

# Messages

## Register Interchain Query

[MsgRegisterInterchainQuery](#) can be submitted by smart-contract only via [MsgRegisterInterchainQuery](#) transaction:

message

[MsgRegisterInterchainQuery](#)

```
{ // defines a query type: kv or tx now string query_type =
```

```
1 ;
```

```
// is used to define KV-storage keys for which we want to get values from remote chain repeated
```

```
KVKey keys =
```

```
2 ;
```

```
// is used to define a filter for transaction search ICQ string transactions_filter =
```

```
3 ;
```

```
// is IBC connection ID for getting ConsensusState to verify proofs string connection_id =
```

```
4 ;
```

```
// is used to specify how often (in neutron blocks) the query must be updated uint64 update_period =
```

```
5 ;
```

```
// is the signer of the message string sender =
```

```
6 ; }
```

message

[KVKey](#)

```
{ // Path (storage prefix) to the storage where you want to read value by key // (usually name of cosmos-sdk module: 'staking', 'bank', etc.) string path =
```

```
1 ; // Key you want to read from the storage bytes key =
```

```
2 ; } Note: the maximum allowed number of KVKey values for a single InterchainQuery equals to 32. Currently query_type can take the following values:
```

- kv
- - query values
- from Cosmos-SDK KV-storage on remote chain which are stored under some keys
- . In this case kv\_keys
- must be filled in.
- tx
- - query to search for transactions on remote chain.transactions\_filter
- describes a filter by which the [ICQ relayer](#)
- will perform the transactions search. [The transaction filter is described in more detail in the overview](#)
- :

[MsgRegisterInterchainQuery](#) returns [MsgRegisterInterchainQueryResponse](#) where id is unique identifier of newly registered interchain query on success:

message

[MsgRegisterInterchainQueryResponse](#)

```
{ uint64 id =
```

```
1 ; }
```

## State modifications

- increments last registered query id;
- generates new [RegisteredQuery](#)
- ;
- save the record in storage under incremented query id;

## Events

Emits [EventTypeNeutronMessage](#) with action equals query\_updated .

## Update Interchain Query

Note: as well as for query registration, for query updates the maximum allowed number of KVKey values for a single InterchainQuery equals to 32. [MsgUpdateInterchainQueryRequest](#) can be submitted only by the owner of corresponding Interchain Query:

message

MsgUpdateInterchainQueryRequest

{ uint64 query\_id =

1 ; repeated

KVKey new\_keys =

2 ; uint64 new\_update\_period =

3 ; string new\_transactions\_filter =

4 ; string sender =

5 ;

// is the signer of the message and owner of the corresponding ICQ } Returns just an empty [MsgUpdateInterchainQueryResponse](#) on success:

message

MsgUpdateInterchainQueryResponse

{ }

## State modifications

- [Updates](#)
- a corresponding RegisteredQuery
- structure.

## Events

Emits [EventTypeNeutronMessage](#) with action equals query\_updated .

## Remove Interchain Query

[MsgRemoveInterchainQueryRequest](#) can be submitted only by the owner of corresponding Interchain Query within the query's service period or by anyone beyond it. Read more about this message permissions [here](#) .

message

MsgRemoveInterchainQueryRequest

{ uint64 query\_id =

1 ; string sender =

2 ;

// is the signer of the message and the owner of corresponding ICQ } Returns just an

empty [MsgRemoveInterchainQueryResponse](#) on success:

message

MsgRemoveInterchainQueryResponse

{ }

### State modifications

- [Removes](#)
- a correspondingRegisteredQuery
- structure.
- Also removes the query results ([immediately](#)
- for a KV query [deferred in the ICQ module EndBlock](#)
- for a TX query).

### Events

Emits [EventTypeNeutronMessage](#) with action equals query\_removed .

### Submit Query Result

[MsgSubmitQueryResult](#) can be submitted by any Neutron account via MsgSubmitQueryResult transaction:

message

MsgSubmitQueryResult

{ uint64 query\_id =

1 ; string sender =

2 ;

// is the IBC client ID for an IBC connection between Neutron chain and target chain (where the result was obtained from)

string client\_id =

3 ; QueryResult result =

4 ; }

message

QueryResult

{ repeated

StorageValue kv\_results =

1 ; Block block =

2 ; uint64 height =

3 ; uint64 revision =

4 ; bool allow\_kv\_callbacks =

5 ; }

message

StorageValue

{ // is the substore name (acc, staking, etc.) string storage\_prefix =

1 ;

// is the key in IAVL store bytes key =

2 ;

```

// is the value in IAVL store bytes value =
3 ;

// is the Merkle Proof which proves existence of key-value pair in IAVL storage tendermint . crypto . ProofOps Proof =
4 ; }

message

Block

{ // We need to know block X+1 to verify response of transaction for block X // since LastResultsHash is root hash of all
results from the txs from the previous block google . protobuf . Any next_block_header =

1 ;

// We need to know block X to verify inclusion of transaction for block X google . protobuf . Any header =

2 ;

TxValue tx =

3 ; }

message

TxValue

{ tendermint . abci . ResponseDeliverTx response =

1 ;

// is the Merkle Proof which proves existence of response in block with height next_block_header.Height tendermint . crypto .
Proof delivery_proof =

2 ;

// is the Merkle Proof which proves existence of data in block with height header.Height tendermint . crypto . Proof
inclusion_proof =

3 ;

// is body of the transaction bytes data =

4 ; } Returns just an empty MsgSubmitQueryResultResponse on success:

message

MsgSubmitQueryResultResponse

{ }

```

### State modifications

- get registered interchain query info byMsgSubmitQueryResult.query\_id
- ;
- for everyresult
- inMsgSubmitQueryResult.result.kv\_results
- : \* read IBC connection consensus state from IBC keeper storage withregistered\_query.ConnectionID
- - ,MsgSubmitQueryResult.result.revision
- - ,MsgSubmitQueryResult.result.height+1
- - ;
- - verifyresult.Proof
- - with Merkle Root Hash from consensus state;
- saveMsgSubmitQueryResult.result.kv\_results
- to the storage: \* clearMsgSubmitQueryResult.result

- - from the proofs, Neutron doesn't need them anymore;
- - save cleared result to storage with `keyregistered_query.id`
- - ;
- - `setregistered_query.last_submitted_result_remote_height`
- - `toresult.height`
- - ;
- - `setregistered_query.last_submitted_result_local_height`
- - to the current Neutron height;
- `callbackMsgSubmitQueryResult.result.kv_results`
- to the appropriate smart-contract if needed;
- for every block
- `inMsgSubmitQueryResult.result.blocks`
- `:* verifyblock.next_block_header`
- - `andblock.header`
- - by calling [clientKeeper.UpdateClient\(header\)](#)
- - ;
- - [verify](#)
- - `block.txs`
- - with verified headers;
- `process everytransaction`
- in every block
- `fromMsgSubmitQueryResult.result.blocks`
- `:* check`
- - transaction was not processed previously to avoid double submitting
- - save generated record to the storage with composite key `bigEndianBytes(registered_query.id) + bigEndianBytes(last_submitted_transaction_id)`
- - prefixed by [SubmittedTxKey](#)
- - ;
- - [callback](#)
- - transaction to the appropriate smart-contract;
- - [save](#)
- - transaction's hash to the storage to approach double-submission preventing mechanics. [Previous Overview Next Client](#)