---

title: How to Set Up Tellor as your Oracle description: A guide to get started with integrating the Tellor oracle into your protocol author: "Tellor" lang: en tags: ["solidity", "smart contracts", "oracles"] skill: beginner published: 2021-06-29 source: Tellor Docs sourceUrl: https://docs.tellor.io/tellor/

Pop Quiz: Your protocol is just about finished, but it needs an oracle to get access to off chain data...What do you do?

(Soft) Prerequisites {#soft-prerequisites}

This post aims to make accessing an oracle feed as simple and straightforward as possible. That said, we're assuming the following about your coding skill-level to focus on the oracle aspect.

Assumptions:

- you can navigate a terminal
- you have npm installed
- you know how to use npm to manage dependencies

Tellor is a live and open-sourced oracle ready for implementation. This beginner's guide is here to showcase the ease with which one can get up and running with Tellor, providing your project with a fully decentralized and censorship-resistant oracle.

Overview {#overview}

Tellor is an oracle system where parties can request the value of an offchain data point (e.g. BTC/USD) and reporters compete to add this value to an onchain data-bank, accessible by all Ethereum smart contracts. The inputs to this data-bank are secured by a network of staked reporters. Tellor utilizes crypto-economic incentive mechanisms, rewarding honest data submissions by reporters and punishing bad actors through the issuance of Tellor’s token, Tributes (TRB), and a dispute mechanism.

In this tutorial we'll go over:

- Setting up the initial toolkit you'll need to get up and running.
- Walk through a simple example.
- List out testnet addresses of networks you currently can test Tellor on.

UsingTellor {#usingtellor}

The first thing you'll want to do is install the basic tools necessary for using Tellor as your oracle. Use this package to install the Tellor User Contracts:

```
npm install usingtellor
```

Once installed this will allow your contracts to inherit the functions from the contract 'UsingTellor'.

Great! Now that you've got the tools ready, let's go through a simple exercise where we retrieve the bitcoin price:

BTC/USD Example {#btcusd-example}

Inherit the UsingTellor contract, passing the Tellor address as a constructor argument:

Here's an example:

```solidity import "usingtellor/contracts/UsingTellor.sol";

contract PriceContract is UsingTellor { uint256 public btcPrice;

//This Contract now has access to all functions in UsingTellor
```

constructor(address payable _tellorAddress) UsingTellor(_tellorAddress) public {}

function setBtcPrice() public { bytes memory _b = abi.encode("SpotPrice",abi.encode("btc","usd")); bytes32 _queryId = keccak256(_b);

```
uint256 _timestamp;
bytes _value;

(_value, _timestamp) = getDataBefore(_queryId, block.timestamp - 15 minutes);

btcPrice = abi.decode(_value,(uint256));
```

} } ```

For a complete list of contract addresses refer here.

For ease of use, the UsingTellor repo comes with a version of the Tellor Playground contract for easier integration. See here for a list of helpful functions.

For a more robust implementation of the Tellor oracle, check out the full list of available functions here.