Smart Contract

Overview

All Attestation contracts must inherit theAttestation base contract.

The base contract is ERC721 compliant, which defines the data structure of an Attestation report, and function methods that allow adding and revoking an Attestation.

Once an attestor has successfully submitted an Attestation report, the Attestation contract mints an ERC721 NFT to the attestor's wallet.

Automata 2.0 has currently built various attestation contracts for the various attestation types, namely:

- SingleStepOptimisticAttestation.sol
- MultiStepOptimisticAttestation.sol
- ConsensusBasedAttestation.sol
-

Full details about the various attestation types can be found in theAttestation section.

Attestation

The Attestation contract itself is considered to be abstract, it is not intended to be deployed as an independent contract. Instead, it is served as a base contract with built-in methods and virtual methods that need to be implemented in the derived contract.

TheAttestation Report object structures are defined here.

SingleStepOptimisticAttestation

Introduction of the SingleStepOptimisticAttestation.

If you're looking to build upon theSingleStepOptimisticAttestation , there are 3 pivotal methods awaiting your implementation:

1. _needVerify
2.
    - To decide whether the attestation should be verified instantly.
3.

```

Copy function_needVerify()internalpurereturns(bool)

```

1. _verify
2.
    - To validate the correctness and authenticity of the submitted attestation.
3.

```

Copy function_verify(bytescalldatadata) internalviewreturns(boolverified,bytesmemorydataHash)

```

1. _generateSVG
2.
    - To produce a visual representation (SVG image) for the corresponding NFT.
3.

```

Copy function_generateSVG(uint256tokenId)internalviewreturns(stringmemory)

```

MultiStepOptimisticAttestation

Introduction of the MultiStepOptimisticAttestation.

If you're looking to build upon theMultiStepOptimisticAttestation , there are 5 pivotal methods awaiting your implementation:

1. _startChallenge
2.
    - To handle the logic of starting a challenge.
3.

```

Copy function_startChallenge(bytes32hash)internalvirtual

```

1. _handleEvidence
2.
    - The challenger and the defender post their evidence on-chain.
3.

```

Copy function_handleEvidence(bytes32hash,bytesmemorydata)internalvirtual

```

1. _resolveChallenge
2.
    - To judge the winner of this challenge.
3.

```

Copy function_resolveChallenge(bytes32hash)internalvirtualreturns(boolchallengeSuccess)

```

1. _cancelChallenge
2.
    - To handle the logic of cancelling a challenge.
3.

```

Copy function_cancelChallenge(bytes32hash)internalvirtual

```

1. _generateSVG
2.
    - To produce a visual representation (SVG image) for the corresponding NFT.
3.

```

Copy function_generateSVG(uint256tokenId)internalviewreturns(stringmemory)

```

ConsensusBasedAttestation

Introduction of the ConsensusBasedAttestation.

If you're looking to build upon theConsensusBasedAttestation , there are 3 pivotal methods awaiting your implementation:

1. _verifyAndUpdateChallenge
2.
    - To verify whether the signature is valid and from your off-chain consensus party.
3.

```

Copy function_verifyAndUpdateSignature( bytes32hash, boolagree, bytescalldatasignatures )internalvirtualreturns(boolvalid)
```

```

1. _judgeAttestationStatus
2. 
    - To judge the attestation status according to your consensus mechanism.
3. 

```

Copy function_judgeAttestataionStatus(bytes32hash) internalvirtualreturns(AttestationStatusstatus)

```

1. _generateSVG
2. 
    - To produce a visual representation (SVG image) for the corresponding NFT.
3. 

```

Copy function_generateSVG(uint256tokenId)internalviewreturns(stringmemory)

```

Previous Attestor Next Machine Attestation Last updated6 months ago On this page *Overview * Attestation * SingleStepOptimisticAttestation * MultiStepOptimisticAttestation * ConsensusBasedAttestation

Was this helpful?
```