Deposit Box

Simple deposit box example that showcases the basic request propose lifecycle of the UMA Optimistic Oracle

UMA Contract Lifecycle

UMA's Optimistic Oracle allows contracts to quickly request and receive price information. A request is made when a contract submits the following parameters with a request to the Optimistic Oracle contract:

- identifier:
- price identifier being requested.
- timestamp:
- timestamp of the price being requested.
- ancillaryData:
- additional arguments passed with the price request.
- currency:
- ERC20 token used for payment of rewards and fees. Must be approved for use with the DVM.
- reward:
- reward offered to a successful proposer. Will be paid by the caller. Note: this can be 0.
-

Proposers respond to price requests by referencing off-chain price feeds to submit the price of an asset. In return for their work they will receive a pre-defined proposal reward set by the Requestor. To propose prices, the Proposer is required to stake a proposal bond. In the event that the price information they proposed is disputed and deemed incorrect, the Proposer will lose their bond.

Disputers can refute a price submitted by a Proposer within the proposal liveness period by referencing their own off-chain price feeds. The proposal liveness period is a pre-defined amount of time a proposal can be disputed before the Requestor receives the price of the asset.

If Disputers do not refute the price submitted by the Proposer within the proposal liveness period, the price is sent to the Requestor. If a proposal is disputed, the price will be submitted to UMA’s DVM and resolved after a 48-96 hour voting period.

Integrating with the Optimistic Oracle

We continue to use theOptimisticDepositBox contract as an example. The OptimisticDepositBox is a minimal financial contract that allows a user to deposit collateral into a contract and later withdraw their collateral corresponding to a desired USD amount. When the user wants to make a withdrawal, a price request is made to the Optimistic Oracle.

Let's first take a look at the OptimisticDepositBox constructor. Optimistic Oracle price requests require the use of a whitelisted identifier and collateral. The OptimisticDepositBox uses the protocol Finder to discover UMA protocol contracts and call theisOnWhitelist method on theAddressWhitelist and theisIdentifierSupported method on theIdentifierWhitelist contract to confirm both are whitelisted before deployment.

```

Copy constructor( address_collateralAddress, address_finderAddress, bytes32_priceIdentifier, address_timerAddress )nonReentrant()Testable(_timerAddress) { finder=FinderInterface(_finderAddress); require(_getIdentifierWhitelist().isIdentifierSupported(_priceIdentifier),"Unsupported price identifier"); require(_getAddressWhitelist().isOnWhitelist(_collateralAddress),"Unsupported collateral type"); collateralCurrency=IERC20(_collateralAddress); priceIdentifier=_priceIdentifier; }

...

function_getIdentifierWhitelist()internalviewreturns(IdentifierWhitelistInterface) { returnIdentifierWhitelistInterface(finder.getImplementationAddress(OracleInterfaces.IdentifierWhitelist)); }

function_getAddressWhitelist()internalviewreturns(AddressWhitelist) { returnAddressWhitelist(finder.getImplementationAddress(OracleInterfaces.CollateralWhitelist)); }

```

Example arguments used to deploy the contract can be found and tested in theOptimisticDepositBox fixture . A whitelisted USDC contract is used for collateral, the Finder and Timer addresses use existing UMA ecosystem contracts that are deployed, and a price identifier that has beenwhitelisted with the IdentifierWhitelist contract.

```

Copy // Deploy the OptimisticDepositBox contract. constoptimisticDepositBox=await(
```

awaitgetContractFactory("OptimisticDepositBox",deployer)
).deploy(usdc.address,parentFixture.finder.address,identifier,parentFixture.timer.address);

```

Requesting a Price

A price request is made when therequestWithdrawal method is called. A depositor submits thedenominatedCollateralAmount to withdrawal and arequestPrice call is constructed with the following argumentsshown above and in therequestPrice interface :

```

Copy functionrequestWithdrawal(uint256denominatedCollateralAmount) public noPendingWithdrawal(msg.sender) nonReentrant { OptimisticDepositBoxDatastoragedepositBoxData=depositBoxes[msg.sender]; require(denominatedCollateralAmount>0,"Invalid collateral amount");

// Update the position data for the user. depositBoxData.withdrawalRequestAmount=denominatedCollateralAmount; depositBoxData.withdrawalRequestTimestamp=getCurrentTime();

emitRequestWithdrawal(msg.sender,denominatedCollateralAmount,depositBoxData.withdrawalRequestTimestamp);

// A price request is sent for the current timestamp. _requestOraclePrice(depositBoxData.withdrawalRequestTimestamp); }

.....

// Requests a price for priceIdentifier at requestedTime from the Optimistic Oracle. function_requestOraclePrice(uint256requestedTime)internal{ OptimisticOracleInterface oracle=_getOptimisticOracle(); // For other use cases, you may need ancillary data or a reward. Here, they are both zero. oracle.requestPrice(priceIdentifier,requestedTime,"",IERC20(collateralCurrency),0); }

```

After price request functionality has been implemented into your contracts, you can test your changes by creating a test in the developer quickstart repo similar to the OptimisticDepositBox test below:

```

Copy awaitexpect(optimisticeDepositBox.connect(depositor).requestWithdrawal(amountToWithdraw)) .to.emit(optimisticDepositBox,"RequestWithdrawal") .withArgs(depositor.address,amountToWithdraw,requestTimestamp);

```

Price Proposals

The code snippets above represent the core functionality for deploying a minimal contract and requesting a price. The next step in the contract lifecycle after a request has been made is a price can be proposed.

Here is theproposePriceFor function from the Optimistic Oracle contract that is used to propose a price for requests. The price proposal will revert if the parameters do not match an existing price request. For reference, thedeveloper quickstart repo demonstrates how to call theproposePriceFor method.

Liveness

An important aspect to consider when using the Optimistic Oracle is the liveness period which is the number of seconds a proposal must wait before a price can be resolved and contracts can be settled. AdefaultLiveness value is set (currently 7,200) or developers can set a customLiveness value.

```

Copy request.expirationTime=getCurrentTime().add(request.customLiveness!=0?request.customLiveness:defaultLiveness);

```

To set acustomLiveness value with our OptimisticDepositBox example above, we could add line 5 to change the liveness period from 7,200 seconds to 3,600:

```

Copy function_requestOraclePrice(uint256requestedTime)internal{ OptimisticOracleInterface oracle=_getOptimisticOracle(); // For other use cases, you may need ancillary data or a reward. Here, they are both zero.

oracle.requestPrice(priceIdentifier,requestedTime,"",IERC20(collateralCurrency),0);
oracle.setCustomLiveness(priceIdentifier,requestedTime,"",3600); }

```

Proposal Bond

A proposal bond is a preloaded reward to incentivize price proposals. Similar to the liveness period, a custom bond amount can be set for a price request.

Two important aspects to remember when setting a proposal bond is:

- When making a price request, the caller must approve the contract to transfer the value of the proposer reward for the collateral being used.
- The value corresponds to the collateral decimal value. So setting the reward to 1 for USDC that uses 6 decimals would be 1000000 while an 18 decimal token would be 1.
-

The below shows an example of setting the proposer bond to 50 USDC (6 decimals):

```

Copy function_requestOraclePrice(uint256requestedTime)internal{ OptimisticOracleInterface oracle=_getOptimisticOracle();
// For other use cases, you may need ancillary data. Here, the ancillary data is set to zero.
oracle.requestPrice(priceIdentifier,requestedTime,"",IERC20(collateralCurrency),0);
oracle.setBond(priceIdentifier,requestedTime,"",50000000); }

```

Settlement

The OptimisticDepositBox contract uses theexecuteWithdrawal method to settle contracts. It first checks if a price has been resolved by calling thehasPrice method on the Optimistic Oracle which reverts if the request is not settled or settleable. If a price has been resolved,settleAndGetPrice is called.

```

Copy function_getOraclePrice(uint256withdrawalRequestTimestamp)internalreturns(uint256) { OptimisticOracleInterface oracle=_getOptimisticOracle(); require(oracle.hasPrice(address(this),priceIdentifier,withdrawalRequestTimestamp,""), "Unresolved oracle price"); int256oraclePrice=oracle.settleAndGetPrice(priceIdentifier,withdrawalRequestTimestamp,"");

// For simplicity we don't want to deal with negative prices. if(oraclePrice<0) { oraclePrice=0; } returnuint256(oraclePrice); }

```

Previous OO Quick Start Next Internal Optimistic Oracle Last updated1 month ago On this page Was this helpful?Edit on GitHub
```