Getting Started

There is a basic demo interacting with theQueryDemo contract hosted athttps://vaa.dev/#/ccq

To get started, we will look at a simpleeth_call request to get the total supply of WETH on Ethereum.

RPC Basics

Before we dig into anything Queries specific, let’s look at how to make aneth_call against a public Ethereum RPC. Before we can make a request, we need some information about the contract we want to call.

- to
- : the contract to call
-
    - WETH is0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2
- *
- data
- : the method identifier and ABI-encoded parameters
-
    - web3.eth.abi.encodeFunctionSignature("totalSupply()")
-
    - →0x18160ddd
- *
- block id
- : the block number, hash, or tag, likelatest
- ,safe
- , orfinalized
-

```

Copy // Request curl https://ethereum.publicnode.com -X POST --data '{"jsonrpc":"2.0","method":"eth_call","params":[{"to":"0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2","data":"0x18160ddd"},"latest"],"id":1}' // Result { "jsonrpc":"2.0", "id":1, "result":"0x000000000000000000000000000000000000000000029fd3d129b582d7949e71" }

```

Converting that result from hex gets us3172615244782286193073777 . You can compare your result to theRead Contract tab in Etherscan. Your result will be different as WETH is minted/burned over time.

Construct a Query

For this part, we will use theWormhole Query SDK along withaxios for our RPC requests.

```

Copy npm i @wormhole-foundation/wormhole-query-sdk axios

```

In order to make anEthCallQueryRequest , we need a specific block number or hash as well as the call data to request.

To get the latest block, we can request it from a public node usingeth_getBlockByNumber .

```

Copy constrpc="https://ethereum.publicnode.com"; constlatestBlock:string=( awaitaxios.post(rpc,{ method:"eth_getBlockByNumber", params:["latest",false], id:1, jsonrpc:"2.0", }) ).data?.result?.number;

```

Then construct the call data

```

Copy constcallData:EthCallData={ to:"0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2",// WETH data:"0x18160ddd",// web3.eth.abi.encodeFunctionSignature("totalSupply()") };

```

Finally, put it all together in aQueryRequest

```
Copy constrequest=newQueryRequest( 0,// nonce [ newPerChainQueryRequest( 2,// Ethereum Wormhole Chain ID newEthCallQueryRequest(latestBlock,[callData]) ), ] );
```

This request consists of onePerChainQueryRequest , which is anEthCallQueryRequest to Ethereum. You can log this out as JSON to see the structure.

```
Copy console.log(JSON.stringify(request,undefined,2)); // { // "nonce": 0, // "requests": [ // { // "chainId": 2, // "query": { // "callData": [ // { // "to": "0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2", // "data": "0x18160ddd" // } // ], // "blockTag": "0x11e9068" // } // } // ], // "version": 1 // }
```

Mock a Query

For easier testing, the Query SDK provides aQueryProxyMock which will perform the request and sign the result with thedevnet guardian key. Themock call returns the same format as the Query Proxy.

```
Copy constmock=newQueryProxyMock({2:rpc }); constmockData=awaitmock.mock(request); console.log(mockData); // { // signatures: ['...'], // bytes: '...' // }
```

This response is suited for on-chain use, but the SDK also includes a parser so we can read the results in the client.

```
Copy constmockQueryResponse=QueryResponse.from(mockData.bytes); constmockQueryResult=( mockQueryResponse.responses[0].responseasEthCallQueryResponse ).results[0]; console.log( Mock Query Result: {mockQueryResult}({BigInt(mockQueryResult)}) ); // Mock Query Result: 0x0000000000000000000000000000000000000000000029fd09d4d81addb3ccfee (3172556167631284394053614)
```

Testing this all together might look like the following

```
Copy import{ EthCallData, EthCallQueryRequest, EthCallQueryResponse, PerChainQueryRequest, QueryProxyMock, QueryRequest, QueryResponse, }from"@wormhole-foundation/wormhole-query-sdk"; importaxiosfrom"axios";

constrpc="https://ethereum.publicnode.com"; constcallData:EthCallData={ to:"0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2",// WETH data:"0x18160ddd",// web3.eth.abi.encodeFunctionSignature("totalSupply()") };

(async()=>{ constlatestBlock:string=( awaitaxios.post(rpc,{ method:"eth_getBlockByNumber", params:["latest",false], id:1, jsonrpc:"2.0", }) ).data?.result?.number; if(!latestBlock) { console.error(× Invalid block returned); return; } console.log("Latest Block: ",latestBlock,({BigInt(latestBlock)})); consttargetResponse=awaitaxios.post(rpc,{ method:"eth_call", params:[callData,latestBlock], id:1, jsonrpc:"2.0", }); // console.log(finalizedResponse.data); if(targetResponse.data.error) { console.error(×{targetResponse.data.error.message}); } consttargetResult=targetResponse.data?.result; console.log("Target Result: ",targetResult,({BigInt(targetResult)})); // form the query request constrequest=newQueryRequest( 0,// nonce [ newPerChainQueryRequest( 2,// Ethereum Wormhole Chain ID newEthCallQueryRequest(latestBlock,[callData]) ), ] ); // console.log(JSON.stringify(request, undefined, 2)); constmock=newQueryProxyMock({2:rpc }); constmockData=awaitmock.mock(request); // console.log(mockData); constmockQueryResponse=QueryResponse.from(mockData.bytes); constmockQueryResult=( mockQueryResponse.responses[0].responseasEthCallQueryResponse ).results[0]; console.log( Mock Query Result: {mockQueryResult}({BigInt(mockQueryResult)}) ); })();
```

Fork Testing

It is common to test against a local fork of mainnet with something like

```
```

Copy anvil--fork-url https://ethereum.publicnode.com

```
```

In order for mock requests to verify against the mainnet Core bridge contract, we need to replace the current guardian set with the single devnet key used by the mock.

Here's an example for Ethereum mainnet, where the-a parameter is the[Core bridge address](#) on that chain.

```
```

Copy npx @wormhole-foundation/wormhole-cli evm hijack-a0x98f3c9e6E3fAce36bAAd05FE09d375Ef1464288B-g0xbeFA429d57cD18b7F8A4d91A2da9AB4AF05d0FBe

```
```

If you are usingEthCallWithFinality , you will need to mine additional blocks (32, on Anvil) after the latest transaction in order for it to become finalized. Anvil supports[auto-mining](#) with the-b flag if you want to test code that waits naturally for the chain to advance. For integration tests, you may want to simplyanvil_mine with0x20 .

Make a QueryRequest

```
```

Copy constserialized=request.serialize(); constproxyResponse= (awaitaxios.post)< QueryProxyQueryResponse>(QUERY_URL, { bytes:Buffer.from(serialized).toString("hex"), }, { headers:{"X-API-Key":YOUR_API_KEY} });

```
```

A testnet Query Proxy is available athttps://testnet.ccq.vaa.dev/v1/query

A mainnet Query Proxy is available athttps://api.wormholelabs.xyz/v1/query

Verify a QueryResponse On-Chain

A[QueryResponse abstract contract](#) is provided to assist with verifying query responses. Broadly, using a query response on-chain comes down to 3 steps.

1. Parse and verify the query response.
2.
    1. TheparseAndVerifyQueryResponse
3.
    1. handles verifying the Guardian signatures against the current guardian set stored in the Core bridge contract.
4. 3.
5. Validate the request details. This may be different for every integrator depending on their use case, but generally checks the following.
6.
    1. Is the request against the expected chain?
7.
    1. Is the request of the expected type? TheparseEthCall*
8.
    1. helpers perform this check when parsing.
9.
    1. Is the resulting block number and time expected? Some consumers might require that a block number be higher than the last, or the block time be within the last 5 minutes.validateBlockNum
10.
    1. andvalidateBlockTime
11.
    1. can help with the checks.
12.
    1. Is the request for the expected contract and function signature? ThevalidateMultipleEthCallData
13.
    1. can help with non-parameter-dependent cases.
14.
    1. Is the result of the expected length for the expected result type?
15. 10.
16. abi.decode
17. the result.
18.

See the[QueryDemo](#) contract for an example and read the docstrings of the above methods for detailed usage instructions.

Submit a QueryResponse On-Chain

TheQueryProxyQueryResponse result requires a slight tweak when submitting to the contract to match the format offunction parseAndVerifyQueryResponse(bytes memory response, IWormhole.Signature[] memory signatures) . A helper function,signaturesToEvmStruct , is provided in the SDK for this.

For example, submitting the transaction to the demo contract:

```

Copy consttx=awaitcontract.updateCounters( 0x{response.data.bytes}, signaturesToEvmStruct(response.data.signatures) );

```

Last updated1 month ago

Was this helpful? Edit on GitHub