How to create tokens on Tezos Platform?

leewayhertz.com/create-tokens-on-tezos

The global economy is inevitably growing towards a digital ecosystem. From investment to money transfer, organizations across the planet are embracing digitization with the adoption of digital tokens. In the quest for a more secure and private way to manage finances over the web, blockchain serves as a perfect solution. With roots in blockchain and data security, digital tokens are set to drive the securitization of assets in the most transformative ways.

At the heart of blockchains lay decentralized programs that aim at bringing consensus among thousands of nodes. Tezos is an innovative blockchain platform that amends its economic protocol through a voting mechanism and focuses on formal methods to improve safety. Apart from offering an original proof-of-stake consensus algorithm, it can also be used as a decentralized smart contract platform. Being considered one of the largest cryptos, Tezos reached a market capitalization value of about $25 billion in January 2021. The Tezos prediction stated by Digital Coin Price states that the coin would reach 5.13 dollars by the end of 2021.

This article will help you understand the necessary stages involved in the process of creating digital tokens on Tezos. In addition, it emphasizes answering the following significant questions related to the generation of digital tokens on Tezos.

- What is the FA 1.2 protocol?
- How to create digital tokens on Tezos?

What is the FA 1.2 protocol?

For any digital token, its functionality and wallet compatibility is defined by its standard. In Tezos, the token standards are written in TZIP (Tezos Interoperability Proposal) format. It led to the evolution of following different proposals, where each proposal is known by its FA number.

TZIP # 5: Financial Application 1 (FA 1) – Abstract Ledger.

TZIP #7: Financial Application 1.2 (FA 1.2) – Approvable Ledger.

TZIP# 12: Financial Application 2 (FA 2) – Multi-Asset Interface.

FA1.2 refers to an ERC-20 like fungible token standard (TZIP-7) for Tezos. At its core, FA1.2 comprises a ledger that maps identities to token balances, offers a standard API for token transfer operations and provides approval to external contracts or accounts for transferring a user's tokens. In addition, it supports the interactions with the smart contract. The FA1.2 protocol helps

- implement token transfer operations.
- provide approvals to spend tokens from other accounts.
- associate identities with balances.
- resistance to ERC-20 attack vector vulnerability.

How to create tokens on Tezos?

The development of tokens on the Tezos platform implements FA 1.2 protocol. Therefore, it is essential to have a concise idea of the contract interface that needs to be implemented. The contract implements the tezos standard known as tzip7 or FA 1.2. This standard represents the second version of fungible tokens.

The overall stages of the contract interface comprise five methods to generate the digital tokens on Tezos.

1. Transfer Method: It is used to send created tokens in contact from one address to another.

2. Get allowance Method: It helps to check the number of tokens available to you from another user.

3. Get Balance Method: It allows the contract to return the balance of address, which is passed to the function.

4. Get total supply Method: It helps to get or retrieve the total amount of tokens in return in this contract.

5. Approve Method: It helps to manage the balance by someone else by providing consent and specifying the number of allowable tokens to use.

Here are the crucial stages for creating the digital tokens on Tezos.

Stage 1: Contract creation and completion of the additional requirements (Preliminary steps)

This stage involves executing all the following preliminary steps like gathering requirements and settings before creating the contract.

Step 1: Create a contract with the necessary settings and initialize the truffle project.

Step 2: Add a constant with the name “no operations” and assign it the value of an empty list of operations.

This step is necessary because it will reduce writing an empty array and pass the constant directly in return function value. Thus, it will slightly simplify and reduce the amount of code in the contract.

Stage 2: Describing Interface and transfer of tokens in the contract

The initial method for implementing the contract interface is the transfer method. This method involves the transfer of tokens within the contract. So, the steps for transferring the tokens are as stated below.

Step 1: Declare the parameter type and describe the transfer parameters. Michaelson is a built-in legal language type used for smart contracts. So, the Michaelson pair is also used as a data storage structure. This pair has specific symbolic field names for effective implementation of the standard interface.

Step 2: Declare Michaelson pair and pass the address through the left value to transfer the funds.

Step 3: Declare Michaelson pair inside the right value and add “To” address via left value and specific “amount value, “representing the size to be transferred.

Step 4: Declare the “type” amount, where “type” is a natural number.

(Such structure allows to access the Michelson code quickly as possible, which is why it’s been adopted as the standard basis.)

It’s important to remember that all interface names and variable names should strictly match the names in the description of the FA 1.2 standards.)

Step 5: Move the unnecessary unit stub in actions and describe case ‘p’ from actions. Finally, we make a standard common transfer call from the transfer method and call it in case.

Step 6: Create a transfer function by describing the function, naming it and the interface function and then, transferring the constant parameters.

(It is essential to pass all the input parameters such as “from,” “to,” and the “value amount.”These values remain constant throughout overall function execution).

Step 7: We need to transfer our storage. So, we Pass the variable storage specifically because that’s what we’ll be changing in the code.

Step 8: Describe the function “block with.” This function does not generate any transaction. So, we can describe the return value by adding the no options constant value to the return value and then add to the storage.

Step 9: Call transfer in our main so that there will be no need to return it in the future.

Step 10: We pass all our parameters upon indexes which are stored in pairs. We make use of double indexes to access a pair inside a pair. Here, we use index 1 directly within the type pair inside and call the element upon the index.

Stage 3: Description of the transfer method

The description of the transfer method involves the following steps.

Step 1: It is essential to get an account to transfer the funds. So, declare the type “account variable” and assign it to type” nil.”

Step 2: To describe the type “account,” we need to describe the storage of the contract.

Step 3: Declare a new type, name it “account,” and assign a type “record” to it. This account contains two fields. The first field represents the balance of this type “amount,” while the second field indicates allowances or permission of this account for balance management.

Step 4: Assign the type “map” to the allowances field with the address key and “amount” as the second value. This amount value represents the number of tokens allowed to another address.

Step 5: Create an auxiliary function and name it ”get account.” On calling this function, we receive initialized type of structure. It implies that this function shows a record with initialized fields for further work if this account is not found in storage. If the account is located in storage, it will return instantly by specifying the parameters as “address” for finding the account address. and it will return a type “account.”

Step 6: Describe the function structure and then create an empty type account record. Next, create it as a variable, assign a type account to it and initialize the record. Next, we write our record keyword and initialize the fields of this record and assign the "0n" balance to it. Here, 'n' indicates a natural number.

Step 7: Initialize the allowances as an empty map. So we initialize our map by indicating a value which it should contain and return our account.

Step 8: To call upon the "get account method" to transfer, we pass the 'from' address and insert it.

Step 9: Expand the storage and describe its structure. The contract storage also contains two fields. The first field is total supply, i.e., the total amount of tokens produced in the contract for use. The second is a ledger, which is the repository of all accounts.

Step 10: Declare these fields through a big map structure where "address" is its key and "account" is the second parameter.

A big map is an entirely independent structure and is stored separately from the contract. So, when a contract contains a big map, it is created in the Tezos network and assigned with a specific 'int' value. Later, the contract calls the big map by this int value.

Step 11: Translate the optional value returned from the big map into "case." It is because the key may not be on the map and it will return an empty value.

Step 12: Pass the key in square brackets and describe what should be done with the value returned from the big map. It is possible to ignore this step if nothing is found, but on finding a specific instance account in the ledger, assign it to the account and close the case.

In this way, the helper function is created.

Stage 4: Update Balance and Get Destination Account

After describing the transfer method and creating the helper function, it is crucial to update the balance and acquire a destination account according to the following steps.

Step1: Check the balance with "if" so that it should be greater than the value to be sent to another user. If the balance is less, it will encounter an error and be represented as "Not enough balance'.

Step2: Check the allowances of this contract. If the account called by this method doesn't match the value of the 'from' address, we need to run a checkup, create a block as a scope, and close it as "else skip"

Step3: Create a spender allowance constant value by assigning "0n' as a stub to get the amount of tokens in the block for transfer

Step 4: Create another helper function, “get allowances,” and pass the account owner. This function will return the allowed amount for transfer by another account.

Step 5: Pass the spender address as the second parameter that wants to conduct this transaction and the third parameter will pass our storage and return the amount allowed.

Step 6: There is no need to return the block, remove it and call “case” from the owner account, allowances and transfer “spender.” If anything’s found, then we’ll return this exact value. If the map entry is empty, then we return zero and close our case.

Thus, we have completed the helper function.

Step 7: Now, let’s call it in the allowances check block. We call the get allowances function, transfer sender account as “tezos sender,” as the address tries to call this function for storage. The function fetches the available balance and checks whether the spender allowance balance is less than the value transferred in this transaction. It encounters an error if the allowance is less.

Step 8: Subtraction of transferrable value from allowance value. Hence, we update the “senderaccount.allowance” map from “tezos.center” and assign the spender allowance minus transferrable value. Make it into an abs function to get a natural number as

(senderAccount.allowances[Tezos.sender]:= abs(spenderAllowance – value);

Step 9: On completing all the necessary checks, we need to update the balances. It is performed by calling the sender balance record and subtracting the required value from it. Then, it gets translated into the abs function to get a value in the form of a natural number.

Hence, it is updated as sender account in ledger, passed from there and assigned the newly updated sender account as

s.ledger[from]:=senderAccount;

Step10: The eventual step is to get a destination account or the designation account. First, declare it as a designation account and call the already written get account helper function. Then, add value to the destination account balance, which gets transferred to its account.

Step 11: Update the destination account “to” account in the ledger and call for the compilation. If you get a compilation error, then change the name of the input parameters because from and to are the lego reserved names and continue writing our contract.

Stage 5: Implement GET Balance Function Method

This method plays a crucial role in implementing the get balance function by accepting specific parameters. The associated steps are given below.

Step 1: Discuss the parameters (params).

Step 2: Declare a new type of “balance params” with Michaelson pair.

The first record should be the address whose balance has to be returned. Let’s name it the “owner.”

The second record should be the address of the contract, which expects to accept the amount. The response will return to this address.

On implementing this function, declare the get balance function and pass all the required parameters. The first parameter is “owner,” the second parameter is “contract,” while the third parameter is “storage.” These parameters are constant values.

Step 3: Describe the structure of the gap balance function.

Step 4: Generate the owner account with the “getaccount” function call. Then, move to return value, transfer the list and describe the tezos transaction.

Step 5: Once the Tezos transaction is declared, make a call, pass amount (natural number as a parameter), call balance and pass zero amount as the second parameter and contract accordingly where is necessary to return.

Step 6: We call our function in main and copy the name. After receiving the transfer parameters, make the get balance call and then call the elements of the pair by indexes.

Step 7: At first, call upon the zero parameters and pass storage.

Step 8: As the size of the contract grows, it is necessary to split the contract into a few smaller files and merge them with “include.”

Step 9: Create a new folder and name it main.

Step 10: Execute truffle build to check compilation errors. Call the owner account instead of the owner’s address as an account type. Again, check for compilation.

Step 11: Once the build is successful, transfer the contract to main and add the already familiar field to truffle config (the contracts directory) and specify the path to the main folder from the place of the call through the slash.

Step 12: Save the file and carry types over to a separate file by creating file “types.lego” and the patrial’s folder. Next, move the “types” and add “include” of this file to the FA 1.2 legal file path. It is done by specifying the path to the patrial’s folder and adding the types.

Step 13: Check the build to know that the types are carried over to a separate file.

Stage 6: Carry the helper function

The steps mentioned below will guide to complete process of carrying the helper function.

Step 1: Carry the helper function by creating another file in the patrial's folder and naming it helpers.lego.

Step 2: Move "getaccount" and get allowance functions to this path.

Step 3: Recheck the build. Here, the project fails to assemble due to the absence of the get account function. So, we add this function to FA 1.2 and make the helper function "include" it.

Step 4: Make a new file and call it "transfer.lego", and move the transfer method to this path by copying the helper functions and type.

Step 5: On transferring lego, run another build to check if everything goes well. Hence, the contract is divided into separate files and the main contracts become more readable now.

Stage 7: Implementing getallowance function

Implementation of the getallowance function involves the following steps.

Step 1: Add this function to the contract interface and describe its parameters and other functions.

Step 2: Call these parameters as "allowance params" and describe them as a separate "type" and indicate the Michelson pair for it. It also contains another Michaelson pair inside it. It contains three values. It will contain a pair from two and a contract where you need to return the allowance value.

Step3: Rename the "getallowance" to "get allowance amt" function to all possible files wherever necessary to avoid logical compilation error because we have already created the getallowance helper function.

Step 4: Specify the type correctly, save the file and compile it to check that the function is renamed correctly at all places.

Step 5: Perform the following sub-steps.

- Describe the key value of a function.
- Get an allowance.
- Pass all our parameters like the owner address, spender address, a contract where the transaction gets executed with the value, and storage from where we'll take our allowance value.

Step 6: Create a constant owner account expression and assign a call of our getaccount method to it. We will pass "owner" and 's' and then retrieve the tezos transaction.

Step 7: Create a new variable with the name “spender allowance amt” and call the getallowance method.

Step 8: Call the tezos transaction, pass spender allowance amt, zero amount tezos to it, and the contract where it is necessary to return the transaction accordingly.

Step 9: Call the getallowance function in main, pass all our parameters by index. Then, describe the transactions and add them to the storage list. Thus, the compilation becomes successful on calling truffle build.

Stage 8: Implementing Get total supply Method

The steps for implementing Get total supply method are mentioned below.

Step 1: Describe this method into “get the total supply interface” and transfer custom parameters to it. For each method, we create custom parameters so that it is easy to expand. If you want to shift away from this implementation, you can quickly expand the input parameters.

Step 2: Describe the type “total supply,” and it will simply be the contract receiving ‘amt’ as a parameter.

Step 3: Describe the function “get total supply.” It will be accepting the contract where it’s necessary to return parameters and storage from where the data is fetched accordingly. Declare all constants as there is no need to change them.

Step 4: We retrieve, return and describe our function without block. As we can return the list of our tezos stop transaction to resp at once, we describe the returning pair. Add ‘s’ and ‘tezos.’ The tezos start transaction, pass the total supply from storage, pass the zero amount, and a contract where it is necessary to send this transaction.

Step 5: Call this function at the main point, take parameters from the case and then call” get total supply,” where we transfer the parameter entirely as it is single storage.

Step 6: Check for the compilation to ensure the successful implementation of this method.

Stage 9: Implementing Approve Method

It is an eventual stage that involves the implementation of the approval method according to the following steps.

Step 1: Describe approve method in the interface and create parameters for it.

Step 2: Describe the type “approved params” containing a Michaelson pair.

Step 3: Declare the Michaelson pair, which contains the spender address, the address to whom we want to give the right to dispose of our balance and the balance itself next to it.

Step 4: Describe approve function, pass two input parameters, “spender” and “address. Then, pass the value to the type “amount” and declare “storage” as the last parameter. This function keeps changing its storage and hence is declared as a variable. If the function changes storage, avoid executing the new transactions and everything will be happening inside the contract. Therefore we can at once write parameters that will return. These are empty “no operation” operations and the storage has a modified value.

Step 5: Get the sender account similar to other functions. But, first, call the get account function, pass the value of tezos.sender in the account which called upon this method initially and then passed the storage.

Step 6: Declare spender allowance to get the already allowed balance of this user.

Step 7: Call get allowances method, pass the sender account and spender address.

Step 8: Check that at least one of these values is zero. It means that the spender allowance amt should be zero.

Step 9: Add the amount allowed to a particular account. We can get concurrency and overwrite storage if both values are not equal to zero.

Since several transactions in one block are executed asynchronously. In this case, we can overwrite this storage with several transactions and get inconsistent data.

Step 10: Run sender account allowances spender and update it with the assigned value.

Step 11: Update the sender account in the ledger reassigning the already updated sender account to it.

Step 12: Call approve call params and pass “approve”, ” params 0″, “params 1” and “storage”.

Step 13: On running the compiler, the code runs successfully. Thus, the contract is written precisely, which corresponds to FA 1.2 token interface.

A smart contract in Michelson works in a specific way and any tampering attack with it will make the contract fail and keep the tokens secured. Having acquired a lot of traction in the FinTech industry, creating digital tokens on Tezos lead to a novel way of executing transaction processing. Moreover, with features native to Tezos, such as enhanced liquidity, accessibility at low cost, secure custody, and upgradeability of the Tezos protocol, Tezos remains an attractive choice for the generation of digital tokens.

LeewayHertz is one of the leading blockchain development companies that can help you create and launch digital tokens on various platforms. From consultation to creating digital tokens and building a blockchain-based product, our team can assist you through every step of generating tokens on Tezos. For more details, connect to our Tezos experts' team and get a free consultation.