

# Data Feeds API Reference

When you use data feeds, retrieve the feeds through the `AggregatorV3Interface` and the proxy address. Optionally, you can call variables and functions in the `AccessControlledOffchainAggregator` contract to get information about the aggregator behind the proxy.

## AggregatorV3Interface

Import this interface to your contract and use it to run functions in the proxy contract. Create the interface object by pointing to the proxy address. For example, on Sepolia you could create the interface object in the constructor of your contract using the following example:

```
/* * Network: Sepolia * Data Feed: ETH/USD * Address: 0x694AA1769357215DE4FAC081bf1f309aDC325306 /constructor()
(priceFeed=AggregatorV3Interface(0x694AA1769357215DE4FAC081bf1f309aDC325306));
```

 To see examples for how to use this interface, read the [Using Data Feeds](#) guide.

You can see the code for the [AggregatorV3Interface](#) contract on GitHub.

### Functions in AggregatorV3Interface

Name	Description	<a href="#">decimals</a>	The number of decimals in the response	<a href="#">description</a>	The description of the aggregator that the proxy points to	<a href="#">getRoundData</a>	Get data from a specific round.
		<a href="#">latestRoundData</a>	Get data from the latest round.	<a href="#">version</a>	The version representing the type of aggregator the proxy points to.		

#### decimals

Get the number of decimals present in the response value.

`function decimals() external view returns (uint8);` \* RETURN: The number of decimals.

#### description

Get the description of the underlying aggregator that the proxy points to.

`function description() external view returns (string memory);` \* RETURN: The description of the underlying aggregator.

#### getRoundData

Get data about a specific round, using the `roundId`.

`function getRoundData(uint80 roundId) external view returns (uint80 roundId, int256 answer, uint256 startedAt, uint256 updatedAt, uint80 answeredInRound);` Parameters:

- `_roundId`: The round ID

Return values:

- `roundId`: The round ID
- `answer`: The data that this specific feed provides. Depending on the feed you selected, this answer provides asset prices, reserves, NFT floor prices, and other types of data.
- `startedAt`: Timestamp of when the round started.
- `updatedAt`: Timestamp of when the round was updated.
- `answeredInRound`: Deprecated - Previously used when answers could take multiple rounds to be computed

#### latestRoundData

Get the data from the latest round.

`function latestRoundData() external view returns (uint80 roundId, int256 answer, uint256 startedAt, uint256 updatedAt, uint80 answeredInRound)` Return values:

- `roundId`: The round ID.
- `answer`: The data that this specific feed provides. Depending on the feed you selected, this answer provides asset prices, reserves, NFT floor prices, and other types of data.
- `startedAt`: Timestamp of when the round started.
- `updatedAt`: Timestamp of when the round was updated.
- `answeredInRound`: Deprecated - Previously used when answers could take multiple rounds to be computed

#### version

The version representing the type of aggregator the proxy points to.

`function version() external view returns (uint256)` \* RETURN: The version number.

## AccessControlledOffchainAggregator

This is the contract for the aggregator. You can call functions on the aggregator directly, but it is a best practice to use the [AggregatorV3Interface](#) to run functions on the proxy instead so that changes to the aggregator do not affect your application. Read the aggregator contract only if you need functions that are not available in the proxy.

The aggregator contract has several variables and functions that might be useful for your application. Although aggregator contracts are similar for each data feed, some aggregators have different variables. Use the `typeAndVersion()` function on the aggregator to identify what type of aggregator it is and what version it is running.

Always check the contract source code and configuration to understand how specific data feeds operate. For example, the [aggregator contract for BTC/USD on Arbitrum](#) is different from the aggregators on other networks.

For examples of the contracts that are typically used in aggregator deployments, see the [EBOCR repository](#) on GitHub.

### Variables and functions in AccessControlledOffchainAggregator

This contract imports `OffchainAggregator` and `SimpleReadAccessController`, which also include their own imports. The variables and functions lists include the publicly accessible items from these imported contracts.

A simple way to read the variables or functions is to get the ABI from a blockchain explorer and point the ABI to the aggregator address. To do this in Remix, follow the [Using the ABI with AtAddress](#) guide in the Remix documentation. As an example, you can find the ABI for the BTC/USD aggregator by viewing the [contract code in Etherscan](#).

Variables:

Name	Description	<a href="#">LINK</a>	The address for the LINK token contract on a specific network.	<a href="#">billingAccessController</a>	The address for the billing <code>AccessController</code> , which limits access to the <a href="#">billing configuration</a> for the aggregator.
		<a href="#">checkEnabled</a>	A boolean that indicates if access is limited to addresses on the internal access list.	<a href="#">maxAnswer</a>	This value is no longer used on most Data Feeds. Evaluate if your use case for Data Feeds requires a custom circuit breaker and implement it to meet the needs of your application. See the <a href="#">Risk Mitigation</a> page for more information.
		<a href="#">minAnswer</a>	This value is no longer used on most Data Feeds. Evaluate if your use case for Data Feeds requires a custom circuit breaker and implement it to meet the needs of your application. See the <a href="#">Risk Mitigation</a> page for more information.	<a href="#">owner</a>	The address that owns this aggregator contract. This controls which address can execute specific functions. Functions:

Name	Description	<a href="#">decimals</a>	Return the number of digits of precision for the stored answer. Answers are stored in fixed-point format.	<a href="#">description</a>	Return a description for this data feed. This is different depending on which feed you select.
		<a href="#">getAnswer</a>	Deprecated - Do not use this function	<a href="#">getBilling</a>	Retrieve the current billing configuration
		<a href="#">getRoundData</a>	Get the full information for the most recent round including the answer and update timestamps	<a href="#">getRoundData</a>	Get the full information for a specific aggregator round including the answer and update timestamps
		<a href="#">getTimestamp</a>	Deprecated - Do not use this function	<a href="#">hasAccess</a>	Check if an address has internal access
		<a href="#">latestAnswer</a>	Deprecated - Do not use this function	<a href="#">latestConfigDetails</a>	Return information about the current offchain reporting protocol configuration
		<a href="#">latestRoundData</a>	Get the full information for the most recent round including the answer and update timestamps	<a href="#">latestTimestamp</a>	Deprecated - Do not use this function
		<a href="#">latestTransmissionDetails</a>	Get information about the most recent answer	<a href="#">linkAvailableForPayment</a>	Get the amount of LINK on this contract that is available to make payments to oracles. This value can be negative if there are outstanding payment obligations.
		<a href="#">oracleObservationCount</a>	Returns the number of observations that oracle is due to be reimbursed for	<a href="#">owedPayment</a>	Returns how much LINK an oracle is owed for its observations.
		<a href="#">requesterAccessController</a>	Returns the address for the access controller contract	<a href="#">transmitters</a>	The oracle addresses that can report answers to this aggregator.
		<a href="#">typeAndVersion</a>	Returns the aggregator type and version. Many aggregators are <code>AccessControlledOffchainAggregator 3.0.0</code> , but there are other variants in production. The version is for the type of aggregator, and different from the contract version.	<a href="#">validatorConfig</a>	Returns the address and the gas limit for the validator contract
		<a href="#">version</a>	Returns the contract version. This is different from <code>typeAndVersion</code> for the aggregator.		

#### decimals

Return the number of digits of precision for the stored answer. Answers are stored in fixed-point format.

```
function decimals() external view returns (uint8 decimalPlaces);
```

#### [description](#)

Return a description for this data feed. Usually this is an asset pair for a price feed.

```
function description() public view override checkAccess returns (string memory) { return super.description(); }
```

#### [getAnswer](#)

This function is deprecated. Do not use this function.

#### [getBilling](#)

Retrieve the current billing configuration.

```
function getBilling() external view returns (uint32 maximumGasPrice, uint32 reasonableGasPrice, uint32 microLinkPerEth, uint32 linkGweiPerObservation, uint32 linkGweiPerTransmission)
{ Billing memory billing = s_billing; return (billing.maximumGasPrice, billing.reasonableGasPrice, billing.microLinkPerEth, billing.linkGweiPerObservation, billing.linkGweiPerTransmission); }
```

#### [getRoundData](#)

Get the full information for a specific aggregator round including the answer and update timestamps. Use this to get the full historical data for a round.

```
function getRoundData(uint80 roundId) public view override checkAccess returns (uint80 roundId, int256 answer, uint256 startedAt, uint256 updatedAt, uint80 answeredInRound)
{ return super.getRoundData(_roundId); }
```

#### [getTimestamp](#)

This function is deprecated. Do not use this function.

#### [hasAccess](#)

Check if an address has internal access.

```
function hasAccess(address user, bytes memory calldata) public view virtual override returns (bool) { return super.hasAccess(_user, calldata) || _user == tx.origin; }
```

#### [latestAnswer](#)

This function is deprecated. Do not use this function.

#### [latestConfigDetails](#)

Return information about the current offchain reporting protocol configuration.

```
function latestConfigDetails() external view returns (uint32 configCount, uint32 blockNumber, bytes16 configDigest) { return (s_configCount, s_latestConfigBlockNumber, s_hotVars.latestConfigDigest); }
```

#### [latestRound](#)

This function is deprecated. Do not use this function.

#### [latestRoundData](#)

Get the full information for the most recent round including the answer and update timestamps.

```
function latestRoundData() public view override checkAccess returns (uint80 roundId, int256 answer, uint256 startedAt, uint256 updatedAt, uint80 answeredInRound) { return super.latestRoundData(); }
```

#### [latestTimestamp](#)

This function is deprecated. Do not use this function.

#### [latestTransmissionDetails](#)

Get information about the most recent answer.

```
function latestTransmissionDetails() external view returns (bytes16 configDigest, uint32 epoch, uint8 round, int192 latestAnswer, uint64 latestTimestamp) { require(msg.sender == tx.origin, "Only callable by EOA"); return (s_hotVars.latestConfigDigest, uint32(s_hotVars.latestEpochAndRound >> 8), uint8(s_hotVars.latestEpochAndRound), s_transmissions[s_hotVars.latestAggregatorRoundId].answer, s_transmis
```

#### [linkAvailableForPayment](#)

Get the amount of LINK on this contract that is available to make payments to oracles. This value can be negative if there are outstanding payment obligations.

```
function linkAvailableForPayment() external view returns (int256 availableBalance) { // there are at most one billion LINK, so this cast is safe int256 balance = int256(LINK.balanceOf(address(this))); // according to the argument in the definition of totalLINKDue, // totalLINKDue is never greater than 2**172, so this cast is safe int256 due = int256(totalLINKDue()); // safe from overflow according to above sizes return int256(balance) - int256(due); }
```

#### [oracleObservationCount](#)

Returns the number of observations that oracle is due to be reimbursed for.

```
function oracleObservationCount(address signerOrTransmitter) external view returns (uint16) { Oracle memory oracle = s_oracles[_signerOrTransmitter]; if (oracle.role == Role.Unset) { return 0; } return s_oracleObservationsCounts[oracle.index] - 1; }
```

#### [owedPayment](#)

Returns how much LINK an oracle is owed for its observations.

```
function owedPayment(address transmitter) public view returns (uint256) { Oracle memory oracle = s_oracles[_transmitter]; if (oracle.role == Role.Unset) { return 0; } Billing memory billing = s_billing; uint256 linkWeiAmount = uint256(s_oracleObservationsCounts[oracle.index] - 1) * uint256(billing.linkGweiPerObservation) * 1 gwei; linkWeiAmount += s_gasReimbursementsLinkWei[oracle.index] - 1; return linkWeiAmount; }
```

#### [requesterAccessController](#)

Returns the address for the access controller contract.

```
function requesterAccessController() external view returns (AccessControllerInterface) { return s_requesterAccessController; }
```

#### [transmitters](#)

The oracle addresses that can report answers to this aggregator.

```
function transmitters() external view returns (address[] memory) { return s_transmitters; }
```

#### [typeAndVersion](#)

Returns the aggregator type and version. Many aggregators are `AccessControlledOffchainAggregator 2.0.0`, but there are other variants in production. The version is for the type of aggregator, and different from the contract version.

```
function typeAndVersion() external pure virtual override returns (string memory) { return "AccessControlledOffchainAggregator 2.0.0"; }
```

#### [validatorConfig](#)

Returns the address and the gas limit for the validator contract.

```
function validatorConfig() external view returns (AggregatorValidatorInterface validator, uint32 gasLimit) { ValidatorConfig memory vc = s_validatorConfig; return (vc.validator, vc.gasLimit); }
```

#### [version](#)

Returns the contract version. This is different from the typeAndVersion for the aggregator.

```
function version() external view returns (uint256);
```