Multi-Token (ERC-1155)

Suggest Edits

The Multi-Token template is an audited, ready-to-deploy smart contract for the ERC-1155 multi-token standard. ERC-1155 is a versatile token standard that allows for creating and managing multiple types of tokens within a single smart contract. Unlike other token standards, like ERC-20 and ERC-721, ERC-1155 supports fungible and non-fungible tokens, providing flexibility for various use cases.

The ERC-1155 standard enables the creation of tokens representing different types of assets, such as digital collectibles, in-game items, unique artwork, and more, all within the same contract. This reduces the need to deploy separate contracts for different token types, improving efficiency and reducing costs.

Some use cases of the standard include:

- Gaming Assets:
- With ERC-1155, developers can create game assets that can be fungible or non-fungible. For example, fungible ERC-1155 tokens can represent in-game currencies, while non-fungible ERC-1155 tokens can represent unique weapons, characters, or virtual land.
- Digital Collectibles:
- Similar to ERC-721, ERC-1155 can be used to create and trade digital collectibles. However, ERC-1155 offers additional flexibility, allowing for the creation of fungible and non-fungible tokens under the same contract. This enables the creation of collections with varying levels of scarcity and uniqueness.
- Tokenized Real-World Assets:
- ERC-1155 tokens can also represent ownership of real-world assets such as real estate or shares in a company. By combining fungible and non-fungible tokens, ERC-1155 offers a more efficient solution for fractional ownership of assets.
- Batch Operations:
- One of the significant advantages of ERC-1155 is the ability to perform batch operations. Developers can transfer multiple tokens in a single transaction, making it more cost-efficient and reducing gas fees.

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

Deployment Parameters

The Multi-Token template creates a smart contract representing and controlling any number of token types. These tokens can of the ERC-20, ERC-721 or any other standard. 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: aea21da6-0aa2-4971-9a1a-5098842b1248

Template Deployment Parameters

Parameter Type Required Description name String X Name of the contract - stored 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. primarySaleRecipient String X The recipient address for first-time sales. platformFeeRecipient String

The recipient address for all sale fees. 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 to platformFeeRecipient. 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. contractUri String

The URL for the marketplace metadata of your contract. trustedForwarders String[]

A list of addresses that can forward ERC2771 meta-transactions to this contract. Here is an example of the templateParameters JSON object within the request body todeploy a contract from a template for the ERC-1155 Multi-Token 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 Multi-Token Contract", "defaultAdmin": "0x4F77E56dfA40990349e1078e97AC3Eb479e0dAc6", "primarySaleRecipient": "0x4F77E56dfA40990349e1078e97AC3Eb479e0dAc6", "royaltyRecipient": "0x4F77E56dfA40990349e1078e97AC3Eb479e0dAc6", "royaltyPercent": 0.05 }

Common Functions

This section lists the most commonly used functions on the Multi-Token template, along with their respective parameters and potential failure scenarios. These functions include:

- mintTo [write]
- safeTransferFrom [write]
- setApprovalForAll [write]
- setTokenURI [write]
- burn [write]
- safeBatchTransferFrom [write]
- balanceOfBatch [read]
- balanceOf [read]
- nextTokenIdToMint [read]
- uri [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.

mintTo [write]

The mintTo function allows you to create new NFTs or increase the supply of existing NFTs. It is a flexible function that can cater to both scenarios.

Parameters:

Parameter Type Description _to address The address to which the newly minted NFT will be assigned. _tokenId unit256 The unique identifier for the NFT. If the value is set to type(uint256).max , the function will assign the next available token ID. Otherwise, it will assign the provided _tokenId value. _uri calldata The Uniform Resource Identifier (URI) for the NFT's metadata. It specifies the location from where the metadata can be retrieved. _amount unit256 The amount of the newly minted NFTs to be assigned. Failure Scenarios:

- Insufficient Role:
- The
- mintTo
- function is defined with the
- onlyRole(MINTER_ROLE)
- modifier, meaning only addresses with the
- MINTER_ROLE
- can call this function. The function will revert and fail if the caller does not have the necessary role.
- Token ID Overflow:
- If the
- _tokenId
- parameter is set to
- type(uint256).max
- (the maximum value for a uint256), the function will attempt to create a new token and assign the next available token ID. However, an overflow can occur if the
- nextTokenIdToMint
- variable has reached its maximum value. This overflow condition will cause the function to fail.
- Invalid Token ID:
- If the
- _tokenId
- parameter is not set to
- type(uint256).max
- , the function will attempt to mint an NFT with the specified token ID. However, if the provided
- _tokenId
- value is greater than or equal to the value of
- nextTokenIdToMint
- , the function will revert and fail with the following error message.
- "invalid id"
- Existing Token ID with Non-Empty URI:

- When
- _tokenId
- is provided and already exists, the function checks whether the associated metadata URI for that token ID is empty. If the URI is not empty, the token has already been minted and has an associated URI. In this case, the function will fail and revert, preventing the same token ID from being minted multiple times.
- Minting to Zero Address:
- The function checks whether the
- _to
- address is the zero address
- address(0)
- . Minting tokens to the zero address is prohibited, as it represents an invalid or non-existent address. If
- _to
- is the zero address, the function will fail and revert with the following error message.
- "ERC1155: mint to the zero address"
- Rejection by ERC1155Receiver Contract:
- If the recipient address
- _to
- is a contract, the function will attempt to call the
- onERC1155Received
- function of that contract to check if the contract supports receiving the NFT. If the contract's
- onERC1155Received
- function rejects the transfer by returning a value other than
- IERC1155ReceiverUpgradeable.onERC1155Received.selector
- , the function will revert and fail with the following error message.
- "ERC1155: ERC1155Receiver rejected tokens"

Notes:

- Creating New NFTs:
- When you pass
- type(uint256).max
- via the
- _tokenId
- parameter, the function will create a new NFT with
- _tokenId
- equal to
- nextTokenIdToMint
- and assign it to the specified
- _to
- address. The
- _uri
- parameter allows you to provide the metadata URI for the newly created NFT. The
- amount
- parameter allows you to specify how many instances of this NFT with the given ID should be minted.
- Increasing Supply of Existing NFTs:
- If you pass an existing token ID via the
- tokenId
- parameter, the function will increase the supply of that specific NFT. Instead of creating a new token ID, the function will mint additional instances of the existing NFT, adding to the current supply. Again, the
- _amount
- parameter determines how many additional instances of the NFT should be minted.

Solidity // Lets an account with MINTER_ROLE mint an NFT. function mintTo(address _to, uint256 _tokenId, string calldata _uri, uint256 _amount) external onlyRole(MINTER_ROLE) { uint256 tokenIdToMint; if (_tokenId == type(uint256).max) { tokenIdToMint = nextTokenIdToMint; nextTokenIdToMint += 1; } else { require(_tokenId < nextTokenIdToMint, "invalid id"); tokenIdToMint = _tokenId; }

// _mintTo is re-used. mintTo just adds a minter role check. _mintTo(_to, _uri, tokenIdToMint, _amount); }

safeTransferFrom [write]

The safeTransferFrom function allows for transferring a specified amount of a particular token ID from one address from to another address to .

Parameters:

Parameter Type Description from address The address of the current token owner, from whom the tokens will be transferred. to address The address of the recipient who will receive the transferred tokens. id uint256 The unique identifier for transferring the token. amount uint256 The amount of tokens being transferred. This represents the number of tokens to

be transferred. data bytes Optional additional data to pass to the receiver contract if it is a contract. This can include custom arguments or instructions for the receiving contract. Failure Scenarios:

- Transfer to Zero Address:
- The function verifies if the to address is the zero address
- address(0)
- . Transfers to the zero address are not permitted, as it represents an invalid or non-existent address. If the
- to
- address is the zero address, the function fails and reverts with the following error message.
- "ERC1155: transfer to the zero address"
- Insufficient Balance:
- The function checks if the from address has a sufficient balance of the specified token ID (id) to perform the transfer. If the balance exceeds the specified amount, the function fails and reverts to the following error message.
- "ERC1155: insufficient balance for transfer"
- Caller Not Authorized:
- The function verifies if the caller of the
- _msgSender()
- function is either the owner of the tokens (
- from
-) or has been approved as an operator for
- from
- . If the caller is neither the token owner nor an approved operator, the function fails and reverts with the following error message.
- "ERC1155: caller is not token owner or approved"
- Before/After Token Transfer Hooks:
- The function calls the
- _beforeTokenTransfer
- and
- _afterTokenTransfer
- hooks to update any necessary state or perform additional checks. These hooks may contain custom business logic that can cause the transfer to fail if certain conditions are not met.
- ERC1155Receiver Contract Rejection:
- If the
- to
- address is a contract, the function attempts to call the
- onERC1155Received
- function of that contract to check if the contract supports receiving the tokens. If the contract's
- onERC1155Received
- function rejects the transfer by returning a value other than
- IERC1155ReceiverUpgradeable.onERC1155Received.selector
- , the function fails and reverts with the following error message.
- "ERC1155: ERC1155Receiver rejected tokens"

Solidity // See IERC1155-safeTransferFrom. function safeTransferFrom(address from, address to, uint256 id, uint256 amount, bytes memory data) public virtual override { require(from == _msgSender() || isApprovedForAll(from, _msgSender()), "ERC1155: caller is not token owner or approved"); _safeTransferFrom(from, to, id, amount, data); }

setApprovalForAll [write]

The setApprovalForAll function is used to set the approval status for an operator to manage all tokens of the caller (owner) on their behalf.

Parameters:

Parameter Type Description operator address The operator's address for whom the approval status is set. The operator will be able to manage all tokens owned by the caller. approved bool The boolean value indicates whether the operator is approved (true) or disapproved (false) to manage all tokens on behalf of the caller. Failure Scenarios:

- The function requires that the caller (owner) cannot set the approval status for themselves. If the operator address provided is the same as the owner address, the function will fail with the given following error message.
- "ERC1155: setting approval status for self"

Notes:

- Once an operator is approved using the
- setApprovalForAll
- function, they can act on behalf of the token owner. This includes performing actions such as transferring tokens.

Solidity // See {IERC1155-setApprovalForAll}. function setApprovalForAll(address operator, bool approved) public virtual override { _setApprovalForAll(_msgSender(), operator, approved); }

// Approve operator to operate on all ofowner tokens // Emits an {ApprovalForAll} event. function _setApprovalForAll(address owner, address operator, bool approved) internal virtual { require(owner != operator, "ERC1155: setting approval status for self"); _operatorApprovals[owner][operator] = approved; emit ApprovalForAll(owner, operator, approved); }

setTokenURI [write]

The setTokenURI function is used to set the metadata URI for a given NFT.

Parameters:

Parameter Type Description tokenId unit256 The unique identifier of the NFT for which the metadata URI needs to be set. uri string The URI string represents the metadata's location associated with the NFT. Failure Scenarios:

- The function requires the metadata URI to have a length greater than zero. This error occurs when the input parameter
- _uri
- is an empty string.
- "NFTMetadata: empty metadata"
- This error occurs if the
- _canSetMetadata()
- function returns false. It indicates that the caller has no authority or permission to set the metadata for the given NFT.
- "NFTMetadata: not authorized to set metadata"
- This error occurs when
- uriFrozen
- is true, indicating that the metadata is frozen and cannot be updated.
- "NFTMetadata: metadata is frozen"

Solidity // Sets the metadata URI for a given NFT. 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); }

// Sets the metadata URI for a given NFT. function _setTokenURI(uint256 _tokenId, string memory _uri) internal virtual { require(bytes(_uri).length > 0, "NFTMetadata: empty metadata."); _tokenURI[_tokenId] = _uri;

emit MetadataUpdate(_tokenId); }

burn [write]

This function allows a token owner to burn a specified amount (value) of tokens they own.

Parameters:

Parameter Type Description account address The address of the token owner who wants to burn their tokens. id unit256 The unique identifier of the token to be burned. value unit256 The amount of tokens to be burned. Failure Scenarios:

- This error occurs when the caller of the burn function is neither the owner of the tokens nor approved to burn them. The caller must either be the account that owns the tokens or have approval from the owner to burn the tokens.
- "ERC1155: caller is not owner nor approved"
- The function checks that the burning amount does not exceed the available balance. This error occurs if the amount of tokens specified to be burned (
- amount
-) is greater than the balance of tokens (
- fromBalance
-) owned by the specified account.
- "ERC1155: burn amount exceeds balance"
- This error occurs if the from address (the address from which the tokens are being burned) is the zero address (0x000...). This address is generally reserved as an invalid or non-existent address and cannot be used for token burning.
- "ERC1155: burn from the zero address"

Solidity // Lets a token owner burn the tokens they own (i.e. destroy for good) function burn(address account, uint256 id, uint256 value) public virtual { require(account == _msgSender() || isApprovedForAll(account, _msgSender()), "ERC1155: caller is not owner nor approved.");

_burn(account, id, value); }

safeBatchTransferFrom [write]

This function enables the safe transfer of multiple ERC1155 tokens from one address (from) to another address (to) in a batch.

Parameters :

Parameter Type Description from address The address from which the tokens are transferred. to address The address to which the tokens are transferred. ids uint256[] An array of unique identifiers of the tokens to be transferred. amounts uint256[] An array specifying the corresponding amounts of tokens to be transferred for each ID. data bytes Additional data to pass along with the transfer. Optional parameter. Failure Scenarios:

- This error occurs if the caller of the function is neither the token owner nor approved to perform the transfer. The caller must either be the from address or have approval from the from address to transfer the tokens.
- "ERC1155: caller is not token owner or approved"
- This error occurs if the lengths of the
- ids
- and amounts arrays do not match. Each ID should have a corresponding amount to be transferred. The arrays should have the same length.
- "ERC1155: ids and amounts length mismatch"
- This error occurs if the
- to
- address is the zero address (0x000...). Transferring tokens to the zero address is not allowed as it is generally used to represent an invalid or non-existent address.
- "ERC1155: transfer to the zero address"
- This error occurs if the from address does not have a sufficient balance of tokens to transfer. The function checks that the from address has enough tokens of each ID to fulfill the transfer.
- "ERC1155: insufficient balance for transfer"
- This error occurs if the to address is a contract and the contract does not implement the
- onERC1155BatchReceived
- function from the
- IERC1155ReceiverUpgradeable
- interface or if the function returns a value other than
- onERC1155BatchReceived.selector
- . This check ensures that the receiving contract can handle the transferred tokens properly.
- "ERC1155: ERC1155Receiver rejected tokens"

Solidity // IERC1155-safeBatchTransferFrom function safeBatchTransferFrom(address from, address to, uint256[] memory ids, uint256[] memory amounts, bytes memory data) public virtual override { require(from == _msgSender() || isApprovedForAll(from, _msgSender()), "ERC1155: caller is not token owner or approved"); _safeBatchTransferFrom(from, to, ids, amounts, data); }

balanceOfBatch [read]

The balanceOfBatch function retrieves the balances of multiple accounts for multiple token IDs in a single function call.

Parameters:

Parameter Type Description accounts address[] An array of addresses representing the accounts to query the balances for. ids unit256[] An array of unique identifiers of the tokens to query the balances for. Failure Scenarios:

- Mismatched Array Lengths:
- The function requires that the length of the accounts array is equal to the length of the
- ids
- array. If this condition is not met, it will throw a required exception with the following error message.
- "ERC1155: accounts and ids length mismatch"

Solidity // IERC1155-balanceOfBatch // Requirements: // accounts and ids must have the same length. function balanceOfBatch(address[] memory accounts, uint256[] memory ids) public view virtual override returns (uint256[] memory) { require(accounts.length == ids.length, "ERC1155: accounts and ids length mismatch");

uint256[] memory batchBalances = new uint256;

for (uint256 i = 0; i < accounts.length; ++i) { batchBalances[i] = balanceOf(accounts[i], ids[i]); }

return batchBalances; }

balanceOf [read]

The balanceOf function retrieves the balance of a specific account for a particular token ID.

Parameters:

Parameter Type Description account address The EVM address for which the balance is being queried. id unit256 The unique token identifier for which the balance is being queried. Failure Scenarios:

- Zero Address:
- The function requires that the account parameter is not set to the zero address
- address(0)
- . If this condition is not met, it will throw a required exception with the following error message.
- "ERC1155: address zero is not a valid owner".

Solidity // See IERC1155-balanceOf // Requirements: // account cannot be the zero address. function balanceOf(address account, uint256 id) public view virtual override returns (uint256) { require(account != address(0), "ERC1155: address zero is not a valid owner"); return _balances[id][account]; }

uri [read]

The URI function retrieves the associated URI with a specific token ID. This URI provides a way to access metadata and additional information about the token.

Parameters:

Parameter Type Description tokenId unit256 This is the unique token identifier for retrieving the URI. Solidity // Returns the URI for a tokenId function uri(uint256 _tokenId) public view override returns (string memory) { return _tokenURI[_tokenId]; }

Public Variables

Public variables are accessible from within the contract and can be accessed from external contracts. Solidity automatically generates a getter function for public state variables.

nextTokenIdToMint [read]

The nextTokenIdToMint variable is a public constant on the smart contract. An unsigned integer (uint256) represents the next token ID minted or created when type(uint256).max is passed to the mintTo function.

Solidity // The next token ID of the NFT to mint. uint256 public nextTokenIdToMint; Updated22 days ago *Table of Contents * * Deployment Parameters * * Common Functions * * * mintTo [write] * * * safeTransferFrom [write] * * * setApprovalForAll [write] * * * setTokenURI [write] * * * burn [write] * * * safeBatchTransferFrom [write] * * * balanceOfBatch [read] * * * balanceOf [read] * * * uri [read] * * Public Variables * * * nextTokenIdToMint [read]