Hello Wormhole

This tutorial contains a solidity contract (HelloWormhole.sol) that can be deployed onto many EVM chains to form a fully functioning cross-chain application.

Specifically, we will write and deploy a contract onto many chains that allows users to request, from one contract, that aGreetingReceived event be emitted from a contracts on adifferent chain .

This also allows users to pay for their custom greeting to be emitted on a chain that they do not have any gas funds for!

Getting Started

Included in therepository is:

- Example Solidity Code
- Example Forge local testing setup
- Testnet Deploy Scripts
- Example Testnet testing setup
-

Environment Setup

- Node 16.14.1 or later, npm 8.5.0 or later:https://docs.npmjs.com/downloading-and-installing-node-js-and-npm
- forge 0.2.0 or later:https://book.getfoundry.sh/getting-started/installation
-

Testing Locally

Pull the code down from github and cd into the directory, then build and test it.

```

Copy gitclonehttps://github.com/wormhole-foundation/hello-wormhole.git cdhello-wormhole npmrunbuild forgetest

```

Expected output is

```

Copy Running1testfortest/HelloWormhole.t.sol:HelloWormholeTest [PASS]testGreeting()(gas:777229)
Testresult:ok.1passed;0failed;finishedin3.98s

```

Deploying to Testnet

You will need a wallet with at least 0.05 Testnet AVAX and 0.01 Testnet CELO.

- Obtain testnet AVAX here
- Obtain testnet CELO here
-

```

Copy EVM_PRIVATE_KEY=your_wallet_private_keynpmrundeploy

```

Testing on Testnet

You will need a wallet with at least 0.02 Testnet AVAXObtain testnet AVAX here

You must have also deployed contracts onto testnet (as described in the above section).

To test sending and receiving a message on testnet, execute the test as such:

```

Copy EVM_PRIVATE_KEY=your_wallet_private_keynpmruntest
```

```

Explanation of the HelloWormhole Cross-chain Contract

Let’s take a simple HelloWorld solidity application, and take it cross-chain!

Single-chain HelloWorld solidity contract

This single-chain HelloWorld smart contract allows users to send greetings. In other words, it allows them to cause an eventGreetingReceived to be emitted with their greeting!

```

Copy // SPDX-License-Identifier: UNLICENSED pragmasolidity^0.8.13;

contractHelloWorld{ eventGreetingReceived(stringgreeting,addresssender);

string[]publicgreetings;

/ *@noticeReturns the cost (in wei) of a greeting*/ functionquoteGreeting()publicviewreturns(uint256cost) { return0; }

/ *@noticeUpdates the list of 'greetings' * and emits a 'GreetingReceived' event with 'greeting*/ functionsendGreeting(stringmemorygreeting)publicpayable{ uint256cost=quoteGreeting(); require(msg.value==cost); emitGreetingReceived(greeting,msg.sender); greetings.push(greeting); } }

```

Taking HelloWorld cross-chain using Wormhole Automatic Relayers

Suppose we want users to be able to request, through their Ethereum wallet, that a greeting be sent to Avalanche, and vice versa.

Let us begin writing a contract that we can deploy onto Ethereum, Avalanche, or any number of other chains, to enable greetings be sent freely between each contract, irrespective of chain.

We'll want to implement the following function:

```

Copy / *@noticeUpdates the list of 'greetings' * and emits a 'GreetingReceived' event with 'greeting' * on the HelloWormhole contract at * chain 'targetChain' and address 'targetAddress'* / functionsendCrossChainGreeting(uint16targetChain, addresstargetAddress, stringmemorygreeting)publicpayable;

```

The Wormhole Relayer contract lets us do exactly this! Let’s take a look at the Wormhole Relayer contract interface.

```

Copy / *@noticePublishes an instruction for the default delivery provider * to relay a payload to the addresstargetAddress on chain targetChain * with gas limitgasLimit and msg.value equal to receiverValue * * targetAddress must implement the IWormholeReceiver interface * * This function must be called with msg.value equal to quoteEVMDeliveryPrice(targetChain, receiverValue, gasLimit) * * Any refunds (from leftover gas) will be paid to the delivery provider. In order to receive the refunds, use the sendPayloadToEvm function * with refundChain and refundAddress as parameters **@paramtargetChain in Wormhole Chain ID format *@paramtargetAddress address to call on targetChain (that implements IWormholeReceiver)*@parampayload arbitrary bytes to pass in as parameter in call to targetAddress *@paramreceiverValue msg.value that delivery provider should pass in for call to targetAddress (in targetChain currency units)*@paramgasLimit gas limit with which to calltargetAddress. *@returnsequence sequence number of published VAA containing delivery instructions*/ functionsendPayloadToEvm(uint16targetChain, addresstargetAddress, bytesmemorypayload, uint256receiverValue, uint256gasLimit)externalpayablereturns(uint64sequence);

```

The Wormhole Relayer network is powered byDelivery Providers , who perform the service of watching for Wormhole Relayer delivery requests and performing the delivery to the intended target chain as instructed.

In exchange for calling your contract attargetAddress ontargetChain and paying the gas fees that your contract consumes, they charge a source chain fee. The fee charged will depend on the conditions of the target network and the fee can be requested from the delivery provider:
```

```
Copy (deliveryPrice,) = quoteEVMDeliveryPrice(targetChain, receiverValue, gasLimit)
```

So, following this interface, we can implementsendCrossChainGreeting by simply calling sendPayloadToEvm with the payload being some information we'd like to send, such as the greeting and the sender of the greeting.

```
Copy uint256constantGAS_LIMIT=50_000;

IWormholeRelayerpublicimmutablewormholeRelayer;

/ @noticeReturns the cost (in wei) of a greeting/ functionquoteCrossChainGreeting( uint16targetChain )publicviewreturns(uint256cost) { // Cost of requesting a message to be sent to // chain 'targetChain' with a gasLimit of 'GAS_LIMIT' (cost,)=wormholeRelayer.quoteEVMDeliveryPrice( targetChain, 0, GAS_LIMIT ); }

/ @noticeUpdates the list of 'greetings' * and emits a 'GreetingReceived' event with 'greeting' * on the HelloWormhole contract at * chain 'targetChain' and address 'targetAddress' / functionsendCrossChainGreeting( uint16targetChain, addresstargetAddress, stringmemorygreeting )publicpayable{ bytesmemorypayload=abi.encode(greeting,msg.sender); uint256cost=quoteCrossChainGreeting(targetChain); require(msg.value==cost,"Incorrect payment"); wormholeRelayer.sendPayloadToEvm{value:cost}( targetChain, targetAddress, payload, 0,// no receiver value needed GAS_LIMIT ); }
```

A key part of this system, though, is that the contract at thetargetAddress must implement theIWormholeReceiver interface.

Since we want to allow sending and receiving messages by theHelloWormhole contract, we must implement this interface.

```
Copy // SPDX-License-Identifier: Apache 2

pragmasolidity^0.8.0;

/ @noticeInterface for a contract which can receive Wormhole messages./ interfaceIWormholeReceiver{ / @noticeWhen a send is performed with this contract as the target, this function will be * invoked by the WormholeRelayer contract * NOTE: This function should be restricted such that only the Wormhole Relayer contract can call it. * * We also recommend that this function checks that sourceChain and sourceAddress are indeed who * you expect to have requested the calling of send on the source chain * * The invocation of this function corresponding to the send request will have msg.value equal * to the receiverValue specified in the send request. * * If the invocation of this function reverts or exceeds the gas limit * specified by the send requester, this delivery will result in a ReceiverFailure. * @parampayload - an arbitrary message which was included in the delivery by the * requester. @paramadditionalVaas - Additional VAAs which were requested to be included in this delivery. * They are guaranteed to all be included and in the same order as was specified in the * delivery request. @paramsourceAddress - the (wormhole format) address on the sending chain which requested * this delivery. @paramsourceChain - the wormhole chain ID where this delivery was requested.@paramdeliveryHash - the VAA hash of the deliveryVAA. * NOTE: These signedVaas are NOT verified by the Wormhole core contract prior to being provided * to this call. Always make sure parseAndVerify() is called on the Wormhole core contract * before trusting the content of a raw VAA, otherwise the VAA may be invalid or malicious. */ functionreceiveWormholeMessages( bytesmemorypayload, bytes[]memoryadditionalVaas, bytes32sourceAddress, uint16sourceChain, bytes32deliveryHash )externalpayable; }
```

AftersendPayloadToEvm is called on the source chain, the off-chain Delivery Provider will pick up the VAA corresponding to the message. It will then call thereceiveWormholeMessages method on thetargetChain andtargetAddress specified.

So, in receiveWormholeMessages, we want to:

1. Update the latest greeting
2. Emit a 'GreetingReceived' event with the 'greeting' and sender of the greeting
3.

Note: It is crucial that only the Wormhole Relayer contract can call receiveWormholeMessages

To provide certainty about the validity of the payload, we must restrict the msg.sender of this function to only be the Wormhole Relayer contract. Otherwise, anyone could call this receiveWormholeMessages endpoint with fake greetings, source chains, and source senders.

And voila, we have a full contract that can be deployed to many EVM chains, and in totality would form a full cross-chain application powered by Wormhole!

Users with any wallet can request greetings to be emitted on any chain that is part of the system.

How does it work?

Check out Part 2 for an in-depth explanation of how Wormhole Relayer causes contracts on other blockchains to be called with the appropriate inputs!

Full Cross-chain HelloWormhole solidity contract

See the full implementation of the HelloWormhole.sol contract and the full Github repository with testing infrastructure

Wormhole integration complete?

Let us know so we can list your project in our ecosystem directory and introduce you to our global, multichain community!

Reach out now!

Last updated1 month ago

Was this helpful? Edit on GitHub