Sandboxed Oracle Environment

Only required for testing the dispute flow Whenever the assertion is disputed on a production network this triggers DVM request to be resolved by UMA voters. This might be impractical to fully simulate this in a testing environment, hence, developers can usemocked oracle contract to resolve requests as they had been returned by DVM.

Interacting with Optimistic Oracle requires having whitelisted price identifier and bonding currency. In production networks these are approved by UMA token holders as part of governance voting. When testing, it might be easier for developers to use their own set of UMA ecosystem contracts where they have full control over such whitelisting process.

Follow the written tutorial below or check out the video tutorial onYoutube .

Deploying sandboxed Oracle environment

A set of sandboxed UMA ecosystem contracts can be deployed using forge script fromOptimistic Oracle V3 developers quick-start repository .

After cloning the repository make sure you have the latest Foundry version by runningfoundryup command (see more installation details onFoundry book). From the root of the repository install dependencies with:

```

Copy forgeinstall

```

All script parameters are controlled through environment variables that should be exported before running the script:

- ETH_RPC_URL
- : URL of the RPC node to use for broadcasting deployment transactions on the desired test network.
- MNEMONIC
- : Mnemonic of the account to use for deployment (derived account with 0 index will be used by default).
- ETHERSCAN_API_KEY
- : API key for Etherscan, used for verifying deployed contracts.
- DEFAULT_IDENTIFIER
- : Default identifier used by Optimistic Oracle V3 when resolving disputes. If not provided, this defaults toASSERT_TRUTH
- identifier. The script will also whitelist this identifier.
- DEFAULT_LIVENESS
- : Default liveness in seconds used by Optimistic Oracle V3 when settling assertions. If not provided, this defaults to7200
- seconds.
- DEFAULT_CURRENCY
- : Default currency used by Optimistic Oracle V3 when bonding assertions and disputes. The script will also whitelist this currency. If not provided, the script would also deploy amintable ERC20 token
- and use it as the default currency based on following parameters:
  - DEFAULT_CURRENCY_NAME
  - : Name of the new token. If not provided, this defaults toDefault Bond Token
  - .
  - DEFAULT_CURRENCY_SYMBOL
  - : Symbol of the new token. If not provided, this defaults toDBT
  - .
  - DEFAULT_CURRENCY_DECIMALS
  - : Number of decimals of the new token. If not provided, this defaults to18
  - .
- *
- MINIMUM_BOND
- : Minimum bond amount in Wei of default currency required by Optimistic Oracle V3 when accepting new assertions. If not provided, this defaults to100e18
- Wei.

-

Once all required environment variables are setup, deploy and verify the set of sandboxed UMA ecosystem contracts with:

```

Copy forgescript\ --broadcast \ --fork-url ETH_RPC_URL \ --mnemonics"MNEMONIC"\ --sender(castwalletaddress--mnemonic"MNEMONIC")\ --verify \ script/OracleSandbox.s.sol

```

At the top of the script output the addresses of the deployed contracts should be logged:

- Finder
- : Used to locate other UMA ecosystem contract addresses. Deployment script will have added newStore
- ,AddressWhitelist
- ,IdentifierWhitelist
- ,MockOracleAncillary
- andOptimisticOracleV3
- implementations.
- Store
- : Stores final fees of bonding currencies. The script will have set the final fee of default currency to half of provided minimum bond amount since Optimistic Oracle V3 calculates minimum bond twice the value of this final fee.
- AddressWhitelist
- : Stores approved currency addresses that can be used for bonding assertions and disputes. The script will have already whitelisted the default currency.
- IdentifierWhitelist
- : Stores approved identifiers that can be used to resolve disputes at DVM. The script will have already whitelisted the default identifier.
- MockOracleAncillary
- : Optimistic Oracle will use this contract to resolve disputes equivalent to DVM voting on mainnet. Once requested, anyone can push desired test price through this mock oracle contract.
- TestnetERC20
- : Unless existing default currency was provided the script will have deployed this new token that is whitelisted as a bonding currency. Anyone can mint new tokens by calling itsallocateTo(address ownerAddress, uint256 value)
- method.
- OptimisticOracleV3
- : Optimistic Oracle that resolves assertions made by integrating contracts.
-

Resolving dispute requests

Whenever the assertion is disputed, Optimistic Oracle V3 callsrequestPrice method on the Oracle that triggers DVM vote on mainnet. In the sandboxed environment this would resolve toMockOracleAncillary contract instead.

The disputed request is uniquely identified by its identifier, timestamp and ancillary data that should be referenced by the testing developer pushing resolved price to the mocked oracle. TheMockOracleAncillary contract will emitPriceRequestAdded event with following parameters:

- address indexed requester
- : Address of the calling contract (Optimistic Oracle V3 in the sandboxed setup).
- bytes32 indexed identifier
- : Identifier referencing the rules on how the request should be resolved. By default Optimistic Oracle V3 usesASSERT_TRUTH
- identifier unless the integrating contract provided other value.
- uint256 time
- : Optimistic Oracle V3 will set this to timestamp when the original assertion was made.
- bytes ancillaryData
- : Optimistic Oracle V3 will put theassertionId
- and address of the asserter referencing the disputed assertion.
- bytes32 indexed requestId
- : This is hashed value of identifier, time and ancillary data provided by theMockOracleAncillary
- contract just as convenient method to resolve request by one parameter instead of three separate parameters.
-

Note that production voting contract on mainnet emits this differently and abovePriceRequestAdded is provided only as a convenience for developers when interacting with theMockOracleAncillary contract in their testing environment.

MockOracleAncillary contract provides two alternative methods for testing developers to resolve request:

- pushPrice(bytes32 identifier, uint256 time, bytes memory ancillaryData, int256 price)
- where request is referenced by its individual parameters.
- pushPriceByRequestId(bytes32 requestId, int256 price)
- where request is referenced by the hashed value of its identifying parameters.
-

While testing developer can pass anyint256 price value, Optimistic Oracle V3 can only interpret value of1e18 to represent truthful assertion. Any other value would settle assertion as being false. Also note thatMockOracleAncillary contract will only accept the first pushed price to resolve a given request.

Example dispute flow

As an illustration to the section above this example walks through resolving disputed assertion withcast tool from Foundry. This also requiresjq being installed for parsing returned values.

Just for the sake of simplicity setMINIMUM_BOND environment variable to0 when runningforge script command as discussed in thesandbox deployment section . This would allow to skip minting and approving assertion/dispute bonds. Take a note of deployed contracts log and export Optimistic Oracle V3 address toOOV3_ADDRESS andMockOracleAncillary address toMOCK_ORACLE_ADDRESS environment variables.

Also to keep this simple perform all steps from the same address:

```

Copy exportUSER_ADDRESS=(castwalletaddress--mnemonic"MNEMONIC")

```

Start with generating test assertion and grab the resultingassertionId emitted by Optimistic Oracle V3 (corresponds to topic index1 in theAssertionMade event):

```

Copy exportASSERTION_TX=(castsend--mnemonic"MNEMONIC"--jsonOOV3_ADDRESS "assertTruthWithDefaults(bytes,address)" (cast--from-utf8"test claim") USER_ADDRESS|jq-r'.transactionHash') exportASSERTION_ID=(castreceipt--jsonASSERTION_TX|jq-r--argOOV3_LOWERCASE(echoOOV3_ADDRESS|tr[:upper:] [:lower:]) '.logs[] | select(.address==OOV3_LOWERCASE).topics[1]')

```

Dispute the above assertion and grab the resultingrequestId emitted byMockOracleAncillary (corresponds to topic index3 in thePriceRequestAdded event):

```

Copy exportDISPUTE_TX=(castsend--mnemonic"MNEMONIC"--jsonOOV3_ADDRESS "disputeAssertion(bytes32,address)" ASSERTION_ID USER_ADDRESS|jq-r'.transactionHash') exportREQUEST_ID=(castreceipt--jsonDISPUTE_TX|jq-r--argMOCK_LOWERCASE(echoMOCK_ORACLE_ADDRESS|tr[:upper:] [:lower:]) '.logs[] | select(.address==MOCK_LOWERCASE).topics[3]')

```

Now resolve the disputed assertion as truthful by pushing1e18 as price toMockOracleAncillary contract:

```

Copy castsend--mnemonic"MNEMONIC"MOCK_ORACLE_ADDRESS"pushPriceByRequestId(bytes32,int256)"REQUEST_ID(cast--to-wei1)

```

This allows us to settle the assertion at Optimistic Oracle V3:

```

Copy castsend--mnemonic"MNEMONIC"OOV3_ADDRESS"settleAssertion(bytes32)"ASSERTION_ID

```

Finally, you can verify that Optimistic Oracle V3 has correctly settled the assertion (this should returntrue):

```
```

Copy castcallOOV3_ADDRESS"getAssertionResult(bytes32)(bool)"ASSERTION_ID

```

Further sandbox configuration

The deployment script documented in thesection above covers most common use cases and should be sufficient to test example contracts from theOptimistic Oracle V3 integration quickstart repository . Though, when developing your own integrating contracts it might be required to further configure the sandboxed oracle environment.

Whitelisting of bonding currencies

In case the integrating contract requires handling assertion bonds in multiple currencies all of them should be approved at theAddressWhitelist contract. On production this should go through UMA governance while in the sandboxed environment the deployer owns this contract and can calladdToWhitelist method directly. Assuming the address ofAddressWhitelist contract is exported toADDRESS_WHITELIST and required bonding currency is exported toBONDING_CURRENCY environment variables, this can be done by:

```

Copy castsend--mnemonic"MNEMONIC"ADDRESS_WHITELIST"addToWhitelist(address)"BONDING_CURRENCY

```

In order to test the scenario on how the integrating contracts would behave if UMA governance removed support for a particular bonding currency, you can test this by callingremoveFromWhitelist method on theAddressWhitelist contract:

```

Copy castsend--mnemonic"MNEMONIC"ADDRESS_WHITELIST"removeFromWhitelist(address)"BONDING_CURRENCY

```

Note that Optimistic Oracle V3 is caching whitelisted bonding currencies. Hence, before testing impact on the integrating contracts somebody should sync the contract cache (anyone has authority to do this):

```

Copy castsend--mnemonic"MNEMONIC"OOV3_ADDRESS"syncUmaParams(bytes32,address)"(cast--format-bytes32-string"")BONDING_CURRENCY

```

Changing the final fee

In case the UMA governance was to change the configured final fee for a particular bonding currency this would also change the minimum assertion bond required by Optimistic Oracle V3. In order to test such impact on integrating contracts in the testing environment, the deployer would need to callsetFinalFee on theStore contract (exported asSTORE_ADDRESS ). E.g. to set final fee to1000e18 , you would call:

```

Copy castsend--mnemonic"MNEMONIC"STORE_ADDRESS"setFinalFee(address,(uint256))"BONDING_CURRENCY"((cast--to-wei1000))"

```

Again, before testing impact on integrating contracts someone would need to sync new final fee at the Optimistic Oracle V3:

```

Copy castsend--mnemonic"MNEMONIC"OOV3_ADDRESS"syncUmaParams(bytes32,address)"(cast--format-bytes32-string"")BONDING_CURRENCY

```

Whitelisting of identifiers

If the integrating contract plans to make assertions with multiple identifiers, the developer would need to add them toIdentifierWhitelist contract (exported asIDENTIFIER_ADDRESS ) in the sandboxed environment, e.g. addingCUSTOM_IDENTIFIER :
```

```

Copy castsend--mnemonic"MNEMONIC"IDENTIFIER_ADDRESS"addSupportedIdentifier(bytes32)"(cast--format-bytes32-string"CUSTOM_IDENTIFIER")

```

In order to test the scenario on how the integrating contracts would behave if UMA governance removed support for a particular identifier, you can test this by callingremoveSupportedIdentifier method on theIdentifierWhitelist contract:

```

Copy castsend--mnemonic"MNEMONIC"IDENTIFIER_ADDRESS"removeSupportedIdentifier(bytes32)"(cast--format-bytes32-string"CUSTOM_IDENTIFIER")

```

Note that Optimistic Oracle V3 is caching whitelisted identifiers. Hence, before testing impact on the integrating contracts somebody should sync the contract cache (anyone has authority to do this):

```

Copy castsend--mnemonic"MNEMONIC"OOV3_ADDRESS"syncUmaParams(bytes32,address)"(cast--format-bytes32-string"CUSTOM_IDENTIFIER")0x0000000000000000000000000000000000000000

```

Previous Escalation Managers Next oSnap Last updated1 month ago On this page

Was this helpful? Edit on GitHub