Collateral Onboarding Guide for Oracle Domain Teams

Directory

1. Overview

The Maker Protocol enables users to generate Dai against an array of assets, including ETH, WBTC, and USDC among others. In order to ensure that all collateral assets are priced correctly in the Protocol, an Oracle system is used to provide a real-time stream of market price data for each asset in a secure and resilient manner. The Oracle system needs to be resistant to attacks, be they at the technical, crypto-economic, or informational level.

This price information provided by Oracles is utilized to determine whether Maker Vault positions are sufficiently capitalized or below the collateralization ratio and subject to liquidation. Oracle price information also determines the maximum amount of Dai a user can generate against their collateral. Given the widespread effects the Oracles have on the Maker Protocol, it is imperative that the Data Models used in their implementations are refined to mitigate risk and optimize performance. These Data Models define how data is sourced and how it is filtered into a canonical price.

This guide captures the best practices of long-time Community smart contract developers as well as the current Oracle Domain Team. It intends to show developers interested in working for the Oracle Domain Team or independently contributing to MakerDAO how the Oracle System and its component parts function and are built upon. We hope that this guide evolves as the Community improves on the processes included herein and as the number of Oracle Domain Teams expands. Thanks for reading!

1. Learning Objectives

2. Learn about the Oracles Domain Work

3. Learn how to onboard an Oracle for collateral onboarding through the MIP10 framework.
4.

1. Prerequisites

2. Introduction to the Maker Protocol

3. Oracle Module
4. MIPs
5. Solc 0.5.12
6. Dapp tools
7. Familiarity withseth and dapp
8. .
9. Workingnix environment
10. for testing.
11.

1. Introduction to MIP10c3

MIP10 defines Oracle Management in the Maker Protocol, including onboarding an Oracle, updating a Data Model, and performing other tasks that ensure Maker Oracles function properly.MIP10c3 specifically covers the onboarding of a new Oracle, where Oracle is defined as the on-chain price source. Onboarding a new collateral on the Maker Protocol requires creating a new Oracle, and having theOracle Feeds updated to start serving it.

Below, you can find all the steps defined in MIP10c3. While additional information and best practices are provided, they are not formally part of the MIP specifications.

This document does not cover the inclusion of Oracle customers to the Maker Protocol.

Overview of the collateral onboarding process

1. Technical Complexity Evaluation
2.
   - Produce an assessment of the planned collateral
3. *
4. Oracle Tooling Software Updates
5.
   - Adapt the Oracle software to gather pricing information on the collateral and use the deployed smart contracts
6. *
7. Oracle Smart Contract Preparation
8.
   - Make the necessary modifications to the contract and compile them.
9. *
10. Integration Testing
11.
    - Run a single Feed and Relayer to confirm they interface well with each other and with the smart contracts.
12. *
13. Deploy Kovan and Mainnet Smart Contracts

14.
    - Deploy the compile contract to Kovan and Mainnet with the proper permissions
15. *
16. Finalize and Publish Tooling Software Updates
17.
    - Final update to the software to point to the deployed mainnet contract.
18. *
19. Publish the Collateral Onboarding Oracle Assessment (MIP10c3-SP)
20.
    - Use the sub-proposal template to gather the information.
21. *
22. Staging Tests
23.
    - Long-running test to ensure stability
24. *
25. Deployment of Oracle Software to feeds and relayers
26.
    - Coordinate with the dark feeds, light feeds, and relayers to update to the new software version and monitor the freshly updated network.
27. *
28.

1. Technical Complexity Evaluation

Determine the complexity of Oracle integration for the collateral type and how long it would take to implement such a solution.

Data Sources

First, determine the number and availability of public data sources needed to construct a reliable, resilient, and secure Oracle for the proposed collateral asset. Start by looking at the list of the exchanges where the collateral is currently being traded, which can be obtained fromCoinGecko andMessari , or from the promoters of the token listing. The exchanges listed will vary in quality; therefore, take into account the following criteria when evaluating an exchange as a potential price source:

- Trading volume and order book depth: We are looking for consistently deep order books and large historic trading volume. Taken together, these two criteria ensure that the price of the collateral on that exchange will not be significantly more volatile than the collateral itself (i.e., large orders will not skew the price significantly more than the other sources).
- Exchange reputation: Some exchanges report inflated numbers, due in part to wash trading. Consult industry resources, such asMessari.io
- 's “Real Volume” list of trusted exchanges to assess the exchange reputation. Even if some exchanges might exhibit some wash trading, it might still be used as a data source if it’s found out that it has a significant proportion of legitimate trading.
- Availability of an API: Does the exchange make available a public API that can be consulted by the price feeds.
- Existing exchange integration: Exchanges already integrated with the Protocol are favored because this reduces the amount of work required for integrating the price source in the existing tooling.
-

For certain collateral types, crypto exchanges might not be the right source of pricing data, and particular rules might apply. For instance

- For some stablecoins, it might be desirable for their price to be a fixed value.
- Wrapped assets for which the protocol uses the underlying asset price instead of the wrapped version. Ex.: WBTC that is priced assuming a 1:1 equivalent with BTC.
-

Data Models

The required data models for the new collateral must be easily implementable and not require complex development. Ideally, the data model will be usable directly by the existing tooling without modification. If modifications are needed, provide an estimate of the time and effort required to implement.

Examples of functionality that might require extra development:

- Use of on-chain sources
- Authenticated/private API
- Non-HTTP API
- Use of a function other than the median for aggregation, such as median or other special rules such as removing the lowest and highest values.
-

Externalities

As part of the technical complexity evaluation, external dependencies that the integration of a price for feed would add to the Oracle must be identified. The objective is to minimize adding risks, costs, and latency to the Oracle Protocol that would have to be managed in the long term.

Example:

- Paid API access or free API access that could be throttled or disabled.
- Slow API sources or data that requires a significant amount of processing.
- High reliance on a specific price feed.
-

Timeline

The Oracle Domain Team must ensure that it can deliver the price source in the timeline expected by the community:

- Does adding these features interfere with ongoing or planned development of new tooling?
- Does adding these features require work and/or coordination from other stakeholders, such as the Smart Contracts Team(s)?
- How long might it take to design, implement, test, and deploy such a solution?
-

1. Build the Data Model

The data model for a collateral type is the formula used by the oracles to determine its price. It is comprised of:

- List of price sources: A list of trading pairs that include the collateral, typically coming from 5 to 6 exchanges.

- Quorum: Minimum number of feeds necessary to provide price information for the Oracle price to be considered valid.
- Feed models: How the individual feeds aggregate the price sources. Typically, “median” is used.
- Oracle models: How the Oracle will aggregate the price information from the individual feeds. Typically, “median” is used.
-

If the collateral isn’t traded directly to USD, supporting data models will be required (e.g., XYZ eExchange trades ETH with BTC. To use this price source, a BTC/USD data model will be required. A certain number of supporting data models are already used and supported (ETH/USD, BTC/USD, and USD/TUSD).

Oracle Data Model

```

Copy | Source | Asset Pair | Quorum | Feed Model | Oracle Model | | :--------------- | :--------- | :-----: | :---------: | :----------: | | < data source > | < param > | < # > | < model > | < model > |

```

Example (PAXG/USD):

```

Copy | Source | Asset Pair |Quorum | Feed Model | Oracle Model | | :----------- | :------------ | :---: | :---------: | :----------: | | Binance | PAXG/USDT | 13 | Median | Median | | Bitthumb | PAXG/USDT | | Gemini | PAXG/USD | | Kraken | PAXG/USD | | Uniswap | PAXG/ETH |

```

Oracle Supporting Data Model(s)

```

Copy | Source | Asset Pair | Feed Model | | :------------ | :------------ | :----------: | | < data src> | | |

```

Example (USDT/USD):

```

Copy | Source | Asset Pair | Feed Model | | :------------ | :------------ | :----------: | | Binance | BTC/USDT | Median | | BitFinex | USDT/USD | | | FTX | ETH/USDT | | | Huobi | ETH/USDT | | | Kraken | USDT/USD | | | OKEx | BTC/USDT | |

```

1. Oracle Tooling Software Updates

Support of the new Data Model requires updating the off-chain software run by feeds and relayers. Three major pieces of software currently used, setzer, gofer and omnia.

setzer

setzer is the tool used to query the prices from the off-chain sources. It queries a list of external APIs and aggregates the values returned. The aggregated result can then be used by omnia. Oracle domain contributors are invited to fork theGitHub repo and submit a pull request for inclusion in the main branch.

Each collateral price has asetzer-price-XXXusd that returns the current price of the collateral. For instance, if you want to know the current price of Ether, you would runbin/setzer price ETHUSD and obtain the current price:351.2382000000 .

Calling thebin/setzer script translates the request to a call to the scriptlibexec/setzer/setzer-price , which in turn will invokelibexec/setzer/setzer-price-ethusd .

Adding a price script

To add a new collateral, asetzer-price-XXXusd script must be created. Let's take, for example, PAXG, for which the following data model was defined:

```

Copy | Source | Asset Pair |Quorum | Feed Model | Oracle Model | | :----------- | :------------ | :---: | :---------: | :----------: | | Binance | PAXG/USDT | 13 | Median | Median | | Bitthumb | PAXG/USDT | | Gemini | PAXG/USD | | Kraken | PAXG/USD | | Uniswap | PAXG/ETH |

```

The script starts off by setting the pair name, herepaxgusd .

```

Copy

!/usr/bin/env bash

set -e pair=paxgusd

```

Then the list of exchanges used for finding its price is defined. These exchanges were previously defined in setzer-x-price.

```

Copy sources=( binance bitthumb gemini kraken uniswap )

```

The easiest case to cover is a pair against the USD, as it doesn’t require secondary models. Here, the PAXG/USD prices from Gemini and Kraken fit this criteria.

```
Copy case 1 in gemini|kraken) { setzer x-price "1" paxg:usd };;
```

In many cases, a second price must be added to the calculation. For instance, for PAXG/USDT, the USDT price has to be converted to USD.

```
Copy binance|bitthumb) { paxg_usdt=(setzer x-price "1" paxg:usdt) usdt_usd=(setzer price usdtusd) setzer --format "(bc -l <<<"paxg_usdt * usdt_usd")" };;
```

Same thing with PAXG/ETH, for which ETH/USD is required to determine the USD price.

```
Copy uniswap) { paxg_eth=(setzer x-price "1" paxg:eth) eth_usd=(setzer price ethusd) setzer --format "(bc -l <<<"paxg_eth * eth_usd")" };;
```

The script is completed by covering the default case where the commands ls, sources and median are passed to thesetzer---price-commands script.

```
Copy *) { setzer --price-commands "-1-" pair "{sources[@]}" };; esac
```

Testing the script is done by invoking it directly:

```
Copy chmod a+x setzer-price-paxgusd ./setzer-price-paxgusd binance 1922.4023458415 ./setzer-price-paxgusd kraken 1922.0000000000 ./setzer-price-paxgusd gemini 1923.8900000000 ./setzer-price-paxgusd uniswap 1927.3950381315 ./setzer-price-paxgusd bitthumb 1838.4216082708
```

Testing the main setzer script:

```
Copy bin/setzer pairs | grep PXGUSD PXGUSD

bin/setzer price pxgusd 1923.7400000000
```

Adding a price source

If it is necessary to support a new price source, such as a new exchange or a different API, the scriptsetzer-x-price must be modified.

For instance, to change the data model for PAXG/USD to use the PAXG/USDT pricing from the Probit exchange, it is required to determine the API call that will provide the information. In this case, researching theProbit API documentation will allow crafting this HTTP request:

```
Copy echo json=(curl -sS https://api.probit.com/api/exchange/v1/ticker?market_ids=PAXG-USDT) json={"data":[{"last":"1930","low":"1892","high":"1949","change":"14","base_volume":"4.86095411","quote_volume":"9299.20180356","market_id":"PAXG-USDT","time":"2020-10-05T19:38:27.000Z"}
```

jshon can extract the price from the json response:

```
Copy jshon <<<"json" -e data -e 0 -e last -u 1926
```

The following code can then be added tosetzer-x-price .

```
Copy probit) { json=(curl -sS "https://api.probit.com/api/exchange/v1/ticker?market_ids={base^^}-{quote^^}") setzer --format "(jshon <<<"json" -e data -e 0 -e last -u)" };;
```

After adding Probit to thesetzer-x-price script, the new source can be tested by directly invokingsetzer :

```
Copy bin/setzer price paxgusd probit 1926.6460884486
```

gofer

Gofer is another tool used to query the prices from the off-chain sources. If you don’t need to define new external APIs, no code update is needed and only a config change is required. See the section “Finalize and Publish Tooling Software Updates” below for the configuration updates required.

omnia

omnia is the script that periodically calls setzer to obtain price data. When it determines that the price it provided previously needs to be updated, it signs a message that is distributed using the scuttlebutt network.

Adding new collateral prices to omnia requires updating three files.

- oracles-v2/nix/srcs.nix
- to bump the setzer rev with the commit hash. If the setzer PR isn't merged, updateurl
- andref
- to point our work branch to test omnia. Make sure to update it later to point to the main branch once the setzer PR is merged.
- omnia/config/feed.conf
- to add the new pair. Use defaultmsgSpread
- andmsgExpirations
- :
-

Example:

```

Copy "PAXGUSD": { "msgExpiration": 1800, "msgSpread": 1 }

```

- omnia/config/feed.conf
- to add the new pair, including the Medianizer address, which will be updated once it is deployed.
-

Example:

```

Copy "PAXGUSD": { "oracle": "0x", "oracleSpread": 1.0, "oracleExpiration": 15500, "msgExpiration": 1800 }

```

1. Prepare the Medianizer and OSM Contracts

Collateral prices are accessed by the Maker Protocol via the Oracle Security Module smart contract. This contract gets its pricing information from the Medianizer, which is the smart contract that the information is pushed to by the Oracle Relayers.

The Oracle Domain Team is responsible for the Medianizer and OSM contracts. As part of the collateral onboarding process, the Medianizer and OSM smart contracts are deployed on both the Kovan testnet and the Ethereum mainnet. Care must be taken while deploying to ensure that the right permissions are set correctly:

- Feed addresses
- Medianizer read-only access
- OSM read-only access
- Ownership of both contracts.
-

Always verify the latest contract addresses of the Maker Protocol before continuing with the implementation. Source thechangelog for the latest contracts.

Medianizer

Medianizer is responsible for taking the pricing information provided by the Relayers, and verifying that the information is signed by the authorized list of Feeds.

1. Clone the smart contract from the repohttps://github.com/makerdao/median
2. Insrc/median.sol
3. , paste the following code snippet at the end, while changing the contract name and the value of thewat
4. variable.
5.

```

Copy contract MedianPAXGUSD is Median { bytes32 public constant wat = "PAXGUSD";

function recover(uint256 val_, uint256 age_, uint8 v, bytes32 r, bytes32 s) internal pure returns (address) { return ecrecover( keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", keccak256(abi.encodePacked(val_, age_, wat)))), v, r, s ); } }

```

1. Update the dependencies:dapp update
2. Compile with solc 0.5.12:dapp --use solc:0.5.12 build
3.

Oracle Security Module (OSM)

The Oracle Security Module delays the price information provided by the Medianizer contract for use by the Maker Protocol. The OSM does not require modifications, except for passing the address of the Medianizer contract to the constructor.

1. Clone the smart contract from the the repohttps://github.com/makerdao/osm/
2. Update the dependencies:dapp update
3. Compile the contract: dapp --use solc:0.5.12 build
4.

1. Integration Testing

Deploy and Configure Integration Testing Contracts

SetETH_RPC_URL andETH_KEYSTORE to point to the corresponding ethereum node and account for Kovan or another testnet. Deploying smart contracts also requires a fairly large gas limit, such asETH_GAS=5000000 .

1. Deploy the smart contract usingdapp create
2. (Use the contract name you used above, such asMedianPAXGUSD
3.)
4. Note the returned address, this is the contract address that will store in MEDIAN for the other steps.
5. Verify the smart contract on Etherscan by using the flatten source code provided by:hevm flatten --source-file src/median.sol --json-file out/dapp.sol.json
6. Add the list of approved feeds to the contract usinglift
7. . Put the feed address list in theFEEDS
8. variable. For integration testing, use a feed account on which you have control, as you will test omnia message signatures. The oracle-v2 repo conveniently includes a test account in thesmoke-tests
9. directory,0x1f8fbe73820765677e68eb6e933dcb3c94c9b708
10. .
11.

```

Copy FEEDS=("0x1f8fbe73820765677e68eb6e933dcb3c94c9b708") seth send MEDIAN "lift(address[] memory)" "(echo {FEEDS[@]} | sed s/\ /\",\"/g)"

```

1. Set the quorum (i.e., the minimum number of feeds required to make the price valid) in the data model. For the testing, set it to 1, as there is only one feed.
2.

```

Copy seth send MEDIAN "setBar(uint256)" (seth --to-uint256 1)

```

1. Deploy the OSM smart contract:dapp create OSM MEDIAN
2. . Note the returned address, this is the contract address that will store in OSM for the other steps.
3. Verify the smart contract on Etherscan by using the flatten source code provided by:
4.

```

Copy hevm flatten --source-file src/osm.sol --json-file out/dapp.sol.json

```

Etherscan should have already set correctly the Constructor Arguments, which is the Medianizer address.

1. Give OSM access to Medianizer
2.

```

Copy seth send MEDIAN "kiss(address)(uint256)" OSM

```

Without this permission, apoke() to the OSM would fail because it requires the OSM to be able to query the price from the Medianizer.

Run the tests

As a testing prerequisite, install Nix, if you haven't done already

- nix curl -L https://nixos.org/nix/install | sh
- . ~/.nix-profile/etc/profile.d/nix.sh
-

Omnia Feed test

From your oracle-v2 directory:

- cp omnia/config/feed.conf /tmp/omnia.config
- export OMNIA_CONFIG=/tmp/omnia.config
- In/tmp/omnia.conf
- , setfrom, keystore
- andpassword
- to a test account included with the oracle-v2 repo:
-

```

Copy "ethereum": { "from": "0x1f8fbe73820765677e68eb6e933dcb3c94c9b708", "keystore": "../smoke-tests/resources/keys", "password": "../smoke-tests/resources/key" },

```

- Executenix-shell
- (first run will take a long time)
-
    - cd omnia
-
    - ssb-server start &
-
    - ./omnia.sh

- *
-

Confirm that omnia is able to pull the price for your collateral and publish it to the network. You will get an output similar to this:

```
Copy [10/09/20 14:40:47] [V] { "type": "PXGUSD", "version": "1.3.9", "price": 1932.6067700778, "priceHex":
"000000000000000000000000000000000000000000000068c44e81ff0a4aea00", "time": 1602254445, "timeHex":
"000000000000000000000000000000000000000000000000000000005f80766d", "hash":
"6b48ef84fa4484438d791089d4966053134dc0ce68c1935df1c60ef067758723", "signature":
"87cf8fc4e30c0939de3da63f7c8b228dfdad1e80db43783d7316bde1d62eb3285d24b069d58900652dc6589a3cce83af99e7a6c28fd767efa9537c962e614b3d1b",
"sources": { "gemini": "1931.5200000000", "uniswap": "1955.8511034425", "binance": "1933.6935401555", "kraken": "1930.0900000000", "probit":
"1934.0138737164", "bitthumb": "1876.9444477637" } } [10/09/20 14:40:47] Publishing new price message... { "key":
"%Kw2cAauVwC0H8CYjYooiupHq1Owvv2f8llOKlP40kjw=.sha256", "value": { "previous":
"%8lTgn0kCsFP5sG8JNkTGFWdsHPoqs+Uq5uSouI05Tc0=.sha256", "sequence": 17, "author":
"@rE23z01yWbc/dOix9rIHFlkBj1OCWWckNkSxEG5E7mw=.ed25519", "timestamp": 1602254447731, "hash": "sha256", "content": { "type": "PXGUSD",
"version": "1.3.9", "price": 1932.6067700778, "priceHex": "000000000000000000000000000000000000000000000068c44e81ff0a4aea00", "time":
1602254445, "timeHex": "000000000000000000000000000000000000000000000000000000005f80766d", "hash":
"6b48ef84fa4484438d791089d4966053134dc0ce68c1935df1c60ef067758723", "signature":
"87cf8fc4e30c0939de3da63f7c8b228dfdad1e80db43783d7316bde1d62eb3285d24b069d58900652dc6589a3cce83af99e7a6c28fd767efa9537c962e614b3d1b",
"sources": { "gemini": "1931.5200000000", "uniswap": "1955.8511034425", "binance": "1933.6935401555", "kraken": "1930.0900000000", "probit":
"1934.0138737164", "bitthumb": "1876.9444477637" } }, "signature":
"Wx+YRJrhZAJt/K98MLHKix5G1Hxj8t4iaHVy5gXrWz/R09CDaoDNq5wXvMBwmZ4WujPUoAMISm0IKXRTGtvyDg==.sig.ed25519" }, "timestamp":
1602254447732 }
```

Omnia Relayer test

From your oracle-v2 directory:

- cp omnia/config/relayer.conf /tmp/omnia-relayer.conf
- export OMNIA_CONFIG=/tmp/omnia-relayer.conf
- In/tmp/omnia-relayer.conf
- , setfrom,
- keystore
- ,password
- to the values corresponding to any funded account on the testnet you are using. Setnetwork
- to a node RPC URL, such as one from infura.io. Set the oracle address for your collateral to your integration contract address.
- nix-shell
-
    - cd omnia
-
    - If you stopped it, restart ssb usingssb-server start &
-
    - ./omnia.sh
- *
- The Omnia Relayer will read the published message from the Feed, and push it on-chain.
- Make sure that the transaction succeeds (SUCCESS: 1). If necessary, troubleshoot the reverted ethereum transaction.
-

```
Copy [10/09/20 15:28:01] Sending tx... TX: 0x... SUCCESS: 1 GAS USED: 58713 [10/09/20 15:28:10] [V] Sleeping 60 seconds..
```

Validation

Now that Omnia successfully pushed the price information on-chain, confirm that OSM is able to read it properly.

- Poke the OSM contract to read the current value from the Medianizer:seth send OSM "poke()()"
- Read the on-chain value using:
-

```
Copy storage=(seth storage OSM 0x4) price=(seth --from-wei "(seth --to-dec "{storage:34:32}")") echo price 1934.814612130400000000
```

If these steps are successful and that OSM has the price your Feed published earlier, you may proceed to Kovan and Mainnet contract deployments.

6.Kovan and Mainnet Contract Deployment

Follow the steps below for deployment on both Kovan and Mainnet.

1. SetETH_RPC_URL
2. andETH_KEYSTORE
3. to point to the corresponding ethereum node and a funded account.
4. Deploy the smart contract usingdapp create MedianPAXGUSD
5. (updating the contract name) Note the returned address, this is the contract address that will store inMEDIAN
6. for the other steps.
7. Verify the smart contract on Etherscan by using the flatten source code provided by:hevm flatten --source-file src/median.sol --json-file out/dapp.sol.json
8. Add the list of approved feeds to the contract usinglift
9. . Put the feed address list in theFEEDS
10. variable. For Kovan, coordinate with other Oracle Domain Teams to obtain the latest feed list. If necessary, you may extract this list from an existing Medianizer contract using the following script but changing the address to a known working median contract on Kovan:
11.

```

Copy

# !/bin/bash

for i in {0..255} do addr=(seth call 0x64DE91F5A373Cd4c28de3600cB34C7C6cE410C85 'slot(uint8)' i) if [ addr != "0x0000000000000000000000000000000000000000000000000000000000000000" ] then echo "0x(cut -c27-66 <<< addr)" fi done

```

For mainnet, obtain the list of currently approved production light and dark feeds fromMIP10c7 , and compare it with the live feed list from one of the production Medianizer. Coordinate with the other Oracle Domain Teams if you find a discrepancy.

```

Copy FEEDS=("0x1f8fbe73820765677e68eb6e933dcb3c94c9b708")

seth send MEDIAN "lift(address[] memory)" "(echo {FEEDS[@]} | sed s/\ /\",\"/g)"

```

1. Set the quorum (i.e., the minimum number of feeds required to make the price valid) in the data model. Here, it is set to 13:
2.

```

Copy seth send MEDIAN "setBar(uint256)" (seth --to-uint256 13)

```

1. Add the Maker Governance DSPause Proxy as one of the owners of the contract (MCD_PAUSE_PROXY
2. onchangelog.makerdao.com
3.). Note that this does not remove the deployer address as owner. You will need to remove yourselves (deny()
4.) after deploying the OSM.
5.

```

Copy seth send MEDIAN "rely(address)" "0x0e4725db88Bb038bBa4C4723e91Ba183BE11eDf3"

```

1. Deploy the OSM smart contract:dapp create OSM MEDIAN
2. . Note the returned address, this is the contract address and store it in OSM for the other steps.
3. Verify the smart contract on Etherscan by using the flatten source code provided by:
4.

```

Copy hevm flatten --source-file src/osm.sol --json-file out/dapp.sol.json

```

Etherscan should have already set correctly the Constructor Arguments, which is the Medianizer address.

1. Give OSM access to Medianizer
2.

```

Copy seth send MEDIAN "kiss(address)(uint256)" OSM

```

1. Add the Maker Governance DSPause Proxy as one of the owners of the contract (MCD_PAUSE_PROXY
2. onchangelog.makerdao.com
3.).
4.

```

Copy seth send OSM "rely(address)" "0x0e4725db88Bb038bBa4C4723e91Ba183BE11eDf3"

```

1. Enable read access to the Maker Protocol
2.

This step enables read access for the MCD spotter contract, so it can access the prices. Set the address toMCD_SPOT fromchangelog.makerdao.com

```

Copy seth send OSM "kiss(address)" 0x3a042de6413eDB15F2784f2f97cC68C7E9750b2D

```

You may confirm that the spotter has access by checkingbud

```

Copy seth call OSM "bud(address)(uint256)" 0x3a042de6413eDB15F2784f2f97cC68C7E9750b2D
```

```

Permission tightening

Once you confirm that everything has been deployed correctly, with all the right permissions, remove yourself as owner from both the OSM and Medianizer

```

Copy seth send OSM "deny(address)" ETH_FROM

seth send MEDIAN "deny(address)" ETH_FROM

```

1. Finalize and Publish Tooling Software Updates

2. Submit a PR for Setzer

3. Make sure you have testedsetzer price

4. as described above before submitting your pull request.
5.

6. Submit a PR for Omnia

Because Omnia configuration depends on the availability of a specific setzer version and deployed mainnet contracts, we have to wait for the Setzer PR to be merged to mainnet and for the availability of a deployed median contract on mainnet.

- Updateoracles-v2/nix/srcs.nix
- to bump the setzer rev with the commit hash from the mainnet branch.
- Updateoracles-v2/omnia/config/feed.conf
- ,oracles-v2/omnia/config/relayer.conf
- andoracles-v2/omnia/config/relayer-kovan.conf
- to point to the mainnet deployment of the Medianizer contract.
- Updateoracles-v2/systemd/gofer.json
- to add the data model for your new collateral. See thegofer documentation
- for more details and examples.
-

1. Staging Tests

To ensure software stability and avoid dangerous oracle outages in production, it is necessary to run the new Oracle Software and deploy smart contracts for a period of 7 to 14 days on at least 4 feeds and 2 relayers on a testnet environment.

To facilitate these steps, terraform configuration and deployment scripts are made available in adedicated repository . These sample configuration and script allow for the quick and flexible deployment of testing environments using a cloud provider.

The existing Oracle Team maintains an Oracle staging network on Kovan that can be used for this testing. Coordinate with the Oracle Team to join this network, or consider deploying your own.

1. Publish the Collateral Onboarding Oracle Assessment (MIP10c3-SP)

Use the officialMIP10c3-SP template to create a new sub-proposal for the collateral. To be able to complete it, you will need the following information:

- Data Model
- Supporting Data Model
- Mainnet contract addresses for Medianizer and OSM contracts
- Git tags for Setzer and Omnia releases
-

Note that the template is also used to define the Oracle customers (other than the Maker Protocol). Reviewing the inclusion of such customers is not in the scope of this document.

Once completed, submit a PR to the official MIPrepo and publish a post to theforum with the content of the SP.

1. Deployment of Oracle Software to feeds and relayers

In preparation for the addition of collateral to the Maker Protocol, the Oracle Team will coordinate with the dark feeds, light feeds, and relayers to update to the new software version and monitor the freshly updated network.

Feed providers need to always be alert to new updates of their clients and update their clients to latest changes.

Every time there's an update to the configuration in the oracles feed software, every other feed provider must update to that configuration so as to not run into issues when adding new feeds in the future. In other words, every feed provider must have the same version of the software client, with the same configuration.

1. Summary

In this guide, you were introduced to the MIP10c3 process of setting up an oracle for a new collateral that is to be onboarded to the Maker Protocol. The steps involved were selecting the right data sources, updating the software stack, deploying the smart contracts and testing it. The final step was to put it all together for the Maker Governance approval by using the MIP10c3 template. With this guide, you gathered the knowledge to help the Maker Protocol become more decentralized by learning the Oracle Domain Work and help onboard new collaterals into the protocol.

Previous RWA Legal Documentation Operations GuideNext Collateral Technology Last updated1 year ago
```