IBC-Hooks

Initiate a contract call with an incoming IBC token transfer using IBC hooks

Overview

IBC-Hooks is an IBC middleware thatuses incoming ICS-20 token transfers to initiate smart contract calls . This allows for arbitrary data to be passed in along with token transfers. This is useful for a variety of use cases such as cross-chain token swaps, auto-wrapping of SNIP-20 tokens, General Message Passing (GMP) between Secret Network and EVM chains, and much more! The mechanism enabling this is amemo field on every ICS20 transfer packet as of IBC v3.4.0. Wasm hooks is an IBC middleware that parses an ICS20 transfer, and if thememo field is of a particular form, it executes a Wasm contract call.

Note that the metadata in thememo field is not used within ICS-20 itself, but instead, a middleware or custom CosmWasm contract can wrap around the transfer protocol to parse the metadata and execute custom logic based off of it.See more here.

ICS20 Packet Structure

ICS20 is JSON native, so JSON is used for the memo format:

```

Copy { //... other ibc fields omitted for example "data":{ "denom":"denom on counterparty chain (e.g. uatom)",// will be transformed to the local denom (ibc/...) "amount":"1000", "sender":"addr on counterparty chain",// will be ignored and shown to the contract as a null sender (cannot be verifed over IBC) "receiver":"secret1contractAddr", "memo":{ "wasm":{ "contract":"secret1contractAddr", "msg":{ "raw_message_fields":"raw_message_data" } } } } }

```

An ICS20 packet is formatted correctly for Wasm hooks if the following all hold:

- memo
- is not blank
- memo
- is valid JSON
- memo
- has at least one key, with value"wasm"
- memo["wasm"]
- has exactly two entries,"contract"
- and"msg"
- memo["wasm"]["msg"]
- is a valid JSON object
- receiver == memo["wasm"]["contract"]
-

If an ICS20 packet is not directed towards wasmhooks, wasmhooks doesn't do anything. If an ICS20 packet is directed towards wasmhooks, and is formatted incorrectly, then wasmhooks returns an error.

ICS20 Packet Execution Flow

Before Wasm hooks:

- Ensure the incoming IBC packet is cryptographically valid
- Ensure the incoming IBC packet has not timed out
-

In Wasm hooks, before packet execution:

- Ensure the packet is correctly formatted (as defined above)
- Edit the receiver to be the hardcoded IBC module account
-

In Wasm hooks, after packet execution:

- Construct wasm message as defined above
- Execute wasm message
- If wasm message has error, returnErrAck
- Otherwise continue through middleware
-

Auto-wrapping of SNIP-20 Example

The following contract receives funds from IBC, wraps them as SNIP-20 tokens, and then transfers them to the recipient that is specified in the ibc-hooks message:

```

Copy

# [entry_point]

pubfnexecute(_deps:DepsMut, env:Env, info:MessageInfo, msg:Msg)->StdResult { matchmsg { Msg::WrapDeposit{ snip20_address, snip20_code_hash, recipient_address, }=>Ok(Response::default().add_messages(vec![ CosmosMsg::Wasm(cosmwasm_std::WasmMsg::Execute{ contract_addr:snip20_address.clone(), code_hash:snip20_code_hash.clone(), msg:to_binary(&secret_toolkit::snip20::HandleMsg::Deposit{ padding:None}) .unwrap(), funds:info.funds.clone(), }), CosmosMsg::Wasm(cosmwasm_std::WasmMsg::Execute{ contract_addr:snip20_address, code_hash:snip20_code_hash, msg:to_binary(&secret_toolkit::snip20::HandleMsg::Transfer{ recipient:recipient_address, amount:info.funds[0].amount, memo:None, padding:None, }) .unwrap(), funds:vec![], }), ])), } }

```

Ack callbacks

A contract that sends an IBC transfer may need to listen for the acknowledgment (ack) of that packet. To allow contracts to listen toack of specific packets, we provideAck callbacks . The sender of an IBC transfer packet may specify a callback in thememo field of the transfer packet when theack of that packet is received.

Only the IBC packet sender can set the callback

Ack callback implementation

For the callback to be processed, the transfer packet'smemo should contain the following in its JSON:

{"ibc_callback": "secret1contractAddr"}

The WASM hooks will keep the mapping from the packet's channel and sequence to the contract in storage. When anack is received, it will notify the specified contract via anexecute message.

Interface for receiving the Acks and Timeouts

The contract that awaits the callback should implement the following interface for a sudo message:

```

Copy

# [cw_serde]

pubenumIBCLifecycleComplete{

# [serde(rename="ibc_ack")]

IBCAck{ /// The source channel (secret side) of the IBC packet channel:String, /// The sequence number that the packet was sent with sequence:u64, /// String encoded version of the ack as seen by OnAcknowledgementPacket(..) ack:String, /// Weather an ack is a success of failure according to the transfer spec success:bool, },

# [serde(rename="ibc_timeout")]

IBCTimeout{ /// The source channel (secret side) of the IBC packet channel:String, /// The sequence number that the packet was sent with sequence:u64, }, }

/// Message type for sudo entry_point

# [cw_serde]
```

pubenumSudoMsg{

[serde(rename="ibc_lifecycle_complete")]

IBCLifecycleComplete(IBCLifecycleComplete), }

[entry_point]

pubfnsudo(_deps:DepsMut, _env:Env, msg:SudoMsg)->StdResult { matchmsg {
SudoMsg::IBCLifecycleComplete(IBCLifecycleComplete::IBCAck{ channel, sequence, ack, success, })=>todo!(),
SudoMsg::IBCLifecycleComplete(IBCLifecycleComplete::IBCTimeout{ channel, sequence, })=>todo!(), } }

```

Last updated5 days ago On this page *Overview * ICS20 Packet Structure * ICS20 Packet Execution Flow * Auto-wrapping of SNIP-20 Example * Ack callbacks

Was this helpful? Edit on GitHub Export as PDF
```