Beyond Hello Wormhole

In Part 1 (HelloWormhole), we wrote a fully functioning cross-chain application that allows users to request, from one contract, that aGreetingReceived event to be emitted from one of the other contracts on a different chain.

In Part 2 (How does Hello Wormhole Work?), we discussed how the Wormhole Relayer contract works behind the scenes. In summary, it works by publishing a wormhole message with delivery instructions, which alerts a delivery provider to call thedeliver endpoint of the Wormhole Relayer contract on the target chain, finally calling the designatedtargetAddress with the correct inputs

HelloWormhole is a great example application, but has much room for improvement. Let's talk through some ways to improve both the security and features of the application!

Topics covered:

- Restricting the sender
- Refunds
- Chained Deliveries
- Delivering existing VAAs
-

Protections

Issue: The greetings can come from anyone

A user doesn't have to go through the HelloWormhole contract to request a greeting - they can callwormholeRelayer.sendPayloadToEvm{value: cost}(…) themselves!

This is not ideal if, for example, you wanted to store some information in the source HelloWormhole contract every time asendCrossChainGreeting was requested.

Often, it is desirable that all of the requests go through your own source contract.

Solution: We can check, in our implementation ofreceiveWormholeMessages , thatsourceChain andsourceAddress are a valid HelloWormhole contract, and revert otherwise

```

Copy addressregistrationOwner; mapping(uint16=>bytes32) registeredSenders;

modifierisRegisteredSender(uint16sourceChain,bytes32sourceAddress) {
require(registeredSenders[sourceChain]==sourceAddress,"Not registered sender"); _; }

/* * Sets the registered address for 'sourceChain' to 'sourceAddress' * So that for messages from 'sourceChain', only ones from 'sourceAddress' are valid * * Assumes only one sender per chain is valid * Sender is the address that called 'send' on the Wormhole Relayer contract on the source chain) /
functionsetRegisteredSender(uint16sourceChain,bytes32sourceAddress)public{
require(msg.sender==registrationOwner,"Not allowed to set registered sender");
registeredSenders[sourceChain]=sourceAddress; }

```

Example Solution for Problem 1

We provide a base class in theWormhole Solidity SDK that includes the modifier shown above, makes it easy to add these functionalities as such

```

Copy functionreceiveWormholeMessages( bytesmemorypayload, bytes[]memory,// additionalVaas bytes32sourceAddress, uint16sourceChain, bytes32deliveryHash ) public payable override onlyWormholeRelayer isRegisteredSender(sourceChain,sourceAddress) { latestGreeting=abi.decode(payload,(string));

emitGreetingReceived(latestGreeting,sourceChain,fromWormholeFormat(sourceAddress)); }

```

Included in the HelloWormhole repository is anexample contract (andforge tests) that uses these helpers:

You can add these helpers into your own project as such:

```

Copy forgeinstallwormhole-foundation/wormhole-solidity-sdk

```

Feature: Receive Refunds

Often, you cannot predict exactly how much gas your contract will use. To avoid the chance of a 'Receiver Failure' (which occurs when your contract reverts - because, for example, it runs out of gas), you should request a reasonable upper bound for how much gas your contract will use.

However, this means if, e.g. we expect HelloWormhole to take somewhere (uniformly random) between 10000 and 50000 units of gas, we are losing on expectation the cost of 20000 units of gas if we request 50000 units of gas for every request!

Fortunately, theIWormholeRelayer interfaceallows you to receive refunds for any gas you do not end up using in your target contract!

```

Copy

functionsendPayloadToEvm( uint16targetChain, addresstargetAddress, bytesmemorypayload, uint256receiverValue, uint256gasLimit, **uint16refundChain, addressrefundAddress** )externalpayablereturns(uint64sequence);

```

If these are specified, then different logic is applied depending on the values ofrefundChain andtargetChain

If refundChain is equal to targetChain , a refund of

```

Copy targetChainRefundPerGasUnused*(gasLimit-gasUsed)

```

will be sent to addressrefundAddress on the target chain.

- gasUsed
- is the amount of gas your contract (attargetAddress
-) uses in the call toreceiveWormholeMessages
- .
-

Note that this must be less than or equal to gasLimit.

- targetChainRefundPerGasUnused
- is a constant quoted pre-delivery by the delivery provider - this is the second return value of thequoteEVMDeliveryPrice
- function:
-

```

Copy functionquoteEVMDeliveryPrice( uint16targetChain, uint256receiverValue, uint256gasLimit )externalviewreturns(uint256nativePriceQuote,uint256targetChainRefundPerGasUnused);

```

else (if refundChain is not equal to targetChain), then

1. The cost to perform a delivery with a gas limit and receiver value of 0 to the refund chain will be calculated (let's call it BASE_COST)
2. if
3. TARGET_CHAIN_REFUND = targetChainRefundPerGasUnused * (gasLimit - gasUsed)
4. is larger than BASE_COST
5. , then a delivery will be performed, and themsg.value
6. that will be sent torefundAddress
7. onrefundChain
8. will be
9.

```

Copy targetChainWormholeRelayer.quoteNativeForChain(refundChain,TARGET_CHAIN_REFUND-
BASE_COST,deliveryProviderAddress)

```

Note: deliveryProviderAddress here is equal totargetChainWormholeRelayer.quoteDefaultDeliveryProvider

Included in the HelloWormhole repository is anexample contract (andforge tests) that use this refund feature.

Feature: Going from chain A → chain B → chain C

Suppose you wish to request a delivery from chain A to chain B, and then after the delivery has completed on chain B, you wish to deliver some information to chain C.

One way to do this is to callsendPayloadToEvm within the implementation ofreceiveWormholeMessages on chain B. Often in these scenarios, you only have currency on chain A, but you can still request the appropriate amount as yourreceiverValue in your delivery request on chain A.

How do you know how much receiver value to request in your delivery on chain A? Unfortunately, the amount you need depends on a quote that the delivery provider on chain B can provide. Our best recommendation here is to expose this 'receiverValue' amount as a parameter on your contract's endpoint, and have the front-end of your application determine the correct value to pass here by querying the WormholeRelayer contract on chain B.

Included in the HelloWormhole repository is anexample contract (andforge tests) that go from chain A to chain B to chain C, using the recommendation above.

Composing with other Wormhole modules - Requesting Delivery of Existing Wormhole Messages

Often times, we wish to deliver a wormhole message that has already been published (by a different contract).

To do this, use thesendVaasToEvm function, which lets you specify additional published wormhole messages for which the corresponding signed VAAs will be pass in as a parameter in the call totargetAddress

```

Copy / *@noticeVaaKey identifies a wormhole message* *@custom:memberchainId Wormhole chain ID of the chain where this VAA was emitted from *@custom:memberemitterAddress Address of the emitter of the VAA, in Wormhole bytes32 format* @custom:membersequence Sequence number of the VAA */ structVaaKey{ uint16chainId; bytes32emitterAddress; uint64sequence; }

functionsendVaasToEvm( uint16targetChain, addresstargetAddress, bytesmemorypayload, uint256receiverValue, uint256gasLimit, VaaKey[]memoryvaaKeys, uint16refundChain, addressrefundAddress )externalpayablereturns(uint64sequence);

```

For an example usage of this, see the Wormhole Solidity SDK's implementation ofsendTokenWithPayloadToEvm , which use the TokenBridge wormhole module to send tokens!

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

On this page * Protections * Issue: The greetings can come from anyone * Example Solution for Problem 1 * Feature: Receive Refunds * Feature: Going from chain A → chain B → chain C* Composing with other Wormhole modules - Requesting Delivery of Existing Wormhole Messages

Was this helpful? Edit on GitHub