Send Arbitrary Data with Acknowledgment of Receipt

This tutorial will teach you how to use Chainlink CCIP to send arbitrary data between smart contracts on different blockchains and how to track the status of each sent message in the sender contract on the source chain. Tracking the status of sent messages allows your smart contracts to execute actions after the receiver acknowledges it received the message. In this example, the sender contract emits an event after it receives acknowledgment from the receiver.

Note: For simplicity, this tutorial demonstrates this pattern for sending arbitrary data. However, you are not limited to this application. You can apply the same pattern to programmable token transfers.

Before you begin

- This tutorial assumes you have completed theSend Arbitrary Data tutorial.
- Your account must have some ETH tokens onEthereum Sepoliaand MATIC tokens onPolygon Mumbai.
- Learn how toAcquire testnet LINK andFund your contract with LINK .

Tutorial

In this tutorial, you will deploy amessage trackercontract on the source blockchain (Ethereum Sepolia) and anacknowledgeron the destination blockchain (Polygon Mumbai). Throughout the tutorial, you will pay for CCIP fees using LINK tokens. Here is a step-by-step breakdown:

1. Sending and building a CCIP message:Initiate and send a message from themessage trackercontract on Ethereum Sepolia to theacknowledgercontract on Polygon Mumbai. Themessage trackercontract constructs a CCIP message that encapsulates a text string and establishes a tracking status for this message before sending it off.
2. Receiving and acknowledging the message:After theacknowledgercontract receives the text on Polygon Mumbai, it sends back a CCIP message to themessage trackercontract as an acknowledgment of receipt.
3. Updating tracking status:After themessage trackerreceives the acknowledgment, the contract updates the tracking status of the initial CCIP message and emits an event to signal completion.

Deploy themessage tracker(sender) contract

Deploy theMessageTracker.solcontract onEthereum Sepoliaand enable it to send and receive CCIP messages to and fromPolygon Mumbai. You must also enable your contract to receive CCIP messages from theacknowledgercontract.

1. Open the MessageTracker.sol contract in Remix.

Open in Remix What is Remix?Note: The contract code is also available in theExamine the code section. 2. Compile the contract. 3. Deploy the contract onEthereum Sepolia:

1. Open MetaMask and select theEthereum Sepolianetwork.
2. On theDeploy & Run Transactionstab in Remix, selectInjected Provider - MetaMaskin theEnvironmentlist. Remix will use the MetaMask wallet to communicate withEthereum Sepolia.
3. Under theDeploysection, fill in the router address and the LINK token contract address for your specific blockchain. You can find both of these addresses on theSupported Networks page. The LINK token contract address is also listed on theLINK Token Contracts page. ForEthereum Sepolia:
4. The router address is0x0bf3de8c5d3e8a2b34d2beeb17abfcebaf363a59
5. The LINK token address is0x779877A7B0D9E8603169DdbD7836e478b4624789
6. Clicktransactto deploy the contract. MetaMask prompts you to confirm the transaction. Check the transaction details to make sure you are deploying the contract onEthereum Sepolia.
7. After you confirm the transaction, the contract address appears in theDeployed Contractslist. Copy your contract address.
8. Open MetaMask and send0.5LINK to the contract address you copied. Your contract will pay CCIP fees in LINK.
9. Allow thePolygon Mumbaichain selector for both destination and source chains.
10. On theDeploy & Run Transactionstab in Remix, expand themessage trackercontract in theDeployed Contractssection.
11. Call theallowlistDestinationChainfunction with12532609583862916517as the destination chain selector forPolygon Mumbaiandtrueas allowed.
12. Call theallowlistSourceChainfunction with12532609583862916517as the source chain selector forPolygon Mumbaiandtrueas allowed. You can find each network's chain selector on thesupported networks page .

Deploy the acknowledger (receiver) contract

Deploy theAcknowledger.solcontract onPolygon Mumbaiand enable it to send and receive CCIP messages to and fromEthereum Sepolia. You must also enable your contract to receive CCIP messages from themessage trackercontract.

1. Open the Acknowledger.sol contract in Remix.

Open in Remix What is Remix?Note: The contract code is also available in theExamine the code section. 2. Compile the contract. 3. Deploy the contract onPolygon Mumbai:

1. Open MetaMask and select thePolygon Mumbainetwork.
2. On theDeploy & Run Transactionstab in Remix, make sure theEnvironmentis still set toInjected Provider - MetaMask.
3. Under theDeploysection, fill in the router address and the LINK token contract address for your specific blockchain. You can find both of these addresses on theSupported Networks page. The LINK token contract address is also listed on theLINK Token Contracts page. ForPolygon Mumbai:
4. The Router address is0x1035cabc275068e0f4b745a29cedf38e13af41b1.
5. The LINK token address is0x326C977E6efc84E512bB9C30f76E30c160eD06FB.
6. Clicktransactto deploy the contract. MetaMask prompts you to confirm the transaction. Check the transaction details to make sure you are deploying the contract toPolygon Mumbai.
7. After you confirm the transaction, the contract address appears as the second item in theDeployed Contractslist. Copy this contract address.
8. Open MetaMask and send1LINK to the contract address that you copied. Your contract will pay CCIP fees in LINK.
9. Allow theEthereum Sepoliachain selector for both destination and source chains. You must also enable your acknowledger contract to receive CCIP messages from the message tracker you deployed onEthereum Sepolia.
10. On theDeploy & Run Transactionstab in Remix, expand theacknowledgercontract in theDeployed Contractssection. Expand theallowlistDestinationChain,allowlistSender, andallowlistSourceChainfunctions and fill in the following arguments:

FunctionDescriptionValue (Ethereum Sepolia)allowlistDestinationChainCCIP Chain identifier of the target blockchain. You can find each network's chain selector on thesupported networks page16015286601757825753,trueallowlistSenderThe address of the message tracker contract deployed onEthereum SepoliaYour deployed contract address,trueallowlistSourceChainCCIP Chain identifier of the source blockchain. You can find each network's chain selector on thesupported networks page16015286601757825753,true 2. Open MetaMask and select thePolygon Mumbainetwork. 3. For each function you expanded and filled in the arguments for, click thetransactbutton to call the function. MetaMask prompts you to confirm the transaction. Wait for each transaction to succeed before calling the following function. 5. Finally, enable yourmessage trackercontract to receive CCIP messages from theacknowledgercontract you deployed onPolygon Mumbai.

1. On theDeploy & Run Transactionstab in Remix, expand themessage trackercontract in theDeployed Contractssection. Expand theallowlistSenderfunction and fill in youracknowledgercontract address andtrueas allowed.
2. Open MetaMask and select theEthereum Sepolianetwork.
3. Clicktransactto call the function. MetaMask prompts you to confirm the transaction.

At this point, you have onemessage tracker(sender) contract onEthereum Sepoliaand oneacknowledger(receiver) contract onPolygon Mumbai. You sent0.5LINK to themessage trackercontract and1LINK to theacknowledgercontract to pay the CCIP fees.

Send data and track the message status

Initial message

1. Send aHello World!string from yourmessage trackercontract onEthereum Sepoliato youracknowledgercontract deployed onPolygon Mumbai. You will track the status of this message during this tutorial.
2. Open MetaMask and select theEthereum Sepolianetwork.
3. On theDeploy & Run Transactionstab in Remix, expand themessage trackercontract in theDeployed Contractssection.
4. Expand thesendMessagePayLINKfunction and fill in the following arguments:

ArgumentDescriptionValue (Polygon Mumbai)destinationChainSelectorCCIP Chain identifier of the target blockchain. You can find each network's chain selector on thesupported networks page12532609583862916517receiverThe destination smart contract addressYour deployedacknowledgercontract addresstextAnystringHello World! 4. Clicktransactto call the function. MetaMask

prompts you to confirm the transaction.

note

During gas price spikes, your transaction might fail, requiring more than0.5 LINKto proceed. If your transaction fails, fund your contract with moreLINKtokens and try again. 5. Upon transaction success, expand the last transaction in the Remix log and copy the transaction hash. In this example, it is0x69ada1a3d014386440f52e609f08d7c60a23727b2dafccaf10d407fb827d44d3. 2. Open theCCIP Explorer and use the transaction hash that you copied to search for your cross-chain transaction.

After the transaction is finalized on the source chain, it will take a few minutes for CCIP to deliver the data toPolygon Mumbaiand call theccipReceivefunction on youracknowledgercontract. 3. Copy the message ID from the CCIP Explorer transaction details. You will use this message ID to track your message status on themessage trackercontract. In this example, it is0x8773e27ea01eb0e7671da8d63e14a647713cb1274b7110141d249d3b8fce116b. 4. On theDeploy & Run Transactionstab in Remix, expand yourmessage trackercontract in theDeployed Contractssection. 5. Paste the message ID you copied from the CCIP explorer as the argument in themessagesInfogetter function. ClickmessagesInfoto read the message status.

Note the returnedstatus 1. This value indicates that themessage trackercontract has updated your message status to theSentstatus as defined by theMessageStatusenumin themessage trackercontract.

// Enum is used to track the status of messages sent via CCIP.//NotSent indicates a message has not yet been sent.//Sent indicates that a message has been sent to the Acknowledger contract but not yet acknowledged.// ProcessedOnDestination indicates that the Acknowledger contract has processed the message and that// the Message Tracker contract has received the acknowledgment from the Acknowledger contract.enumMessageStatus{NotSent,// 0Sent,// 1ProcessedOnDestination// 2} 6. When the transaction is marked with a "Success" status on theCCIP Explorer , the CCIP transaction and the destination transaction are complete. Theacknowledgercontract has received the message from themessage trackercontract.

Acknowledgment message

Theacknowledgercontract processes the message, sends an acknowledgment message containing the initial message ID back to themessage trackercontract, and emits anAcknowledgmentSentevent. Read thisexplanation for further description.

// Emitted when an acknowledgment message is successfully sent back to the sender contract.// This event signifies that the Acknowledger contract has recognized the receipt of an initial message// and has informed the original sender contract by sending an acknowledgment message,// including the original message ID.eventAcknowledgmentSent(bytes32indexedmessageId,// The unique ID of the CCIP message.uint64indexeddestinationChainSelector,// The chain selector of the destination chain.addressindexedreceiver,// The address of the receiver on the destination chain.bytes32data,// The data being sent back, usually containing the message ID of the original message to acknowledge its receipt.addressfeeToken,// The token address used to pay CCIP fees for sending the acknowledgment.uint256fees// The fees paid for sending the acknowledgment message via CCIP.); 1. Copy youracknowledgercontract address from Remix. Open thePolygon Mumbaiexplorer and search for your deployedacknowledgercontract. Click theEventstab to see the events log.

The first indexed topic (topic1) in theAcknowledgmentSentevent is the acknowledgment message ID sent to themessage trackercontract onEthereum Sepolia. In this example, the message ID is0x0defcd4d21415378e7136b7300940fb8c0e9e046002aa9b54a0362ecb5cee0ec. 2. Copy your own message ID from the indexedtopic1and search for it in theCCIP explorer .

When the transaction is marked with a "Success" status on the CCIP explorer, the CCIP transaction and the destination transaction are complete. Themessage trackercontract has received the message from theacknowledgercontract.

Final status check

When themessage trackerreceives the acknowledgment message, theccipReceivefunction updates the initial message status to2, which corresponds to theProcessedOnDestinationstatus as defined by theMessageStatusenum. The function emits aMessageProcessedOnDestinationevent.

1. Open MetaMask and select theEthereum Sepolianetwork.
2. On theDeploy & Run Transactionstab in Remix, expand yourmessage trackercontract in theDeployed Contractssection.
3. Copy theinitial message IDfrom the CCIP explorer (transaction fromEthereum SepoliatoPolygon Mumbai) and paste it as the argument in themessagesInfogetter function. ClickmessagesInfoto read the message status. It returnsstatus 2and the acknowledgment message ID that confirms this status.
4. Copy yourmessage trackercontract address from Remix. Open theEthereum Sepoliaexplorer and search for your deployedmessage trackercontract. Then, click on theEventstab.

TheMessageProcessedOnDestinationevent is emitted with the acknowledged message ID0x8773e27ea01eb0e7671da8d63e14a647713cb1274b7110141d249d3b8fce116bas indexedtopic2.

// Event emitted when the sender contract receives an acknowledgment// that the receiver contract has successfully received and processed the message.eventMessageProcessedOnDestination(bytes32indexedmessageId,// The unique ID of the CCIP acknowledgment message.bytes32indexedacknowledgedMsgId,// The unique ID of the message acknowledged by the receiver.uint64indexedsourceChainSelector,// The chain selector of the source chain.addresssender// The address of the sender from the source chain.);

Explanation

The smart contracts featured in this tutorial are designed to interact with CCIP to send and receive messages with an acknowledgment of receipt mechanism. The contract code across both contracts contains supporting comments clarifying the functions, events, and underlying logic.

Refer to theSend Arbitrary Data tutorial for more explanation aboutinitializing the contracts ,sending data, paying in LINK , andreceiving data .

Here, we will further explain the acknowledgment of receipt mechanism.

Message acknowledgment of receipt mechanism

This mechanism ensures that a message sent by themessage tracker(sender) contract is received and acknowledged by theacknowledger(receiver) contract. The message status is tracked and stored in themessage trackercontract.

// Enum is used to track the status of messages sent via CCIP.//NotSent indicates a message has not yet been sent.//Sent indicates that a message has been sent to the Acknowledger contract but not yet acknowledged.// ProcessedOnDestination indicates that the Acknowledger contract has processed the message and that// the Message Tracker contract has received the acknowledgment from the Acknowledger contract.enumMessageStatus{NotSent,// 0Sent,// 1ProcessedOnDestination// 2}// Struct to store the status and acknowledger message ID of a message.structMessageInfo{MessageStatus status;bytes32acknowledgerMessageId;}// Mapping to keep track of message IDs to their info (status & acknowledger message ID).mapping(bytes32=>MessageInfo)publicmessagesInfo;

Message trackercontract

Themessage trackercontract acts as the sender, initiating cross-chain communication. It performs the following operations:

- Message sending: Constructs and sends messages to theacknowledgercontract on another blockchain, usingsendMessagePayLINK function. On top of itsfive primary operations , thesendMessagePayLINK function also updates the message status upon sending.
- Status tracking:
- Upon sending a message, themessage trackerupdates its internal state to mark the message asSent(status1). This status is pivotal for tracking the message lifecycle and awaiting acknowledgment.

// Update the message status to SentmessagesInfo[messageId].status=MessageStatus.Sent; * Upon receiving an acknowledgment message from theacknowledgercontract, themessage trackercontract updates the message status fromSent(status1) toProcessedOnDestination(status2). This update indicates that the cross-chain communication cycle is complete, and the receiver successfully received and acknowledged the message.

// Update the message status to ProcessedOnDestinationmessagesInfo[messageId].status=MessageStatus.ProcessedOnDestination;

Acknowledgercontract

Theacknowledgercontract receives the message, sends back an acknowledgment message, and emits an event. It performs the following operations:

- Message receipt: Upon receiving a message via CCIP, theccipReceivefunction decodes it and calls theacknowledgePayLINKfunction nested within theccipReceive function.
- Acknowledgment sending: TheacknowledgePayLINKfunction acts as a customsendMessagePayLINK function nested within theccipReceivefunction. It sends an acknowledgment (a CCIP message) to themessage trackercontract upon the initial message receipt. The data transferred in this acknowledgment message is the initial message ID. It then emits anAcknowledgmentSentevent.

Security and integrity

Both contracts use allowlists to process only messages from and to allowed sources.

- Allowlisting chains and senders:
- ThesendMessagePayLINKfunction is protected by theonlyAllowlistedDestinationChainmodifier, ensuring the contract owner has allowlisted a destination chain.
- TheccipReceivefunction is protected by theonlyAllowlistedmodifier, ensuring the contract owner has allowlisted a source chain and a sender.

- Ensuring the initial message authenticity: Themessage trackercontract first checks that the message awaiting acknowledgment was sent from the contract itself and is currently marked asSent. Once confirmed, the message status is updated toProcessedOnDestination.

Examine the code

MessageTracker.sol

// SPDX-License-Identifier: MITpragmasolidity0.8.19;import{IRouterClient}from"@chainlink/contracts-ccip/src/v0.8/ccip/interfaces/IRouterClient.sol";import{OwnerIsCreator}from"@chainlink/contracts-ccip/src/v0.8/shared/access/OwnerIsCreator.sol";import{Client}from"@chainlink/contracts-ccip/src/v0.8/ccip/libraries/Client.sol";import{CCIPReceiver}from"@chainlink/contracts-ccip/src/v0.8/ccip/applications/CCIPReceiver.sol";import{IERC20}from"@chainlink/contracts-ccip/src/v0.8/vendor/openzeppelin-solidity/v4.8.3/contracts/token/ERC20/IERC20.sol";import{SafeERC20}from"@chainlink/contracts-ccip/src/v0.8/vendor/openzeppelin-solidity/v4.8.3/contracts/token/ERC20/utils/SafeERC20.sol";usingSafeERC20forIERC20;/ * **THIS IS AN EXAMPLE CONTRACT THAT USES HARDCODED VALUES FOR CLARITY. * THIS IS AN EXAMPLE CONTRACT THAT USES UN-AUDITED CODE. * DO NOT USE THIS CODE IN PRODUCTION.** */// **@title - A simple messenger contract for sending/receiving data across chains and tracking the status of sent messages.contractMessageTrackerisCCIPReceiver,OwnerIsCreator{// Custom errors to provide more descriptive revert messages.errorNotEnoughBalance(uint256currentBalance,uint256calculatedFees);// Used to make sure contract has enough balance.errorNothingToWithdraw();// Used when trying to withdraw Ether but there's nothing to withdraw.errorDestinationChainNotAllowlisted(uint64destinationChainSelector);// Used when the destination chain has not been allowlisted by the contract owner.errorSourceChainNotAllowlisted(uint64sourceChainSelector);// Used when the source chain has not been allowlisted by the contract owner.errorSenderNotAllowlisted(addresssender);// Used when the sender has not been allowlisted by the contract owner.errorInvalidReceiverAddress();// Used when the receiver address is 0.errorMessageWasNotSentByMessageTracker(bytes32msgId);// Triggered when attempting to confirm a message not recognized as sent by this tracker.errorMessageHasAlreadyBeenProcessedOnDestination(bytes32msgId);// Triggered when trying to mark a message as** `ProcessedOnDestination` **when it is already marked as such.// Enum is used to track the status of messages sent via CCIP.//** `NotSent` **indicates a message has not yet been sent.//**`Sent` **indicates that a message has been sent to the Acknowledger contract but not yet acknowledged.//** `ProcessedOnDestination` **indicates that the Acknowledger contract has processed the message and that// the Message Tracker contract has received the acknowledgment from the Acknowledger contract.enumMessageStatus{NotSent,// 0Sent,// 1ProcessedOnDestination// 2}// Struct to store the status and acknowledger message ID of a message.structMessageInfo{MessageStatus status;bytes32acknowledgerMessageId;}// Mapping to keep track of allowlisted destination chains.mapping(uint64=>bool)publicallowlistedDestinationChains;// Mapping to keep track of allowlisted source chains.mapping(uint64=>bool)publicallowlistedSourceChains;// Mapping to keep track of allowlisted senders.mapping(address=>bool)publicallowlistedSenders;// Mapping to keep track of message IDs to their info (status & acknowledger message ID).mapping(bytes32=>MessageInfo)publicmessagesInfo;// Event emitted when a message is sent to another chain.eventMessageSent(bytes32indexedmessageId,// The unique ID of the CCIP message.uint64indexeddestinationChainSelector,// The chain selector of the destination chain.addressreceiver,// The address of the receiver on the destination chain.stringtext,// The text being sent.addressfeeToken,// the token address used to pay CCIP fees.uint256fees// The fees paid for sending the CCIP message.);// Event emitted when the sender contract receives an acknowledgment// that the receiver contract has successfully received and processed the message.eventMessageProcessedOnDestination(bytes32indexedmessageId,// The unique ID of the CCIP acknowledgment message.bytes32indexedacknowledgedMsgId,// The unique ID of the message acknowledged by the receiver.uint64indexedsourceChainSelector,// The chain selector of the source chain.addresssender// The address of the sender from the source chain.);IERC20privates_linkToken;/// @notice Constructor initializes the contract with the router address./// @param** ***router The address of the router contract./// @param _link The address of the link contract.constructor(address_router,address_link)CCIPReceiver(_router){s_linkToken=IERC20(_link);}/// @dev Modifier that checks if the chain with the given destinationChainSelector is allowlisted./// @param _destinationChainSelector The selector of the destination chain.modifieronlyAllowlistedDestinationChain(uint64_destinationChainSelector){if(!allowlistedDestinationChains[_destinationChainSelector])revertDestinationChainNotAllowlisted(_destinationChainSelector);;}*****/// @dev Modifier that checks if the chain with the given sourceChainSelector is allowlisted and if the sender is allowlisted./// @param** ***sourceChainSelector The selector of the destination chain./// @param _sender The address of the sender.modifieronlyAllowlisted(uint64_sourceChainSelector,address_sender){if(!allowlistedSourceChains[_sourceChainSelector])revertSourceChainNotAllowlisted(_sourceChainSelector);if(!allowlistedSenders[_sender])revertSenderNotAllowlisted(_sender);;}*****/// @dev Modifier that checks the receiver address is not 0./// @param** ***receiver The receiver address.modifiervalidateReceiver(address_receiver){if(_receiver==address(0))revertInvalidReceiverAddress();;}*****/// @dev Updates the allowlist status of a destination chain for transactions.functionallowlistDestinationChain(uint64_destinationChainSelector,boolallowed)externalonlyOwner{allowlistedDestinationChains[_destinationChainSelector]=allowed;}/// @dev Updates the allowlist status of a source chain for transactions.functionallowlistSourceChain(uint64_sourceChainSelector,boolallowed)externalonlyOwner{allowlistedSourceChains[_sourceChainSelector]=allowed;}/// @dev Updates the allowlist status of a sender for transactions.functionallowlistSender(address_sender,boolallowed)externalonlyOwner{allowlistedSenders[_sender]=allowed;}/// @notice Sends data to receiver on the destination chain./// @notice Pay for fees in LINK./// @dev Assumes your contract has sufficient LINK./// @param _destinationChainSelector The identifier (aka selector) for the destination blockchain./// @param _receiver The address of the recipient on the destination blockchain./// @param _text The text to be sent./// @return messageId The ID of the CCIP message that was sent.functionsendMessagePayLINK(uint64_destinationChainSelector,address_receiver,stringcalldata_text)externalonlyOwneronlyAllowlistedDestinationChain(_destinationChainSelector)v{// Create an EVM2AnyMessage struct in memory with necessary information for sending a cross-chain messageClient.EVM2AnyMessagememoryevm2AnyMessage=_buildCCIPMessage(_receiver,_text,address(s_linkToken));// Initialize a router client instance to interact with cross-chain routerIRouterClient router=IRouterClient(this.getRouter());// Get the fee required to send the CCIP messageuint256fees=router.getFee(_destinationChainSelector,evm2AnyMessage);if(fees>s_linkToken.balanceOf(address(this)))revertNotEnoughBalance(s_linkToken.balanceOf(address(tl// approve the Router to transfer LINK tokens on contract's behalf. It will spend the fees in LINKs_linkToken.approve(address(router),fees);// Send the CCIP message through the router and store the returned CCIP message IDmessageId=router.ccipSend(_destinationChainSelector,evm2AnyMessage);// Update the message status to** `Sent` **messagesInfo[messageId].status=MessageStatus.Sent;// Emit an event with message detailsemitMessageSent(messageId,_destinationChainSelector,_receiver,_text,address(s_linkToken),fees);// Return the CCIP message IDreturnmessageId;}/** @dev Receives and processes messages sent via the Chainlink CCIP from allowed chains and senders. * Upon receiving a message, this function checks if the message's associated data indicates a previously * sent message awaiting acknowledgment. If the message is valid (i.e., its status is* `Sent`*), it updates * the message's status to* `ProcessedOnDestination`*, thereby acknowledging its receipt. It then emits a* `MessageProcessedOnDestination` *event. If the message cannot be validated (e.g., it was not sent or has been tampered with), the function * reverts with a* `MessageWasNotSentByMessageTracker` *error. This mechanism ensures that only messages * genuinely sent and awaiting acknowledgment are marked as* `ProcessedOnDestination`*. * @param any2EvmMessage The CCIP message received, which includes the message ID, the data being acknowledged, * the source chain selector, and the sender's address. */function_ccipReceive(Client.Any2EVMMessagememoryany2EvmMessage)internaloverrideonlyAllowlisted(any2EvmMessage.sourceChainSelector,abi.decode(any2EvmMessage.sender,(address)))// Ensure the source chain and sender are allowlisted for added security{bytes32initialMsgId=abi.decode(any2EvmMessage.data,(bytes32));// Decode the data sent by the receiverbytes32acknowledgerMsgId=any2EvmMessage.messageId;messagesInfo[initialMsgId].acknowledgerMessageId=acknowledgerMsgId;// Store the messageId of the received messageif(messagesInfo[initialMsgId].status==MessageStatus.Sent){// Updates the status of the message to 'ProcessedOnDestination' to reflect that an acknowledgment// of receipt has been received and emits an event to log this confirmation along with relevant details.messagesInfo[initialMsgId].status=MessageStatus.ProcessedOnDestination;emitMessageProcessedOnDestination(acknowledgerMsgId,initialMsgId,any2EvmMessage.sourceChainSelector,abi.de(address)));}elseif(messagesInfo[initialMsgId].status==MessageStatus.ProcessedOnDestination){// If the message is already marked as 'ProcessedOnDestination', this indicates an attempt to// re-confirm a message that has already been processed on the destination chain and marked as such.revertMessageHasAlreadyBeenProcessedOnDestination(initialMsgId);}else{// If the message status is neither 'Sent' nor 'ProcessedOnDestination', it implies that the// message ID provided for acknowledgment does not correspond to a valid, previously// sent message.revertMessageWasNotSentByMessageTracker(initialMsgId);}}/// @notice Construct a CCIP message./// @dev This function will create an EVM2AnyMessage struct with all the necessary information for sending a text./// @param _receiver The address of the receiver./// @param _text The string data to be sent./// @param _feeTokenAddress The address of the token used for fees. Set address(0) for native gas./// @return Client.EVM2AnyMessage Returns an EVM2AnyMessage struct which contains information for sending a CCIP message.function_buildCCIPMessage(address_receiver,stringcalldata_text,address_feeTokenAddress)privatepurereturns(Client.EVM2AnyMessagememory){// Create an EVM2AnyMessage struct in memory with necessary information for sending a cross-chain messagereturnClient.EVM2AnyMessage({receiver:abi.encode(_receiver),// ABI-encoded receiver addressdata:abi.encode(_text),// ABI-encoded stringtokenAmounts:newClient.EVMTokenAmount,// Empty array as no tokens are transferredextraArgs:Client._argsToBytes(// Additional arguments, setting gas limitClient.EVMExtraArgsV1({gasLimit:300_000})),// Set the feeToken to a feeTokenAddress, indicating specific asset will be used for feesfeeToken:_feeTokenAddress});}/// @notice Allows the owner of the contract to withdraw all tokens of a specific ERC20 token./// @dev This function reverts with a 'NothingToWithdraw' error if there are no tokens to withdraw./// @param _beneficiary The address to which the tokens will be sent./// @param _token The contract address of the ERC20 token to be withdrawn.functionwithdrawToken(address_beneficiary,address_token)publiconlyOwner{// Retrieve the balance of this contractuint256amount=IERC20(_token).balanceOf(address(this));// Revert if there is nothing to withdrawif(amount==0)revertNothingToWithdraw();IERC20(_token).safeTransfer(_beneficiary,amount);}} Open in Remix What is Remix?

Acknowledger.sol

// SPDX-License-Identifier: MITpragmasolidity0.8.19;import{IRouterClient}from"@chainlink/contracts-ccip/src/v0.8/ccip/interfaces/IRouterClient.sol";import{OwnerIsCreator}from"@chainlink/contracts-ccip/src/v0.8/shared/access/OwnerIsCreator.sol";import{Client}from"@chainlink/contracts-ccip/src/v0.8/ccip/libraries/Client.sol";import{CCIPReceiver}from"@chainlink/contracts-ccip/src/v0.8/ccip/applications/CCIPReceiver.sol";import{IERC20}from"@chainlink/contracts-ccip/src/v0.8/vendor/openzeppelin-solidity/v4.8.3/contracts/token/ERC20/IERC20.sol";import{SafeERC20}from"@chainlink/contracts-ccip/src/v0.8/vendor/openzeppelin-solidity/v4.8.3/contracts/token/ERC20/utils/SafeERC20.sol";usingSafeERC20forIERC20;/* *THIS IS AN EXAMPLE CONTRACT THAT USES HARDCODED VALUES FOR CLARITY. * THIS IS AN EXAMPLE CONTRACT THAT USES UN-AUDITED CODE. * DO NOT USE THIS CODE IN PRODUCTION.* //// @title - A simple acknowledger contract for receiving data and sending acknowledgement of receipt messages across chains.contractAcknowledgerisCCIPReceiver,OwnerIsCreator{// Custom errors to provide more descriptive revert messages.errorNotEnoughBalance(uint256currentBalance,uint256calculatedFees);// Used to make sure contract has enough balance.errorNothingToWithdraw();// Used when trying to withdraw Ether but there's nothing to withdraw.errorDestinationChainNotAllowlisted(uint64destinationChainSelector);// Used when the destination chain has not been allowlisted by the contract owner.errorInvalidReceiverAddress();// Used when the receiver address is 0.errorSourceChainNotAllowlisted(uint64sourceChainSelector);// Used when the source chain has not been allowlisted by the contract owner.errorSenderNotAllowlisted(addresssender);// Used when the sender has not been allowlisted by the contract owner.stringprivates_lastReceivedText;// Store the last received text.// Mapping to keep track of allowlisted destination chains.mapping(uint64=>bool)publicallowlistedDestinationChains;// Mapping to keep track of allowlisted source chains.mapping(uint64=>bool)publicallowlistedSourceChains;// Mapping to keep track of allowlisted senders.mapping(address=>bool)publicallowlistedSenders;// Emitted when an acknowledgment message is successfully sent back to the sender contract.// This event signifies that the Acknowledger contract has recognized the receipt of an initial message// and has informed the original sender contract by sending an acknowledgment message,// including the original message ID.eventAcknowledgmentSent(bytes32indexedmessageId,// The unique ID of the CCIP message.uint64indexeddestinationChainSelector,// The chain selector of the destination chain.addressindexedreceiver,// The address of the receiver on the destination chain.bytes32data,// The data being sent back, containing the message ID of the initial message to acknowledge.addressfeeToken,// The token address used to pay CCIP fees for sending the acknowledgment.uint256fees// The fees paid for sending the acknowledgment message via CCIP.);IERC20privates_linkToken;/// @notice Constructor initializes the contract with the router address./// @param *router The address of the router contract./// @param _link The address of the link contract.constructor(address_router,address_link)CCIPReceiver(_router){s_linkToken=IERC20(_link);}/// @dev Modifier that checks if the chain with the given destinationChainSelector is allowlisted./// @param _destinationChainSelector The selector of the destination chain.modifieronlyAllowlistedDestinationChain(uint64_destinationChainSelector)*

{if(!allowlistedDestinationChains[_destinationChainSelector])revertDestinationChainNotAllowlisted(_destinationChainSelector);;}/// @dev Modifier that checks if the chain with the given sourceChainSelector is allowlisted and if the sender is allowlisted./// @param *sourceChainSelector The selector of the destination chain./// @param _sender The address of the sender.modifieronlyAllowlisted(uint64_sourceChainSelector,address_sender)*
{if(!allowlistedSourceChains[_sourceChainSelector])revertSourceChainNotAllowlisted(_sourceChainSelector);if(!allowlistedSenders[_sender])revertSenderNotAllowlisted(_sender);;}/// @dev Updates the allowlist status of a destination chain for transactions.functionallowlistDestinationChain(uint64_destinationChainSelector,boolallowed)externalonlyOwner{allowlistedDestinationChains[_destinationChainSelector]=allowed;}/// @dev Updates the allowlist status of a source chain for transactions.functionallowlistSourceChain(uint64_sourceChainSelector,boolallowed)externalonlyOwner{allowlistedSourceChains[_sourceChainSelector]=allowed;}/// @dev Updates the allowlist status of a sender for transactions.functionallowlistSender(address_sender,boolallowed)externalonlyOwner{allowlistedSenders[_sender]=allowed;}/// @notice Sends an acknowledgment message back to the sender contract on the source chain/// and pays the fees using LINK tokens./// @dev This function constructs and sends an acknowledgment message using CCIP,/// indicating the receipt and processing of an initial message. It emits the AcknowledgmentSent event/// upon successful sending. This function should be called after processing the received message/// to inform the sender contract about the successful message reception./// @param _messageIdToAcknowledge The message ID of the initial message being acknowledged./// @param _messageTrackerAddress The address of the message tracker contract on the source chain./// @param _messageTrackerChainSelector The chain selector of the source chain.function_acknowledgePayLINK(bytes32_messageIdToAcknowledge,address_messageTrackerAddress,uint64_messageTrackerChainSelector)private{if(_messageTrackerAddress==address(0))rev
Construct the CCIP message for acknowledgment, including the message ID of the initial message.Client.EVM2AnyMessagememoryacknowledgment=Client.EVM2AnyMessage({receiver:abi.encode(_messageTrackerAddress),// ABI-encoded receiver addressdata:abi.encode(_messageIdToAcknowledge),// ABI-encoded message ID to acknowledgetokenAmounts:newClient.EVMTokenAmount,// Empty array aas no tokens are transferredextraArgs:Client._argsToBytes(// Additional arguments, setting gas limitClient.EVMExtraArgsV1({gasLimit:200_000})),// Set the feeToken to a feeTokenAddress, indicating specific asset will be used for feesfeeToken:address(s_linkToken)});// Initialize a router client instance to interact with the cross-chain router.IRouterClient router=IRouterClient(this.getRouter());// Calculate the fee required to send the CCIP acknowledgment message.uint256fees=router.getFee(_messageTrackerChainSelector,// The chain selector for routing the message.acknowledgment// The acknowledgment message data.);// Ensure the contract has sufficient balance to cover the message sending fees.if(fees>s_linkToken.balanceOf(address(this)))
{revertNotEnoughBalance(s_linkToken.balanceOf(address(this)),fees);}// Approve the router to transfer LINK tokens on behalf of this contract to cover the sending fees.s_linkToken.approve(address(router),fees);// Send the acknowledgment message via the CCIP router and capture the resulting message ID.bytes32messageId=router.ccipSend(_messageTrackerChainSelector,// The destination chain selector.acknowledgment// The CCIP message payload for acknowledgment.);// Emit an event detailing the acknowledgment message sending, for external tracking and verification.emitAcknowledgmentSent(messageId,// The ID of the sent acknowledgment message._messageTrackerChainSelector,// The destination chain selector._messageTrackerAddress,// The receiver of the acknowledgment, typically the original sender._messageIdToAcknowledge,// The original message ID that was acknowledged.address(s_linkToken),// The fee token used.fees// The fees paid for sending the message.);}/// @dev Handles a received CCIP message, processes it, and acknowledges its receipt.///
This internal function is called upon the receipt of a new message via CCIP from an allowlisted source chain and sender./// It decodes the message and acknowledges its receipt by calling _acknowledgePayLINK./// @param any2EvmMessage The CCIP message receivedfunction_ccipReceive(Client.Any2EVMMessagememoryany2EvmMessage)internaloverrideonlyAllowlisted(any2EvmMessage.sourceChainSelector,abi.decode(any2EvmMessage.sender,(address)))// Make sure source chain and sender are allowlisted{bytes32messageIdToAcknowledge=any2EvmMessage.messageId;// The message ID of the received message to acknowledgeaddressmessageTrackerAddress=abi.decode(any2EvmMessage.sender,(address));// ABI-decoding of the message tracker addressuint64messageTrackerChainSelector=any2EvmMessage.sourceChainSelector;// The chain selector of the received messages_lastReceivedText=abi.decode(any2EvmMessage.data,(string));// abi-decoding of the sent text_acknowledgePayLINK(messageIdToAcknowledge,messageTrackerAddress,messageTrackerChainSelector);}/// @notice Fetches the details of the last received message./// @return text The last received text.functiongetLastReceivedMessage()externalviewreturns(stringmemorytext){return(s_lastReceivedText);}/// @notice Allows the owner of the contract to withdraw all tokens of a specific ERC20 token./// @dev This function reverts with a 'NothingToWithdraw' error if there are no tokens to withdraw./// @param _beneficiary The address to which the tokens will be sent./// @param _token The contract address of the ERC20 token to be withdrawn.functionwithdrawToken(address_beneficiary,address_token)publiconlyOwner{// Retrieve the balance of this contractuint256amount=IERC20(_token).balanceOf(address(this));// Revert if there is nothing to withdrawif(amount==0)revertNothingToWithdraw();IERC20(_token).safeTransfer(_beneficiary,amount);}} Open in Remix What is Remix?

Final note

In this example, themessage trackercontract emits an event when it receives the acknowledgment message confirming the initial message reception and processing on the counterpart chain. However, you could think of any other logic to execute when themessage trackerreceives the acknowledgment. This tutorial demonstrates the pattern for sending arbitrary data, but you can apply the same pattern to programmable token transfers.