Access Data Streams Using Automation

Early Access

Data Streams is available on Arbitrum Mainnet and Arbitrum Sepolia in Early AccessContact us to talk to an expert about integrating Chainlink Data Streams with your applications.

This guide shows you how to read data from a Data Streams feed, validate the answer, and store the answer onchain. This guide uses theRemix IDE so you can complete these steps in a web-based development environment. If you prefer to complete these steps using terminal commands, read theGetting Started - Hardhat CLI guide instead.

This example uses aChainlink Automation Log Trigger to check for events that require data. For this example, the log trigger comes from a simple emitter contract. Chainlink Automation then usesStreamsLookupto retrieve a signed report from the Data Streams Engine, return the data in a callback, and run theperformUpkeepfunction on your registered upkeep contract. TheperformUpkeepfunction calls theverifyfunction on the verifier contract.

Disclaimer

This guide represents an example of using a Chainlink product or service and is provided to help you understand how to interact with Chainlink's systems and services so that you can integrate them into your own. This template is provided "AS IS" and "AS AVAILABLE" without warranties of any kind, has not been audited, and may be missing key checks or error handling to make the usage of the product more clear. Do not use the code in this example in a production environment without completing your own audits and application of best practices. Neither Chainlink Labs, the Chainlink Foundation, nor Chainlink node operators are responsible for unintended outputs that are generated due to errors in code.

Before you begin

- If you are new to smart contract development, learn how toDeploy Your First Smart Contract so you are familiar with the tools that are necessary for this guide:* TheSolidity programming language
- TheMetaMask wallet
- TheRemix development environment
- Acquire testnet funds. This guide requires testnet ETH and LINK onArbitrum Sepolia.* Use theArbitrum Bridge to transfer testnet ETH from Ethereum Sepolia to Arbitrum Sepolia. Testnet ETH on Ethereum Sepolia is available at one ofseveral faucets .
- Testnet LINK is available for Arbitrum Sepolia atfaucets.chain.link .
- Learn how toFund your contract with LINK .

Tutorial

Deploy the Chainlink Automation upkeep contract

Deploy an upkeep contract that is enabled to retrieve data from Data Streams. For this example, you will read from the ETH/USD stream with ID0x00027bbaff688c906a3e20a34fe951715d1018d262a5b66e38eda027a674cd1bon Arbitrum Sepolia. See theStream Identifiers page for a complete list of available assets, IDs, and verifier proxy addresses.

1. Open the StreamsUpkeep.sol contract in Remix.

Open in Remix What is Remix? 2. Select theStreamsUpkeep.solcontract in theSolidity Compilertab. 3. Compile the contract. You can ignore the warning messages for this example. 4. Open MetaMask and set the network toArbitrum Sepolia. If you need to add Arbitrum Sepolia to your wallet, you can find the chain ID and the LINK token contract address on theLINK Token Contracts page.

- Arbitrum Sepolia testnet and LINK token contract
- On theDeploy & Run Transactionstab in Remix, selectInjected Provider - MetaMaskin theEnvironmentlist. Remix will use the MetaMask wallet to communicate withArbitrum Sepolia.
- In theContractsection, select theStreamsUpkeepcontract and fill in theverifier proxy addresscorresponding to the stream you want to read from. You can find this address on theStream IDs page. The verifier proxy address for the ETH/USD stream on Arbitrum Sepolia is0x2ff010DEbC1297f19579B4246cad07bd24F2488A.
- Click theDeploybutton to deploy the contract. MetaMask prompts you to confirm the transaction. Check the transaction details to ensure you deploy the contract toArbitrum Sepolia.
- After you confirm the transaction, the contract address appears under theDeployed Contractslist in Remix. Save this contract address for later.

Deploy the emitter contract

This contract emits logs that trigger the upkeep. This code can be part of your dApp. For example, you might emit log triggers when your users initiate a trade or other action requiring data retrieval. For this Getting Started guide, use a very simple emitter so you can test the upkeep and data retrieval.

1. Open the LogEmitter.sol contract in Remix.

Open in Remix What is Remix? 2. Under theSolidity Compilertab, select the0.8.19Solidity compiler and click theCompile LogEmitter.solbutton to compile the contract. 3. Open MetaMask and make sure the network is still set toArbitrum Sepolia. 4. On theDeploy & Run Transactionstab in Remix, ensure theEnvironmentis still set toInjected Provider - MetaMask. 5. Click theDeploybutton to deploy the contract. MetaMask prompts you to confirm the transaction. Check the transaction details to ensure you deploy the contract toArbitrum Sepolia. 6. After you confirm the transaction, the contract address appears in theDeployed Contractslist. Save this contract address for later.

Register the upkeep

Register a newLog triggerupkeep. SeeAutomation Log Triggers to learn more about how to register Log Trigger upkeeps.

1. Go to theChainlink Automation UI forArbitrum Sepoliaand connect your browser wallet.
2. ClickRegister new Upkeep.
3. Select theLog triggerupkeep type and clickNext.
4. Specify the upkeep contract address you saved earlier as theContract to automate. In this example, you can ignore the warning about the Automation compatible contract verification. ClickNext.
5. Specify the emitter contract address that you saved earlier. This tells Chainlink Automation what contracts to watch for log triggers. Then clickNext.
6. Provide the ABI if the contract is not validated. To find the ABI of your contract in Remix, navigate to theSolidity Compilertab. Then, copy the ABI to your clipboard using the button at the bottom of the panel.
7. Select theLogevent as the triggering event in theEmitted logdropdown.Log index topic filtersare optional filters to narrow the logs you want to trigger your upkeep. For this example, leave the field blank. ClickNext.
8. Specify a name for the upkeep.
9. Specify aStarting balanceof 1 testnet LINK for this example. You can retrieve unused LINK later.
10. Leave theCheck datavalue and other fields blank for now, and clickRegister Upkeep. MetaMask prompts you to confirm the transaction. Wait for the transaction to complete.

Fund the upkeep contract

In this example, the upkeep contract pays for onchain verification of reports from Data Streams. The Automation subscription does not cover the cost.

Open MetaMask and send 1 testnet LINK onArbitrum Sepoliato the upkeep contract address you saved earlier.

Emit a log

Now, you can use your emitter contract to emit a log and initiate the upkeep, which retrieves data for the specified Data Streams asset ID.

1. In Remix, on theDeploy & Run Transactionstab, expand your emitter contract under theDeployed Contractssection.
2. Click theemitLogbutton to call the function and emit a log. MetaMask prompts you to accept the transaction.

After the transaction is complete, the log is emitted, and the upkeep is triggered. You can find the upkeep transaction hash in theChainlink Automation UI . Check to make sure the transaction is successful.

View the retrieved price

The retrieved price is stored as a variable in the contract and emitted in the logs.

1. On theDeploy & Run Transactionstab in Remix, expand the details of your upkeep contract in theDeployed Contractssection.
2. Click thelast_retrieved_pricegetter function to view the retrieved price. The answer on the ETH/USD stream uses 18 decimal places, so an answer of2484121000000000000indicates an ETH/USD price of 2484.121. Each stream uses a different number of decimal places for answers. See theStream IDs page for more information.

Alternatively, you can view the price emitted in the logs for your upkeep transaction. You can find the upkeep transaction hash atChainlink Automation UI and view the transaction logs in theArbitrum Sepolia explorer .

Examine the code

The example code you deployed has all the interfaces and functions required to work with Chainlink Automation as an upkeep contract. It follows a similar flow to the trading flow in the Architecture documentation but uses a basic log emitter to simulate the client contract that would initiate aStreamsLookup. The code example usesrevertwithStreamsLookupto convey call information about what streams to retrieve. See theEIP-3668 rationale for more information about how to userevertin this way.

```
// SPDX-License-Identifier: MITpragmasolidity0.8.19;import{Common}from"@chainlink/contracts/src/v0.8/llo-
feeds/libraries/Common.sol";import{StreamsLookupCompatibleInterface}from"@chainlink/contracts/src/v0.8/automation/interfaces/StreamsLookupCompatibleInterface.sol";import{ILogAutomation,Log}fro
feeds/interfaces/IRewardManager.sol";import{IVerifierFeeManager}from"@chainlink/contracts/src/v0.8/llo-
feeds/interfaces/IVerifierFeeManager.sol";import{IERC20}from"@chainlink/contracts/src/v0.8/vendor/openzeppelin-solidity/v4.8.3/contracts/interfaces/IERC20.sol";/ * THIS IS AN EXAMPLE
CONTRACT THAT USES UN-AUDITED CODE FOR DEMONSTRATION PURPOSES. * DO NOT USE THIS CODE IN PRODUCTION. *///  Custom interfaces for IVerifierProxy and
IFeeManagerinterfaceIVerifierProxy{functionverify(bytescalldatapayload,bytescalldataparameterPayload)externalpayablereturns(bytesmemoryverifierResponse);functions_feeManager()ext
The feed ID the report has data foruint32validFromTimestamp;// Earliest timestamp for which price is applicableuint32observationsTimestamp;// Latest timestamp for which price is
applicableuint192nativeFee;// Base cost to validate a transaction using the report, denominated in the chain’s native token (WETH/ETH)uint192linkFee;// Base cost to validate a
transaction using the report, denominated in LINKuint32expiresAt;// Latest timestamp where the report can be verified onchainint192price;// DON consensus median price, carried to 8
decimal places}structPremiumReport{bytes32feedId;// The feed ID the report has data foruint32validFromTimestamp;// Earliest timestamp for which price is
applicableuint32observationsTimestamp;// Latest timestamp for which price is applicableuint192nativeFee;// Base cost to validate a transaction using the report, denominated in the
chain’s native token (WETH/ETH)uint192linkFee;// Base cost to validate a transaction using the report, denominated in LINKuint32expiresAt;// Latest timestamp where the report can be
verified onchainint192price;// DON consensus median price, carried to 8 decimal placesint192bid;// Simulated price impact of a buy order up to the X% depth of liquidity
utilisationint192ask;// Simulated price impact of a sell order up to the X% depth of liquidity
utilisation}structQuote{addressquoteAddress;}eventPriceUpdate(int192indexedprice);IVerifierProxypublicverifier;addresspublicFEE_ADDRESS;stringpublicconstantDATASTREAMS_FEED
This example reads the ID for the basic ETH/USD price report on Arbitrum Sepolia.// Find a complete list of IDs at https://docs.chain.link/data-streams/stream-idsstring[]publicfeedIds=
["0x00027bbaff688c906a3e20a34fe951715d1018d262a5b66e38eda027a674cd1b"];constructor(address_verifier){verifier=IVerifierProxy(_verifier);}// This function uses revert to convey call
information.// See https://eips.ethereum.org/EIPS/eip-3668#rationale for
details.functioncheckLog(Logcalldatalog,bytesmemory)externalreturns(boolupkeepNeeded,bytesmemoryperformData)
{revertStreamsLookup(DATASTREAMS_FEEDLABEL,feedIds,DATASTREAMS_QUERYLABEL,log.timestamp,"");}/ * @notice this is a new, optional function in streams lookup. It is meant to
surface streams lookup errors. * @return upkeepNeeded boolean to indicate whether the keeper should call performUpkeep or not. * @return performData bytes that the keeper should call
performUpkeep with, if * upkeep is needed. If you would like to encode data to decode later, try abi.encode.
/functioncheckErrorHandler(uint256/errCode/,bytesmemory/extraData/)externalpurereturns(boolupkeepNeeded,bytesmemoryperformData){return(true,"0");// Hardcoded to always perform upkeep.//
Read the StreamsLookup error handler guide for more information.// https://docs.chain.link/chainlink-automation/guides/streams-lookup-error-handler}// The Data Streams report bytes is passed here.//
extraData is context data from feed lookup process.// Your contract may include logic to further process this data.// This method is intended only to be simulated offchain by Automation.// The data
returned will then be passed by Automation into performUpkeepfunctioncheckCallback(bytes[]calldatavalues,bytescalldataextraData)externalpurereturns(bool,bytesmemory)
{return(true,abi.encode(values,extraData));}// function will be performed onchainfunctionperformUpkeep(bytescalldataperformData)external{// Decode the performData bytes passed in by CL
Automation.// This contains the data returned by your implementation in checkCallback().(bytes[]memorysignedReports,bytesmemoryextraData)=abi.decode(performData,
(bytes[],bytes));bytesmemoryunverifiedReport=signedReports[0];(,/ bytes32[3] reportContextData */bytesmemoryreportData)=abi.decode(unverifiedReport,(bytes32[3],bytes));// Report verification
feesIFeeManager feeManager=IFeeManager(address(verifier.s_feeManager()));IRewardManager
rewardManager=IRewardManager(address(feeManager.i_rewardManager()));addressfeeTokenAddress=feeManager.i_linkAddress();
(Common.Assetmemoryfee,,)=feeManager.getFeeAndReward(address(this),reportData,feeTokenAddress);// Approve rewardManager to spend this contract's balance in
feesIERC20(feeTokenAddress).approve(address(rewardManager),fee.amount);// Verify the reportbytesmemoryverifiedReportData=verifier.verify(unverifiedReport,abi.encode(feeTokenAddress));//
Decode verified report data into BasicReport structBasicReportmemoryverifiedReport=abi.decode(verifiedReportData,(BasicReport));// Log price from reportemitPriceUpdate(verifiedReport.price);//
Store the price from the reportlast_retrieved_price=verifiedReport.price;}fallback()externalpayable{}}
```

Open in Remix What is Remix?

Initializing the upkeep contract

When deploying the contract, you define the verifier proxy address for the Data Streams feed you want to read from. You can find this address on the Data Streams Feed IDs page. The verifier proxy address provides functions that are required for this example:

- Thes_feeManagerfunction to estimate the verification fees.
- Theverifyfunction to verify the report onchain.

Emitting a log, retrieving, and verifying the report

After registering your upkeep contract with Chainlink Automation with a log trigger, you can emit a log with theemitLogfunction from your emitter contract.

1. The emitted log triggers the Chainlink Automation upkeep.
2. Chainlink Automation then usesStreamsLookupto retrieve a signed report from the Data Streams Engine, returns the data in a callback (checkCallback), and runs theperformUpkeepfunction on your registered upkeep contract.
3. TheperformUpkeepfunction calls theverifyfunction on the verifier contract to verify the report onchain.
4. In this example, theperformUpkeepfunction also stores the price from the report in thelast_retrieved_pricestate variable and emits aPriceUpdatelog message with the price.

Viewing the retrieved price

Thelast_retrieved_pricegetter function of your upkeep contract retrieves the last price stored by theperformUpkeepfunction in thelast_retrieved_pricestate variable of theStreamsUpkeepcontract. Additionally, theperformUpkeepfunction emits aPriceUpdatelog message with the retrieved price.

Optional: Handle Data Streams fetching errors offchain withcheckErrorHandler

When Automation detects the triggering event, it runs thecheckLogfunction of your upkeep contract, which includes aStreamsLookuprevert custom error. TheStreamsLookuprevert enables your upkeep to fetch a report from Data Streams. If the report is fetched successfully, thecheckCallbackfunction is evaluated offchain. Otherwise, thecheckErrorHandlerfunction is evaluated offchain to determine what Automation should do next.

In this example, thecheckErrorHandleris set to always returntrueforupkeepNeeded. This implies that the upkeep is always triggered, even if the report fetching fails. You can modify thecheckErrorHandlerfunction to handle errors offchain in a way that works for your specific use case. Read more aboutusing the StreamsLookup error handler.

Debugging StreamsLookup

Read ourdebugging section to learn how to identify and resolve common errors when usingStreamsLookup.