# #

Service

Service module allows you to define, bind, invoke services on the IRIS HubRead more about iService .

#

Available Commands

Name Description define Define a new servicedefinition Query a service definition bind Bind a service binding Query a service binding bindings Query all bindings of a service definitionset-withdraw-addr Set a withdrawal address for a provider withdraw-addr Query the withdrawal address of a providerupdate-binding Update an existing service binding disable Disable an available service binding enable Enable an unavailable service binding refund-deposit Refund all deposit from a service binding call Initiate a service call request Query a request by the request IDrequests Query active requests by the service binding or request context ID respond Respond to a service requestresponse Query a response by the request ID responses Query active responses by the request context ID and batch counterrequest-context Query a request context update Update a request context pause Pause a running request contextstart Start a paused request contextkill Terminate a request context fees Query the earned fees of a providerwithdraw-fees Withdraw the earned fees of a providerschema Query the system schema by the schema name params Query values set as service parameters.

#

iris tx service define

Define a new service.

iris txservice define[flags] Flags:

Name, shorthand Default Description Required --name

Service name Yes --description

Service description --author-description

Service author description --tags

Service tags --schemas

Content or file path of service interface schemas Yes

#

define a service

iris txservice define\ --name = < service name> \ --description = < service description> \ --author-description= < author description> --tags = tag1,tag2\ --schemas = < schemas content or path/to/schemas.json> \ --chain-id= irishub\ --from = < key-name> \ --fees = 0 .3iris

#

Schemas content example

{ "input" : { "schema" : "http://json-schema.org/draft-04/schema#" , "title" : "BioIdentify service input body" , "description" : "BioIdentify service input body specification" , "type" : "object" , "properties" : { "id" : { "description" : "id" , "type" : "string" } , "name" : { "description" : "name" , "type" : "string" } , "data" : { "description" : "data" , "type" : "string" } } , "required" : ["id" , "data"] } , "output" : { "schema" : "http://json-schema.org/draft-04/schema#" , "title" : "BioIdentify service output body" , "description" : "BioIdentify service output body specification" , "type" : "object" , "properties" : { "data" : { "description" : "result data" , "type" : "string" } } , "required" : ["data"] } }

#

iris query service definition

Query a service definition.

iris queryservice definition[service-name] [flags]

#

Query a service definition

Query the detailed info of the service definition with the specified service name.

iris queryservice definition< service name>

#

iris tx service bind

Bind a service.

iris txservice bind [flags] Flags:

Name, shorthand Default Description Required --service-name

Service name Yes --deposit

Deposit of the binding Yes --pricing

Pricing content or file path, which is an instance ofIrishub Service Pricing JSON Schema Yes --qos

Minimum response time Yes --options

Non-functional requirements options Yes --provider

Provider address, default to the owner

#

Bind an existing service definition

The deposit needs to satisfy the minimum deposit requirement, which is the maximal one betweenprice *MinDepositMultiple andMinDeposit (MinDepositMultiple andMinDeposit are the system parameters, which can be modified through the governance).

iris txservice bind \ --service-name= < service name> \ --deposit = 10000iris\ --pricing = < pricing content or path/to/pricing.json> \ --qos = 50 \ --options = < non-functional requirements options content or path/to/options.json> \ --chain-id= irishub\ --from = < key-name> \ --fees = 0 .3iris

#

Pricing content example

{ "price" : "1iris" }

#

iris query service binding

Query a service binding.

iris queryservice binding< service name> < provider>

#

iris query service bindings

Query all bindings of a service definition.

iris queryservice bindings[service-name] [flags]

#

Query service binding list

iris queryservice bindings< service name> < owner address>

#

iris tx service update-binding

Update a service binding.

```
iris txservice update-binding[ service-name] [ provider-address] [ flags] Flags:
```

Name, shorthand Default Description Required --deposit

Deposit added for the binding, not updated if empty --pricing

Pricing content or file path, which is an instance of Irishub Service Pricing JSON Schema , not updated if empty --qos

Minimum response time, not updated if set to 0 --options

Non-functional requirements options

#

Update an existing service binding

The following example updates the service binding with the additional 10 IRIS deposit

```
iris txservice update-binding< service-name> < provider-address> \ --deposit = 10iris\ --options = < non-functional requirements options content or path/to/options.json> \ --pricing = '{"price":"1iris"}' \ --qos = 50 \ --chain-id= < chain-id> \ --from = < key name> \ --fees = 0 .3iris
```

#

iris tx service set-withdraw-addr

Set a withdrawal address for a provider.

```
iris txservice set-withdraw-addr[ withdrawal-address] [ flags]
```

#

iris query service withdraw-addr

Query the withdrawal address of a provider.

```
iris queryservice withdraw-addr[ provider] [ flags]
```

#

iris tx service disable

Disable an available service binding.

```
iris txservice disable[ service-name] [ provider-address] [ flags]
```

#

iris tx service enable

Enable an unavailable service binding.

```
iris txservice enable [ service-name] [ provider-address] [ flags] Flags:
```

Name, shorthand Default Description Required --deposit

deposit added for enabling the binding

#

Enable an unavailable service binding

The following example enables an unavailable service binding with the additional 10 IRIS deposit.

```
iris txservice enable < service name> < provider-address> --chain-id= irishub--from = < key-name> --fees = 0 .3iris--deposit = 10iris
```

# #

iris tx service refund-deposit

Refund all deposits from a service binding.

```
iris txservice refund-deposit[ service-name] [ provider-address] [ flags]
```

#

Refund all deposits from an unavailable service binding

Before refunding, you shoulddisable the service binding first.

```
iris txservice refund-deposit< service name> < provider-address> --chain-id= irishub--from = < key-name> --fees = 0 .3iris
```

# #

iris tx service call

Initiate a service call.

```
iris txservice call[ flags] Flags:
```

Name, shorthand Default Description Required --service-name

Service name Yes --providers

Provider list to request Yes --service-fee-cap

Maximum service fee to pay for a single request Yes --data

Content or file path of the request input, which is an Input JSON Schema instance Yes --timeout

Request timeout Yes --repeated false Indicate if the reqeust is repetitive (Temporarily disabled in irishub-v1.0.0, will be activated after a few versions) --frequency

Request frequency when repeated, default totimeout --total

Request count when repeated, -1 means unlimited

#

Initiate a service invocation request

```
iris txservice call\ --service-name= < service name> \ --providers = < provider list> \ --service-fee-cap= 1iris\ --data = < request input or path/to/input.json> \ --timeout = 100 \ --repeated \ --frequency = 150 \ --total = 100 \ --chain-id= irishub\ --from = < key name> \ --fees = 0 .3iris
```

#

Input example

```
{ "header" : { ...} , "body" : { "id" : "1" , "name" : "irisnet" , "data" : "facedata" } }
```

# #

iris query service request

Query a request by the request ID.

```
iris queryservice request[ request-id] [ flags]
```

#

Query a service request

iris queryservice request< request-id> TIP

You can retrieve therequest-id inQuery request_id through rpc interface or iris query service requests.

#

Query request_id through rpc interface

Queryblock_results according toblock height throughrpc interface , findnew_batch_request_provider inend_block_events , decode the result with base64 to getrequest_id .

curl -X POST-d '{"jsonrpc":"2.0","id":1,"method":"block_results","params":["10604"]}' http://localhost:26657

#

iris query service requests

Query active requests by the service binding or request context ID.

iris queryservice requests[service-name] [provider] | [request-context-id] [batch-counter] [flags]

#

Query active requests of a service binding

iris queryservice requests< service name> < provider>

#

Query service requests by the request context ID and batch counter

iris queryservice requests< request-context-id> < batch-counter>

#

iris tx service respond

Respond to a service request.

iris txservice respond[flags] Flags:

Name, shorthand Default Description Required --request-id

ID of the request to respond to Yes --result

Content or file path of the response result, which is an instance ofirishub Service Result JSON Schema Yes --data

Content or file path of the response output, which is an Output JSON Schema instance

#

Respond to a service request

iris txservice respond\ --request-id= < request-id> \ --result = < response result or path/to/result.json> \ --data = < response output or path/to/output.json> --chain-id= irishub\ --from = < key-name> \ --fees = 0 .3iris TIP

You can retrieve therequest-id inQuery request_id through rpc interface or iris query service requests.

#

Result example

{ "code" : 200 , "message" : "" }

#

Output example

{ "header" : { ...} , "body" : { "data" : "userdata" } }

#

iris query service response

Query a service response.

iris queryservice response[request-id] [flags] TIP

You can retrieve therequest-id inQuery request_id through rpc interface or iris query service requests.

#

iris query service responses

Query active responses by the request context ID and batch counter.

iris queryservice responses[request-context-id] [batch-counter] [flags]

#

Query responses by the request context ID and batch counter

iris queryservice responses< request-context-id> < batch-counter>

#

iris query service request-context

Query a request context.

iris queryservice request-context[request-context-id] [flags]

#

Query a request context

iris queryservice request-context< request-context-id> TIP

You can retrieve therequest-context-id in the result ofservice call

#

iris tx service update

Update a request context.

iris txservice update[request-context-id] [flags] Flags:

Name, shorthand Default Description Required --providers

Provider list to request, not updated if empty --service-fee-cap

Maximum service fee to pay for a single request, not updated if empty --timeout

Request timeout, not updated if set to 0 --frequency

Request frequency, not updated if set to 0 --total

Request count, not updated if set to 0

#

Update a request context

iris txservice update< request-context-id> \ --providers = < provider list> \ --service-fee-cap= 1iris\ --timeout = 0 \ --frequency = 150 \ --total = 100 \ --chain-id= irishub\ --from = < key name> \ --fees = 0 .3iris

#

iris tx service pause

Pause a running request context.

```
iris txservice pause[ request-context-id] [ flags]
```

#

Pause a running request context

```
iris txservice pause< request-context-id>
```

#

iris tx service start

Start a paused request context.

```
iris txservice start[ request-context-id] [ flags]
```

#

Start a paused request context

```
iris txservice start< request-context-id>
```

#

iris tx service kill

Terminate a request context.

```
iris txservice kill [ request-context-id] [ flags]
```

#

Kill a request context

```
iris txservice kill < request-context-id>
```

#

iris query service fees

Query the earned fees of a provider.

```
iris queryservice fees[ provider] [ flags]
```

#

iris tx service withdraw-fees

Withdraw the earned fees of a provider.

```
iris txservice withdraw-fees[ provider-address] [ flags]
```

#

iris query service schema

Query the system schema by the schema name, only pricing and result allowed.

```
iris queryservice schema[ schema-name] [ flags]
```

#

Query the service pricing schema

```
iris queryservice schema pricing
```

#

Query the response result schema

```
iris queryservice schema result
```

#

iris query service params

Query values set as service parameters.

```
iris queryservice params[ flags]
```