , query messages do not change contract state, and are used just like database queries. You can think of queries as questions you can ask a smart contract.

Let's query our counter smart contract to return the current count. It should be 1, because that was the count we instantiated in the previous section. We query the count by calling the Query Messageget_count {} , which is defined in our msg.rs file.

```
Copy secretcli query compute query secret18vd8fpwxzck93qlwghaj6arh4p7c5n8978vsyg '{"get_count": {}}'
```

The query returns:

```
Copy {"count":"1"}
```

Great! Now that we have queried the contract's starting count, let's execute anincrement{} message to modify the contract state.

Execute Message

AnExecute Message is used for handling messages which modify contract state. They are used to perform actual actions.

Did you know? Messages aren't free! Each transaction costs a small fee, which represents how many resources were required to complete it. This cost is measured ingas units. The counter contract consists of two execute messages:increment{} , which increments the count by 1, andreset{} , which resets the count to anyi32 you want. The current count is 1, let's call the Execute Messageincrement{} to increase the contract count by 1:

```
Copy secretcli tx compute execute secret18vd8fpwxzck93qlwghaj6arh4p7c5n8978vsyg '{"increment": {}}' --from myWallet
```

Pro Tip: SecretCLI automatically encrypts transactions. Only the transaction sender can see the data being sent (and the result). You can think of this as how HTTPS protects your data in transit when accessing a web page. SecretCLI will ask you to confirm the transaction before signing and broadcasting. Upon successful confirmation of the transaction, the terminal will return atransaction hash representing your transaction:

confirming a transaction using SecretCLI Nice work! Now we can query the contract once again to see if the contract state was successfully incremented by 1:

```
Copy secretcli query compute query secret18vd8fpwxzck93qlwghaj6arh4p7c5n8978vsyg '{"get_count": {}}'
```

The query returns:

```

Copy {"count":"2"}

```

Now, we will call one final execute message,reset{} . This will reset the count to ani32 that we specify. I am going to reset the count to 0 by running the following code in SecretCLI:

```

Copy secretcli tx compute execute secret18vd8fpwxzck93qlwghaj6arh4p7c5n8978vsyg '{"reset": {"count": 0}}' --from myWallet

```

Make sure your JSON message is formatted properly!

'{"reset": {"count": 0}}` The query returns:

```

Copy {"count":"0"}

```

Way to go! You have now successfully interacted with a Secret Network smart contract using SecretCLI.

Next Steps

Congratulations! You completed the tutorial and successfully compiled, uploaded, deployed and executed a Secret Contract! The contract is the business logic that powers a blockchain application, but a full application contains other components as well. If you want to learn more about Secret Contracts, or explore what you just did more in depth, feel free to explore these awesome resources:

- Secret University counter contract breakdown
-
    - explains the counter contract in depth
- Millionaire's problem breakdown
-
    - explains how a Secret Contract works
- CosmWasm Documentation
-
    - everything you want to know about CosmWasm
- Secret.JS
-
    - Building a web UI for a Secret Contract
-

Last updated8 months ago On this page *Contract Messages * Query Message * Execute Message * Next Steps

Was this helpful? Edit on GitHub Export as PDF