Wormfare Jetton Smart Contract

Project Overview

The project contains two smart contracts written in the FunC programming language: JettonMinter and JettonWallet. The contracts work together following the official TEP-74 Jettons standard. The initial source code for the contracts was taken from the official TON blockchain repository:

- JettonMinter: https://github.com/ton-blockchain/token-contract/blob/main/ft/jetton-minter-discoverable.fc
- JettonWallet: https://github.com/ton-blockchain/token-contract/blob/main/ft/jetton-wallet.fc

The JettonWallet contract did not get any changes, while the JettonMinter contract got the following changes:

- Max token supply was capped at 300,000,000 tokens;
- The ability to change token content (type and metadata URI) was removed.

The main purpose of the contracts is to mint the official Wormfare token - WOFR.

1. Functional Requirements

1.1 Roles

1. **Admin:** The administrator of the JettonMinter contract, who can mint the supply and pass its role to another wallet.
2. **Token Holder**: Any account holding Jetton tokens. Token holders can transfer or burn their tokens.

1.2 Features

The **JettonMinter** contract includes the following features:

- **TEP-74 Standard Jetton Master Contract Functions**: Provides standard TEP-74 master contract functionality, including **get_jetton_data()** and **get_wallet_address()** functions.
- **TEP-89 Standard Jetton Wallet Discovery Functions:** Provides operation handlers in accordance with TEP-89 standard.
- **Mint**: Allows the admin to mint up to 300M tokens to the provided address.
- **Burn**: Enables any token holder to burn (destroy) their own tokens, reducing the total supply.
- **Change admin**: The admin can pass its role to another address.

The **JettonWallet** contract includes the following features:

- **TEP-74 Standard Jetton Wallet Contract Functions**: Provides standard TEP-74 wallet contract functionality, including transferring and burning tokens.

2. Technical Requirements

This project is developed using FunC, with Blueprint as the development environment and TypeScript for testing and deployment scripting. Base contract implementations were taken from the official TON blockchain repository:

- JettonMinter: https://github.com/ton-blockchain/token-contract/blob/main/ft/jetton-minter-discoverable.fc
- JettonWallet: https://github.com/ton-blockchain/token-contract/blob/main/ft/jetton-wallet.fc

The JettonWallet contract did not get any changes, while the JettonMinter contract got the following changes:

- Max token supply was capped at 300,000,000 tokens;
- The ability to change token content (type and metadata URI) was removed.

In the project folder, the following structure is found:

```
├── contracts
│   ├── imports
│   │   └── stdlib.fc
│   └── jetton
│       ├── discovery-params.fc
│       ├── jetton-minter.fc
│       ├── jetton-utils.fc
│       ├── jetton-wallet.fc
│       ├── op-codes.fc
│       └── params.fc
├── docs
│   └── Wormfare Jetton Smart Contract.pdf
├── scripts
│   ├── deployJettonMinter.ts
├── tests
│   └── JettonMinter.spec.ts
├── wrappers
│   ├── Jetton.ts
│   ├── JettonConstants.ts
│   ├── JettonMinter.compile.ts
│   ├── JettonMinter.ts
│   ├── JettonWallet.compile.ts
│   ├── JettonWallet.ts
│   └── ui-utils.ts
├── README.md
├── jest.config.ts
├── package.json
├── tsconfig.json
└── yarn.lock
```

Start with **README.md** to find all the basic information about the project structure and scripts that are required to test and deploy the contracts.

Inside the **./contracts/jetton** folder, **jetton-minter.fc** and **jetton-wallet.fc** contain the smart contracts that this document describes.

In the **./tests** folder, **JettonMinter.spec.ts** provides the tests of the different methods of both contracts, in Typescript.

In the **./wrappers** folder, **JettonMinter.compile.ts** and **JettonWallet.compile.ts** files provide the build configuration for the contracts. **JettonMinter.ts** and **JettonWallet.ts** provide the interface for interacting with on-chain contracts.

The contract can be deployed using the **deployJettonMinter.ts** script in the **./scripts** folder. In order to do so, **.env.example** must be renamed to **.env.mainnet** (or **.env.testnet** for testnet deployment), and all required data must be provided.

Finally, this document can be found in **./docs**.

2.2. Contract Information

2.2.1. jetton-minter.fc

The **JettonMinter** contract provides basic TEP-74 Jetton master contract functionality.

A typical flow looks as follows:

Contract deployment

The contract is deployed, specifying the admin address and metadata URI.

Initial token mint

The admin mints the whole token supply to a specific address (owner).

Token distribution

The owner distributes the tokens to other wallets and smart contracts.

2.2.1.1. Assets

The **JettonMinter** contract stores the following data in its persistent storage:

- **total_supply:** The current token supply, that is capped at 300,000,000 tokens.
- **admin_address:** The address of the contract administrator.
- **content:** Holds the link to the off-chain metadata JSON file.
- **jetton_wallet_code:** The compiled code of the **JettonWallet** contract that is deployed for each user on the first token mint/transfer to their address.

2.2.1.2. Functions

The contract contains the following functions:

- **get_jetton_data:** Returns the total supply, admin address, content (metadata URI), and compiled **JettonWallet** contract code.
- **get_wallet_address:** Returns the address of the **JettonWallet** contract for the given address.

2.2.1.3. Operations

The contract handles the following operation codes:

- **21 (mint):** Mint the given supply of tokens to the given address. Can be called by the contract's admin only if the current total supply is zero.
- **0x7bdd97de (burn notification):** This operation is received from the **JettonWallet** contract upon burning tokens. The operation decreases the total supply value.
- **0x2c76b973 (provide wallet address):** This operation is a part of the TEP-89 Jetton Wallet Discovery standard.
- **3 (change admin):** Change the contract's admin address.

2.2.2. jetton-wallet.fc

The **JettonWallet** contract provides basic TEP-74 Jetton wallet contract functionality. The contract is being deployed for each new Jetton holder upon the first token transfer to their address.

2.2.2.1. Assets

The **JettonWallet** contract stores the following data in its persistent storage:

- **balance:** The total Jetton amount the current wallet holds.
- **owner_address:** The address of the token holder who owns this wallet.
- **jetton_master_address:** The address of the **JettonMinter** contract.
- **jetton_wallet_code:** The compiled code of the **JettonWallet** contract that is deployed for each user on the first token transfer to their address.

2.2.2.2. Functions

The contract contains the following functions:

- **get_wallet_data:** Returns the balance, owner address, **JettonMinter** contract address, and **JettonWallet** contract code.

2.2.2.3. Operations

The contract handles the following operation codes:

- **0xf8a7ea5 (transfer):** Transfer the given Jetton amount to the given address, deploying the **JettonWallet** contract for the receiver if needed.
- **0x178d4519 (internal transfer):** Handles an incoming Jetton transfer.
- **0x595f07bc (burn):** Burn the given amount of tokens.