# How to create a BEP-20 token on Binance Smart Chain?

leewayhertz.com/create-bep-20-token-on-binance-smart-chain

Every year, the Blockchain industry produces innovative and contemporary developments that challenge the prevailing system and bring upgrades, especially in the decentralized finance space. Binance Smart Chain is a recent development that benefits its users with a rich and growing digital asset ecosystem like never seen before in the decentralized exchange spectrum.

Using Binance Smart Chain, anyone can create and launch their digital token. Creating a BEP-20 token on Binance Smart Chain is quite a simple task. In this article, detailed information on how to create a BEP-20 token is given. Let's jump in and walk through how that works.

- What is Binance Smart Chain?
- What is the BEP-20 token standard?
- How to create the BEP-20 token?

## What is Binance Smart Chain?

It is a dual chain architecture that empowers its users to create their dApps and digital assets on the blockchain and provides the advantage of fast trading.

The main highlights of BSC are:

- EVM Compatible
- Cross-chain transfer
- Proof of Stake authority
- Block time ~3 seconds

BSC runs parallel to Binance Chain and enables Smart Contracts for tokens on the Binance blockchain. An all-new staking mechanism was also introduced for BNB, one of the world's top cryptocurrencies. BSC offers its users:

- cheap transaction fees
- cross chain defi mechanism that increases defi interoperability
- high-performance network holding the capability to produce a block every 3 seconds
- a growing ecosystem with millions of users
- a supportive Binance ecosystem that funds and bootstraps various defi projects

## What is the BEP-20 token standard?

BSC tokens are compliant with the BEP-20 standard, which is similar to the ERC-20 Ethereum standard. Since BEP-20 and ERC-20 are almost identical, it's compatible with both. Binance Smart Chain (BSC) is a fork of the Ethereum mainnet.

Token standards ensure basic functionalities for tokens such as returning a balance, transferring, viewing token ownership, etc. It is important to note that BSC tokens allow swapping for the regular Binance Chain that conforms to the BEP-2 standard. Transactions happening with these tokens on-chain requires a fee payment in BNB. It is like compensation for validators to secure the network.

Let's explore how to create these tokens.

## How to create a BEP-20 token?

The first step is to connect Metamask to Binance Smart Chain. Fill in all the details for your token, including ChainID, Network Name, New RPC URL, etc. Now, open remix.ethereum.org to write a BEP-20 token. Also, simultaneously go to OpenZeppelin-contracts in GitHub. Go to ERC20 in the token folder. Now, create a new file and name it bep-20 sol.

Now, begin writing the Smart Contract:

```
// SPDX-License-Identifier: UNLICENSED

pragma solidity 0.8.4;

/**

* @title SampleBEP20Token

* @dev Elementary BEP20 Token example, where all tokens are pre-assigned to the creator.

* Note they can later distribute these tokens as they wish using `transfer` and other

* `BEP20` functions.

* USE IT ONLY FOR LEARNING PURPOSES. SHOULD BE MODIFIED FOR PRODUCTION

*/

contract SampleBEP20Token {

string public name = “SampleBEP20 Token”;

string public symbol = “SBT”;

uint256 public totalSupply = 1000000000000000000000000; // 1 million tokens

uint8 public decimals = 18;

/**

* @dev Emitted when the ‘value’ tokens are shifted from one account to another.

* Note that `value` may also be zero.

*/

event Transfer(address indexed _from, address indexed _to, uint256 _value);

/**

* @dev Emitted when the allowance of a `spender` for an `owner` is set by

* a call to {approve}. `value` is the new allowance.

*/

event Approval(
```

```
address indexed _owner,
address indexed _spender,
uint256 _value
);
mapping(address => uint256) public balanceOf;
mapping(address => mapping(address => uint256)) public allowance;
/**
* @dev Constructor that gives msg.sender all of the existing tokens.
*/
constructor() {
balanceOf[msg.sender] = totalSupply;
}
/**
* @dev transfers `amount` tokens from the Caller’s account to the recipient’s.
* It returns a bool value that indicates the success of the operation.
* Emits a {Transfer} event.
*/
function transfer(address _to, uint256 _value)
public
returns (bool success)
{
require(balanceOf[msg.sender] >= _value);
balanceOf[msg.sender] -= _value;
balanceOf[_to] += _value;
emit Transfer(msg.sender, _to, _value);
```

```
return true;

}

/**

* @dev Sets `amount` as the allocation of `spender` over the Caller’s tokens.
```

* Returns a boolean value indicating that the operation has succeeded.

* NOTE: Beware that changing the allowance with this method will risk someone using both the new and the old funding by inappropriate transaction ordering. One possible solution to minimize such race conditions is to first decrease the spender’s allocation to 0 and set the desired value later:

Use: https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729

* It emits an {Approval} event.

*/

```
function approve(address _spender, uint256 _value)

public

returns (bool success)

{

allowance[msg.sender][_spender] = _value;

emit Approval(msg.sender, _spender, _value);

return true;

}

/**

* @dev Moves the `amount` tokens from `sender` to `recipient` using the allowance mechanism. `amount` is then deducted from the Caller’s allowance.
```

It will return a boolean value that indicates the success operation succeeded.

* Emits a {Transfer} event.

*/

```
function transferFrom(

address _from,

address _to,

uint256 _value

) public returns (bool success) {

require(_value <= balanceOf[_from]);

require(_value <= allowance[_from][msg.sender]);

balanceOf[_from] -= _value;

balanceOf[_to] += _value;

allowance[_from][msg.sender] -= _value;

emit Transfer(_from, _to, _value);

return true;

}

}
```

On the remix, either you can import the file using the Gist link or create a new file named SampleBEP20Token.sol and add the token contract code given above.

Before moving to deployment, it is necessary to compile the code. Press Ctrl+S, it'll compile the code. Make sure to choose the same compiler version in the remix, like the solidity version specified in the code.

After the successful compilation, move to deployment by clicking the icon below the compiler icon. Set the environment to Injected web3 since we are using Metamask to deploy. Ensure that the SampleBEP20Token contract is selected in the Contract dropdown. Now, we are all prepared to deploy the SBT token. Hit the Deploy button; a metamask popup will trigger for confirmation. Press Confirm to deploy the token.

After the transaction is mined, the contract details and the logs will appear under Deployed Contracts section. The SAMPLEBEP20TOKEN option under Deployed Contracts allows seeing the public methods and variables, which helps in testing deployment.

The Transfer method enables the transfer of tokens from one wallet to another. After entering the recipient's address and amount, press the Transfer button. Metamask popup will ask for confirmation; click on Confirm to push the transaction. After the successful transaction, its status and details will appear on Remix IDE logs.

By following the steps mentioned above, you can deploy as many tokens as you want.

Use the following steps to check the SBT balance of the recipient wallet on Metamask:

- Press the Add Token button at the bottom of the selected Metamask account.
- Fill in the token contract address. The token symbol and token name will automatically get detected. Click on Next.
- The token balance will appear when you click on Add Tokens to add the tokens to the Metamask account. As soon as the token is added, it is displayed in the Asset section of the metamask account.

The same steps are followed to add any deployed tokens on the chosen network; in our case, it was BSC Testnet.

## Conclusion

Binance Smart Chain efficiently fills the gap between various blockchain and dramatically extends the functionality of the original Binance Chain. BNB staking and EVM compatibility promise makes the platform an ideal engine for developers building robust decentralized applications.

If you're looking to build and implement BEP-20 tokens on Binance Smart Chain, we can help you with the correct methodology. Our Blockchain experts are well-versed and highly efficient in their approach. Get in touch with the leading Blockchain professionals and discuss your project requirements.