**Code Examples**

The code for reading Data Feeds is the same across all EVM-compatible blockchains and Data Feed types. You choose different types of feeds for different uses, but the request and response format are the same. To read a feed, specify the following variables:

* **RPC endpoint URL:** This determines which network that your smart contracts will run on. You can use a [node provider service](https://ethereum.org/en/developers/docs/nodes-and-clients/nodes-as-a-service/) or point to your own [client](https://ethereum.org/en/developers/docs/nodes-and-clients/). If you are using a Web3 wallet, it is already configured with the RPC endpoints for several networks and the [Remix IDE](https://remix-project.org/) will automatically detect them for you.
* **LINK token contract address:** The address for the LINK token contract is different for each network. You can find the full list of addresses for all supported networks on the [LINK Token Contracts](https://docs.chain.link/resources/link-token-contracts?parent=dataFeeds) page.
* **Feed contract address:** This determines which data feed your smart contract will read. Contract addresses are different for each network. You can find the availble contract addresses on the following pages:
  + [Price Feed Addresses](https://docs.chain.link/data-feeds/price-feeds/addresses)
  + [Proof of Reserve Feed Addresses](https://docs.chain.link/data-feeds/proof-of-reserve/addresses)
  + [NFT Floor Price Feed Addresses](https://docs.chain.link/data-feeds/nft-floor-price/addresses)

The examples in this document indicate these variables, but you can modify the examples to work on different networks and read different feeds.

This guide shows example code that reads data feeds using the following languages:

* On-chain consumer contracts:
  + [Solidity](https://docs.chain.link/data-feeds/examples#solidity)
  + [Vyper](https://docs.chain.link/data-feeds/examples#vyper)
* Off-chain reads using Web3 packages:
  + [Javascript](https://docs.chain.link/data-feeds/examples#javascript) with [web3.js](https://web3js.readthedocs.io/)
  + [Python](https://docs.chain.link/data-feeds/examples#python) with [Web3.py](https://web3py.readthedocs.io/en/stable/)
  + [Golang](https://docs.chain.link/data-feeds/examples#golang) with [go-ethereum](https://github.com/ethereum/go-ethereum)

**Reading data feeds on-chain**

These code examples demonstrate how to deploy a consumer contract on-chain that reads a data feed and stores the value.

**Solidity**

To consume price data, your smart contract should reference [AggregatorV3Interface](https://github.com/smartcontractkit/chainlink/blob/master/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol), which defines the external functions implemented by Data Feeds.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.7;

import "@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";

contract PriceConsumerV3 {

AggregatorV3Interface internal priceFeed;

/\*\*

\* Network: Sepolia

\* Aggregator: BTC/USD

\* Address: 0x1b44F3514812d835EB1BDB0acB33d3fA3351Ee43

\*/

constructor() {

priceFeed = AggregatorV3Interface(

0x1b44F3514812d835EB1BDB0acB33d3fA3351Ee43

);

}

/\*\*

\* Returns the latest price.

\*/

function getLatestPrice() public view returns (int) {

// prettier-ignore

(

/\* uint80 roundID \*/,

int price,

/\*uint startedAt\*/,

/\*uint timeStamp\*/,

/\*uint80 answeredInRound\*/

) = priceFeed.latestRoundData();

return price;

}

}

[Open in Remix](https://remix.ethereum.org/#url=https://docs.chain.link/samples/PriceFeeds/PriceConsumerV3.sol)[What is Remix?](https://docs.chain.link/getting-started/conceptual-overview#what-is-remix)

The latestRoundData function returns five values representing information about the latest price data. See the [Data Feeds API Reference](https://docs.chain.link/data-feeds/api-reference) for more details.

**Vyper**

To consume price data, your smart contract should import AggregatorV3Interface which defines the external functions implemented by Data Feeds. You can find it [here](https://github.com/smartcontractkit/apeworx-starter-kit/blob/main/contracts/interfaces/AggregatorV3Interface.vy). You can find a PriceConsumer example [here](https://github.com/smartcontractkit/apeworx-starter-kit/blob/main/contracts/PriceConsumer.vy). Read the ***apeworx-starter-kit*** [README](https://github.com/smartcontractkit/apeworx-starter-kit) to learn how to run the example.

**Reading data feeds off-chain**

These code examples demonstrate how to read data feeds directly off chain using Web3 packages for each language.

**Javascript**

This example uses [web3.js](https://web3js.readthedocs.io/) to retrieve feed data from the [BTC / USD feed](https://sepolia.etherscan.io/address/0x1b44F3514812d835EB1BDB0acB33d3fA3351Ee43) on the Sepolia testnet.

web3.jsethers.js

/\*\*

\* THIS IS EXAMPLE CODE THAT USES HARDCODED VALUES FOR CLARITY.

\* THIS IS EXAMPLE CODE THAT USES UN-AUDITED CODE.

\* DO NOT USE THIS CODE IN PRODUCTION.

\*/

const { ethers } = require("ethers") // for nodejs only

const provider = new ethers.providers.JsonRpcProvider("https://rpc.ankr.com/eth\_sepolia")

const aggregatorV3InterfaceABI = [

{

inputs: [],

name: "decimals",

outputs: [{ internalType: "uint8", name: "", type: "uint8" }],

stateMutability: "view",

type: "function",

},

{

inputs: [],

name: "description",

outputs: [{ internalType: "string", name: "", type: "string" }],

stateMutability: "view",

type: "function",

},

{

inputs: [{ internalType: "uint80", name: "\_roundId", type: "uint80" }],

name: "getRoundData",

outputs: [

{ internalType: "uint80", name: "roundId", type: "uint80" },

{ internalType: "int256", name: "answer", type: "int256" },

{ internalType: "uint256", name: "startedAt", type: "uint256" },

{ internalType: "uint256", name: "updatedAt", type: "uint256" },

{ internalType: "uint80", name: "answeredInRound", type: "uint80" },

],

stateMutability: "view",

type: "function",

},

{

inputs: [],

name: "latestRoundData",

outputs: [

{ internalType: "uint80", name: "roundId", type: "uint80" },

{ internalType: "int256", name: "answer", type: "int256" },

{ internalType: "uint256", name: "startedAt", type: "uint256" },

{ internalType: "uint256", name: "updatedAt", type: "uint256" },

{ internalType: "uint80", name: "answeredInRound", type: "uint80" },

],

stateMutability: "view",

type: "function",

},

{

inputs: [],

name: "version",

outputs: [{ internalType: "uint256", name: "", type: "uint256" }],

stateMutability: "view",

type: "function",

},

]

const addr = "0x1b44F3514812d835EB1BDB0acB33d3fA3351Ee43"

const priceFeed = new ethers.Contract(addr, aggregatorV3InterfaceABI, provider)

priceFeed.latestRoundData().then((roundData) => {

// Do something with roundData

console.log("Latest Round Data", roundData)

})

Latest Price:

Latest Price

**Python**

This example uses [Web3.py](https://web3py.readthedocs.io/en/stable/) to retrieve feed data from the [BTC / USD feed](https://sepolia.etherscan.io/address/0x1b44F3514812d835EB1BDB0acB33d3fA3351Ee43) on the Sepolia testnet.

# THIS IS EXAMPLE CODE THAT USES HARDCODED VALUES FOR CLARITY.

# THIS IS EXAMPLE CODE THAT USES UN-AUDITED CODE.

# DO NOT USE THIS CODE IN PRODUCTION.

from web3 import Web3

# Change this to use your own RPC URL

web3 = Web3(Web3.HTTPProvider('https://rpc.ankr.com/eth\_sepolia'))

# AggregatorV3Interface ABI

abi = '[{"inputs":[],"name":"decimals","outputs":[{"internalType":"uint8","name":"","type":"uint8"}],"stateMutability":"view","type":"function"},{"inputs":[],"name":"description","outputs":[{"internalType":"string","name":"","type":"string"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"uint80","name":"\_roundId","type":"uint80"}],"name":"getRoundData","outputs":[{"internalType":"uint80","name":"roundId","type":"uint80"},{"internalType":"int256","name":"answer","type":"int256"},{"internalType":"uint256","name":"startedAt","type":"uint256"},{"internalType":"uint256","name":"updatedAt","type":"uint256"},{"internalType":"uint80","name":"answeredInRound","type":"uint80"}],"stateMutability":"view","type":"function"},{"inputs":[],"name":"latestRoundData","outputs":[{"internalType":"uint80","name":"roundId","type":"uint80"},{"internalType":"int256","name":"answer","type":"int256"},{"internalType":"uint256","name":"startedAt","type":"uint256"},{"internalType":"uint256","name":"updatedAt","type":"uint256"},{"internalType":"uint80","name":"answeredInRound","type":"uint80"}],"stateMutability":"view","type":"function"},{"inputs":[],"name":"version","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"}]'

# Price Feed address

addr = '0x1b44F3514812d835EB1BDB0acB33d3fA3351Ee43'

# Set up contract instance

contract = web3.eth.contract(address=addr, abi=abi)

# Make call to latestRoundData()

latestData = contract.functions.latestRoundData().call()

print(latestData)

[Run this Python example](https://repl.it/@DwightLyle/GetLatestPriceWeb3PY)

**Golang**

You can find an example with all the source files [here](https://github.com/smartcontractkit/smart-contract-examples/tree/main/pricefeed-golang). This example uses [go-ethereum](https://github.com/ethereum/go-ethereum) to retrieve feed data from the [BTC / USD feed](https://sepolia.etherscan.io/address/0x1b44F3514812d835EB1BDB0acB33d3fA3351Ee43) on the Sepolia testnet. To learn how to run the example, see the [README](https://github.com/smartcontractkit/smart-contract-examples/blob/main/pricefeed-golang/README.md).

**NEW FEED REGISTRY**

You can use the [Feed Registry](https://docs.chain.link/data-feeds/feed-registry) to reference data feed assets by name or currency identifier instead of by pair/proxy address.

**Getting a different price denomination**

Chainlink Data Feeds can be used in combination to derive denominated price pairs in other currencies.

If you require a denomination other than what is provided, you can use two data feeds to derive the pair that you need. For example, if you needed a BTC / EUR price, you could take the BTC / USD feed and the EUR / USD feed and derive BTC / EUR using division.

Request Model Diagram

**IMPORTANT**

If your contracts require Solidity versions that are >=0.6.0 <0.8.0, use [OpenZeppelin's SafeMath version 3.4](https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v3.4/contracts/math/SafeMath.sol" \t "_blank).

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.7;

import "@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";

/\*\*

\* Network: Sepolia

\* Base: BTC/USD

\* Base Address: 0x1b44F3514812d835EB1BDB0acB33d3fA3351Ee43

\* Quote: EUR/USD

\* Quote Address: 0x1a81afB8146aeFfCFc5E50e8479e826E7D55b910

\* Decimals: 8

\*/

/\*\*

\* 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.

\*/

contract PriceConverter {

function getDerivedPrice(

address \_base,

address \_quote,

uint8 \_decimals

) public view returns (int256) {

require(

\_decimals > uint8(0) && \_decimals <= uint8(18),

"Invalid \_decimals"

);

int256 decimals = int256(10 \*\* uint256(\_decimals));

(, int256 basePrice, , , ) = AggregatorV3Interface(\_base)

.latestRoundData();

uint8 baseDecimals = AggregatorV3Interface(\_base).decimals();

basePrice = scalePrice(basePrice, baseDecimals, \_decimals);

(, int256 quotePrice, , , ) = AggregatorV3Interface(\_quote)

.latestRoundData();

uint8 quoteDecimals = AggregatorV3Interface(\_quote).decimals();

quotePrice = scalePrice(quotePrice, quoteDecimals, \_decimals);

return (basePrice \* decimals) / quotePrice;

}

function scalePrice(

int256 \_price,

uint8 \_priceDecimals,

uint8 \_decimals

) internal pure returns (int256) {

if (\_priceDecimals < \_decimals) {

return \_price \* int256(10 \*\* uint256(\_decimals - \_priceDecimals));

} else if (\_priceDecimals > \_decimals) {

return \_price / int256(10 \*\* uint256(\_priceDecimals - \_decimals));

}

return \_price;

}

}

[Open in Remix](https://remix.ethereum.org/#url=https://docs.chain.link/samples/PriceFeeds/PriceConverter.sol)[What is Remix?](https://docs.chain.link/getting-started/conceptual-overview#what-is-remix)

**More Aggregator Functions**

Getting the latest price is not the only data that aggregators can retrieve. You can also retrieve historical price data. To learn more, see the [Historical Price Data](https://docs.chain.link/data-feeds/historical-data) page.

To understand different use cases for Chainlink Price Feeds, refer to [Other Tutorials](https://docs.chain.link/getting-started/other-tutorials).