NFT (ERC-721)

Suggest Edits

The NFT template is an audited, ready-to-deploy smart contract for creating and managing NFTs. It implements the ERC-721 standard, which is widely used for representing non-fungible tokens (NFTs) on a blockchain. Unlike ERC-20 tokens, which represent fungible and interchangeable assets, ERC-721 NFTs are unique and non-interchangeable, making them suitable for digital collectibles, gaming assets, and many other use cases.

The ERC-721 NFT standard has gained significant popularity and has been implemented by numerous projects and platforms. Some key use cases for ERC-721 NFTs include:

- Digital Collectibles:
- ERC-721 NFTs are extensively used for creating and trading unique digital collectibles. These collectibles can represent various items such as artwork, trading cards, virtual pets, in-game assets, and more.
- Tokenized Assets:
- ERC-721 NFTs can represent ownership in real-world assets such as real estate, artwork, jewelry, and other physical assets. This enables fractional ownership, providing liquidity and opening up investment opportunities.
- Gaming Assets:
- ERC-721 NFTs are a perfect fit for representing in-game assets, enabling players to own, trade, and transfer virtual items securely and transparently. This functionality has facilitated the emergence of blockchain-based gaming ecosystems.

In this comprehensive guide, you explore the NFT template, which provides all the necessary information to deploy and understand the contract's common functions.

Deployment Parameters

The NFT template creates a customized, fully compliant ERC-721 smart contract. To create a contract using this template, provide the following parameter values when deploying a smart contract template using thePOST: /templates/{id}/deploy API.

Template ID: 76b83278-50e2-4006-8b63-5b1a2a814533

Template Deployment Parameters

Parameter Type Required Description name String X Name of the contract - stored as a property of the contract on-chain. symbol String

Symbol of the token - stored onchain. The symbol is usually 3 or 4 characters in length. defaultAdmin String X The address of the default admin. This address can execute permissioned functions on the contract. You will lose administrative access to the contract if this is not set to an address you control. primarySaleRecipient String X The recipient address for first-time sales. platformFeeRecipient String

The recipient address for all sale fees. You can set this to your address if you are deploying a template on someone else's behalf. platformFeePercent Float

The percentage of sales that go to the platform fee recipient. For example, set it as 0.1 if you want 10% of sales fees to go toplatformFeeRecipient . royaltyRecipient String X The recipient address for all royalties (secondary sales). This allows the contract creator to benefit from further sales of the contract token. royaltyPercent Float X The percentage of secondary sales that go to the royalty recipient. For example, set it as 0.05 if you want royalties to be 5% of secondary sales value. contractUri String

The URL for the marketplace metadata of your contract. This is used on marketplaces like OpenSea. See Contract-level Metadata for more information. trustedForwarders String[]

A list of addresses that can forward ERC2771 meta-transactions to this contract. See ethereum.org for more information. Here is an example of the templateParameters JSON object within the request body todeploy a contract from a template for the ERC-721 NFT template.

In this example, the defaultAdmin , primarySaleRecipient , and royaltyRecipient parameters are the same address but can be set distinctly based on your use case. JSON ... "templateParameters": { "name": "My NFT Contract", "defaultAdmin": "0x4F77E56dfA40990349e1078e97AC3Eb479e0dAc6", "primarySaleRecipient": "0x4F77E56dfA40990349e1078e97AC3Eb479e0dAc6", "royaltyRecipient": "0x4F77E56dfA40990349e1078e97AC3Eb479e0dAc6", "royaltyPercent": 0.05 }

Common Functions

This section lists the most commonly used functions on NFT template, their respective parameters and potential failure scenarios. These functions include:

- approve [write]
- mintTo [write]
- safeTransferFrom [write]
- setTokenURI [write]
- ownerOf [read]
- balanceOf [read]

At this time, failure scenarios and error messages received from the blockchain are not passed through Circle's APIs. Instead, you will receive a genericESTIMATION_ERROR error.

approve [write]

The approve function allows the owner of an ERC721 NFT to approve another address to transfer the token on their behalf.

Parameters:

Parameter Type Description to address The address approved to transfer the token. tokenId unit256 The identifier of the token being approved for transfer. Failure Scenarios:

- If theto
- address matches the current owner of the token (owner), the function will fail. This check ensures that the approval is not granted to the same owner, preventing unnecessary approvals.
- "ERC721: approval to current owner"
- The function requires that the caller
- _msgSender
- either be the token's owner or have been approved for all by the owner. If this condition is not met, the function will fail. This validation prevents unauthorized users from approving transfers on behalf of the token owner.
- "ERC721: approve caller is not token owner or approved for all"

Solidity function approve(address to, uint256 tokenId) public virtual override { address owner = ERC721Upgradeable.ownerOf(tokenId); require(to != owner, "ERC721: approval to current owner");

require(_msgSender() == owner || isApprovedForAll(owner, _msgSender()), "ERC721: approve caller is not token owner or approved for all");

_approve(to, tokenId); }

mintTo [write]

The mintTo function is a function that mints a new NFT and assigns it to a specific address. This function can only be called by an address with the MINTER_ROLE .

Parameters:

Parameter Type Description to address The address to which the minted NFT will be assigned. uri string The URI (Uniform Resource Identifier) of the newly minted NFT. Returns:

Parameter tokenIdToMint uint256 The unique identifier of the minted NFT. Failure Case:

- If the caller of the function does not have the
- MINTER_ROLE
- assigned, the function will fail and throw an exception.
- "AccessControl: account ", StringsUpgradeable.toHexString(account), " is missing role ", StringsUpgradeable.toHexString(uint256(role), 32)
- The function checks if the length of the _uri string is greater than 0, ensuring that the URI is not empty.
- "empty uri."
- The function checks that the to address is not the zero address.
- "ERC721: mint to the zero address"
- The function checks that the
- tokenId
- has not already been created.
- "ERC721: token already minted"

Solidity function mintTo(address _to, string calldata _uri) external onlyRole(MINTER_ROLE) returns (uint256) { // _mintTo is re-used. mintTo just adds a minter role check. return _mintTo(_to, _uri); }

function _mintTo(address _to, string calldata _uri) internal returns (uint256 tokenIdToMint) { tokenIdToMint = nextTokenIdToMint; nextTokenIdToMint += 1;

require(bytes(_uri).length > 0, "empty uri."); _setTokenURI(tokenIdToMint, _uri);

_safeMint(_to, tokenIdToMint);

emit TokensMinted(_to, tokenIdToMint, _uri); }

safeTransferFrom [write]

This function allows the transfer of an ERC721 NFT from the from address to the to address. It requires that the caller is the token owner or has been approved to transfer the token.

Parameters:

Parameter Type Description from address The address that currently owns the token and wants to transfer it. to address The address that will receive ownership of the token. tokenId unit256 The unique identifier of the token being transferred. Failure Scenarios:

- The
- isApprovedOrOwner
- function is called to check if the caller is the token owner or an approved address. This check ensures that the transfer can only be performed by the token owner or an approved address.
- "ERC721: caller is not token owner or approved"
- If the to address is a contract, the
- checkOnERC721Received
- function is called to check if the to address is a contract that implements the
- onERC721Received
- function correctly - according to the ERC721 standard.
- It checks if the token being transferred
- tokenId
- parameter is owned by the
- from
- address parameter.
- "ERC721: transfer from incorrect owner"
- It checks that the
- to
- address parameter is not the zero address. If it is the zero address, the function throws an exception with the message.
- "ERC721: transfer to the zero address"
- If the transfer is restricted on the contract, it still allows burning and minting. It checks whether the
- TRANSFER_ROLE
- is assigned to either the
- from
- or
- to
- address. This ensures that token transfers comply with specific access control restrictions defined by the contract.
- "restricted to TRANSFER_ROLE holders”
- The function will check if the
- to
- address is a contract. If it is, the
- _checkOnERC721Received
- hook will check if the receiver address properly handles the received token. I
- "ERC721: transfer to non ERC721Receiver implementer"

Solidity // safeTransferFrom function safeTransferFrom(address from, address to, uint256 tokenId) public virtual override { safeTransferFrom(from, to, tokenId, ""); }

// safeTransferFrom - with data parameter function safeTransferFrom(address from, address to, uint256 tokenId, bytes memory data) public virtual override { require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: caller is not token owner or approved"); _safeTransfer(from, to, tokenId, data); }

setTokenURI [write]

This function is responsible for setting the metadata URI for a specific NFT token.

Parameters:

Parameter Type Description tokenId unit256 The unique identifier of the NFT token for which the metadata URI is being set. uri string The new metadata URI that will be associated with the NFT token. Failure Scenarios:

- The function checks if the caller is authorized to set the metadata URI. It calls the
- canSetMetadata
- function, which checks the authorization based on certain conditions specified in the contract.
- "NFTMetadata: not authorized to set metadata."
- The function verifies if the metadata URI is not frozen. It checks the
- uriFrozen
- boolean flag to determine if the metadata is in a frozen state. If the metadata is frozen, meaning it cannot be changed, the function will throw an exception.
- "NFTMetadata: metadata is frozen."
- If the provided URI is empty, the function will throw an exception with the message
- "NFTMetadata: empty metadata."

Solidity function setTokenURI(uint256 _tokenId, string memory _uri) public virtual { require(_canSetMetadata(), "NFTMetadata: not authorized to set metadata."); require(!uriFrozen, "NFTMetadata: metadata is frozen."); _setTokenURI(_tokenId, _uri); }

function _setTokenURI(uint256 _tokenId, string memory _uri) internal virtual { require(bytes(_uri).length > 0, "NFTMetadata: empty metadata."); _tokenURI[_tokenId] = _uri;

emit MetadataUpdate(_tokenId); }

ownerOf [read]

This function is used to retrieve the address of the owner of the ERC721 NFT with the specified tokenId .

Parameters:

Parameter Type Description tokenId unit256 The unique identifier of the token for which the owner's address is being fetched. Failure Scenarios

- The function checks if the owner's address is not the zero address. This check is performed to ensure that a valid owner address is returned.
- "ERC721: invalid token ID"

Note:

- The function does not revert if the token doesn't exist. The zero address will be returned.

Solidity function ownerOf(uint256 tokenId) public view virtual override returns (address) { address owner = _ownerOf(tokenId); require(owner != address(0), "ERC721: invalid token ID"); return owner; }

balanceOf [read]

This function retrieves the balance (number of tokens) owned by a specific owner address.

Parameter:

Parameter owner address The address for which the token balance is being fetched. Failure Scenarios:

- The function checks if the
- owner
- address is not the zero address. The zero address represents an invalid or nonexistent address.
- "ERC721: address zero is not a valid owner"

Solidity function balanceOf(address owner) public view virtual override returns (uint256) { require(owner != address(0), "ERC721: address zero is not a valid owner"); return _balances[owner]; } Updated22 days ago * Table of Contents * * Deployment Parameters * * Common Functions * * * approve [write] * * * mintTo [write] * * * safeTransferFrom [write] * * * setTokenURI [write] * * * ownerOf [read] * * * balanceOf [read]