Token (ERC-20)

Suggest Edits

The Token template is an audited, ready-to-deploy smart contract for an ERC-20 token. The ERC-20 standard is the most popular standard for fungible tokens. The token's fungibility allows it to be used for a variety of use cases, including:

- Stablecoins:
- ERC-20 tokens serve as the foundation for stablecoins like USDC, backed by US dollars. To learn more about the USDC contract, see0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48
- on Etherscan.
- Loyalty Points:
- ERC-20 tokens can represent on-chain loyalty points to incentivize and reward users for their activities within a platform or ecosystem.
- Governance:
- ERC-20 tokens can represent governance rights, allowing holders to participate in protocol decisions. Notable examples include tokens utilized by platforms like Uniswap.
- Ownership:
- ERC-20 tokens can also represent fractional ownership in real-world assets, such as houses, ounces of gold, or company shares.

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

Deployment Parameters

The Token template creates a customized, fully compliant ERC-20 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: a1b74add-23e0-4712-88d1-6b3009e85a86

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 on-chain. 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. Important: 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 sales fees. If you deploy a template on someone else's behalf, you can set this to your own address. 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 the platformFeeRecipient . contractUri String

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

A list of addresses that can forward ERC2771 meta-transactions to this contract. Seeethereum.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-20 Token template.

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

Common Functions

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

- approve [write]

- transfer [write]
- mintTo [write]
- burn [write]
- grantRole [write]
- revokeRole [write]
- balanceOf [read]
- allowance [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 lets token owners specify a limit on the number of tokens another account address can spend on their behalf.

Parameters:

Parameter Type Description spender address The owner approves the account address to spend tokens. It could be another smart contract address or an externally owned account address. amount unit256 The number of tokens the owner approves for spending by the specified spender. The amount should be set in the smallest denomination of the ERC-20 token. Failure Scenarios:

- The approve function fails if the
- spender
- is the zero address.
- "ERC20: approve to the zero address"

Notes:

- If
- amount
- is the maximum uint256 value the allowance is not updated when the
- transferFrom
- function is called. This is semantically equivalent to an infinite approval.
- There is no balance check on the
- approve
- function so token owners may approve allowances greater than their current balance.

Solidity function approve(address spender, uint256 amount) public virtual override returns (bool) { address owner = _msgSender(); _approve(owner, spender, amount); return true; }

transfer [write]

Allows the token owner to transfer a specified number of tokens to another account address.

Parameters:

Parameter Type Description to address The address to which the token will be transferred. This can be another user's address or a smart contract address. amount unit256 The number of tokens to transfer. Failure Scenarios:

- The
- to
- address parameter is checked to ensure it is not the zero address
- address(0)
- .
- "ERC20: transfer to the zero address"
- The
- beforeTokenTransfer
- hook is called, which checks
- TOKEN_TRANSFER
- permissions.
- "restricted to TRANSFER_ROLE holders”
- The token owner doesn’t have a sufficient balance to transfer the specified token amount.
- "ERC20: transfer amount exceeds balance"

Solidity function transfer(address to, uint256 amount) public virtual override returns (bool) { address owner = _msgSender();

_transfer(owner, to, amount); return true; }

mintTo [write]

A designated minter can create a specified number of tokens and assign them to a specified account address.

Parameters:

Parameter Type Description to address The address to which the newly minted tokens will be assigned. amount unit256 The number of tokens to be minted and assigned to the specified address. Failure Scenarios:

- The function checks if the caller has the
- MINTER_ROLE
- . The function will fail if the caller does not have the minter role. Expect to see the following error:
- "not minter."
- The function ensures that the account address is not the zero address
- address(0)
- , as this is an invalid address to mint tokens to.
- "ERC20: mint to the zero address"
- The
- beforeTokenTransfer
- hook is called, which checks
- TOKEN_TRANSFER
- permissions.
- "restricted to
- TRANSFER_ROLEholders”

Solidity function mintTo(address to, uint256 amount) public virtual { require(hasRole(MINTER_ROLE, _msgSender()), "not minter."); _mintTo(to, amount); }

burn [write]

Allows a token holder to burn (destroy) a specified number of the token total supply.

Parameters:

Parameter Type Description amount unit256 The number of tokens to be burned. Failure Scenarios:

- The
- _beforeTokenTransfer
- hook is called, which checks
- TOKEN_TRANSFER
- permissions.
- "restricted to TRANSFER_ROLE holders”
- The token owner has no sufficient balance to burn the specified token amount.
- "ERC20: burn amount exceeds balance"

Solidity function burn(uint256 amount) public virtual { _burn(_msgSender(), amount); } Permissions and Roles

Roles are referred to by their bytes32 identifier. For example:

Solidity bytes32 public constant MY_ROLE = keccak256("MY_ROLE"); Roles can also represent a set of permissions. To restrict access to a function call, use hasRole :

Solidity function foo() public { require(hasRole(MY_ROLE, msg.sender)); ... } Roles can be granted and revoked dynamically via the grantRole and revokeRole functions. Each role has an associated admin role, and only accounts that have a role's admin role can call grantRole and revokeRole . By default, the admin role for all roles is DEFAULT_ADMIN_ROLE . Only accounts with the DEFAULT_ADMIN_ROLE can grant or revoke other roles. You can use the _setRoleAdmin function to create more complex role relationships.

grantRole [write]

Grants a specified role to an account. Only an account address that has the admin role assigned can call this function.

Parameters

Parameter Type Description role bytes32 The bytes32 identifier of the role to be granted. account address The address to which the role will be granted. Failure Scenarios:

- The

- onlyRole(getRoleAdmin(role))
- modifier ensures that the function caller has the admin role for the specified role. Only addresses with the admin role can grant roles to other accounts.
- "AccessControl: account ", StringsUpgradeable.toHexString(account), " is missing role ", StringsUpgradeable.toHexString(uint256(role), 32)
- The function checks if the account already has the specified role using the
- hasRole
- function. If the account does not have the role, the function continues. There is no error message.

Solidity function grantRole(bytes32 role, address account) public virtual override onlyRole(getRoleAdmin(role)) { _grantRole(role, account); }

revokeRole [write]

Revoke the specified role from an account address. Only account addresses with the admin role assigned can call this function.

Parameters:

Parameter Type Description role bytes32 The role to be revoked. account address The address from which the role will be revoked. Failure Scenarios:

- The
- onlyRole(getRoleAdmin(role))
- modifier ensures that the function caller has the admin role for the specified role. Only addresses with the admin role can revoke roles from other accounts.
- "AccessControl: account ", StringsUpgradeable.toHexString(account), " is missing role ", StringsUpgradeable.toHexString(uint256(role), 32)

Solidity function revokeRole(bytes32 role, address account) public virtual override onlyRole(getRoleAdmin(role)) { _revokeRole(role, account); }

balanceOf [read]

Retrieves the balance of tokens owned by a specific account address.

Parameter:

Parameter Type Description account address The address for which the token balance is being fetched. Solidity function balanceOf(address account) public view virtual override returns (uint256) { return _balances[account]; }

allowance [read]

Returns the maximum amount the spender is approved to withdraw from the owner’s account. This function retrieves the allowance granted by the owner account address to the spender account address.

Parameters:

Parameter Type Description owner address The address that granted the allowance. spender address The address for which the allowance is being fetched. Solidity function allowance(address owner, address spender) public view virtual override returns (uint256) { return _allowances[owner][spender]; } Updated22 days ago * Table of Contents * * Deployment Parameters * * Common Functions * * * approve [write] * * * transfer [write] * * * mintTo [write] * * * burn [write] * * * grantRole [write] * * * revokeRole [write] * * * balanceOf [read] * * * allowance [read]