Quickstart: Transfer 10 USDC On-Chain

Use the USDC token contract and third-party tools to transfer 10 USDC from one blockchain wallet to another[Suggest Edits]

In this step-by-step guide you will write a script to send 10 USDC between two wallets using the Ethereum Sepolia testnet. You will use third-party tooling such as Javascript libraries, node services, and wallets.

After completing this tutorial, you will be able to integrate USDC payment flows within your app. Building with USDC enables you to move money near-instantly around the globe using decentralized protocols:

- All API calls and transactions in this guide take place within the testnet environment. No financial value funds will be transferred.
- You can adapt the sample code in this tutorial to serve as a framework for your development and create transactions that transfer financial value.

Overview of USDC and its token contract

USDC is a stablecoin backed 1:1 to the U.S dollar and operates natively on many public blockchains. Like many currencies, USDC is powered by a token contract, a programmable piece of code that manages user balances autonomously across a decentralized network. As transactions occur, the token contract automatically updates the digital ledger, ensuring real-time tracking of funds. This mechanism allows individuals and businesses to send and receive dollars seamlessly, capitalizing on the transparency, security, and efficiency of blockchain technology.

Before You Get Started

For this tutorial, you will need the following apps, tools, and/or accounts:

1. Circle Mint Sandbox Account
2. : Get an account that provides access to testnet USDC.
3. MetaMask
4. : A resource for creating wallets, which are devices for storing USDC on chain:
5.
    1. Create two wallets, one for sending and one for receiving testnet USDC transactions on the Ethereum “Sepolia” testnet network. These will serve as the origin wallet (the one you’re sending USDC from) and the destination wallet (the one you’re sending to). Note the addresses for both.
6.
    1. Note the private key for the origin wallet.
7.
    1. Import the Ethereum Sepolia testnetUSDC token contract
8.
    1. into your origin wallet to enable it to read the token balance. The contract’s address is
9.
    1. 0x1c7D4B196Cb0C7B01d743Fbc6116a902379C7238
10.
    1. .
11. You can access all of our testnet scriptshere
12. .
13. Infura
14. : A node service for connecting to the blockchain and submitting your USDC transaction:
15.
    1. Create an API key for the Web3 API offering.
16.
    1. Note the Ethereum “Sepolia” testnet endpoint provided.
17. Node.js
18. : Use the JavaScript runtime environment to run the code locally on your machine.
19. Web3.js
20. : JavaScript library containing functions to call the USDC token contract and initiate transactions
21. Visual Studio
22. : a lightweight but powerful source code editor available as a desktop app

1. Set up a Circle Mint Sandbox Account to access to testnet USDC

VisitCircle's signup page and click "Sign Up."

In your Sandbox account, click the Developer tab in the sidebar and generate an API key.

2. Obtain testnet USDC and native gas token

Here you will source a quantity of testnet USDC to use in this tutorial, as well as testnet ETH to use to pay the gas (transaction fee). Perform the steps shown using Terminal.

1. Configure using API key:1. Navigate tosample-sandbox.circle.com
2.
    1. Click on the settings wheel (in the top right-hand corner.
3.
    1. Insert the API key you got from the sandbox account.
4.
    1. Click on the hamburger button in the top left corner to access API endpoints.
5. Mint testnet USDC via simulated bank transfer fromCore Functionality
6. set:1. POST:https://api-sandbox.circle.com/v1/businessAccount/banks/wires
7.
    1.
        1. Click on the pre-fill button to add a simulated bank account used for testing.
8.
    1.
        1. Copy the ID value.
9.
    1. GET:https://api-sandbox.circle.com/v1/businessAccount/banks/wires/{id}/instructions
10.
    1.
        1. From the response, note the trackingRef and accountNumber.
11.
    1. POST:https://gpi-sandbox.circle.com/v1/mocks/payments/wire
12.
    1.
        1. Use the trackingRef and accountNumber values from the previous step.
13. Add recipient address fromPayouts API
14. set:1. POST:https://api-sandbox.circle.com/v1/addressBook/recipients
15.
    1.
        1. Provide your origin wallet address, set the blockchain to ETH, and input your email (as recipient), bns (the nickname for your destination wallet), and your user nickname.
16.
    1.
        1. Note the ID from the response.
17.
    1. GET:https://api-sandbox.circle.com/v1/addressBook/recipients/{id}
18.
    1.
        1. Verify that the address has been successfully added.
19. Perform payout in USDC and native gas token (ETH) fromPayouts API
20. set:1. POST:https://api-sandbox.circle.com/v1/payouts
21.
    1.
        1. Input the desired dollar amount and set the Currency to USD. The destination should be the recipient ID that corresponds to your destination wallet address, and the destination type is address_book. Also, provide your recipient email.
22.
    1.
        1. Repeat the above step, but choose the native gas token ETH instead of USD.
23.
    1.
        1. GET:https://api-sandbox.circle.com/v1/payouts/{id}
24.
    1.
        1. Use this to monitor the payout status.

3. Install Web3.js library to enable API calls

Install Web3.js by running the command below in Terminal.

// Type this command to install library npm install web3

4. Customize sample JavaScript code by inputting variables

Create Sample File

In Visual Studio, create a new JavaScript file called send10.js. Copy the code below and paste it into this new file:

JavaScript // 1. The script imports the Web3 library. const Web3 = require('web3');

// 2. Input the INFURA endpoint. const infuraEndpoint = ''; //Insert the INFURA Network Endpoint

// 3. The script initializes the web3 instance using the INFURA endpoint. const web3 = new Web3(new Web3.providers.HttpProvider(infuraEndpoint));

// 4. Input USDC token contract for Ethereum. const tokenAddress = ''; //USDC TokenAddress

// 5. The script invokes the ABI (Application Binary Interface). This is a representation of the smart contract's functions and how to call them. const minTokenAbi = [{"inputs":[{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"value","type":"uint256"}],"name":"transfer","outputs":[{"internalType":"bool","name":"","type":"bool"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"account","type":"address"}],"name":"balanceOf","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[],"name":"decimals","outputs":[{"internalType":"uint8","name":"","type":"uint8"}],"stateMutability":"view","type":"function"}]

// 6. The script uses the ABI and the token address to create a new contract instance. const contract = new web3.eth.Contract(minTokenAbi, tokenAddress)

// 7. Input the addresses and the private key; specify number of tokens to send const fromAddress ='' // Input the origin wallet address const toAddress = '' // Input the destination wallet address const privatekey = '' // Input the private key of the sender's wallet, required to sign the transaction

const amount = ; // Specify the number of USDC tokens to send

async function main() { // 8. The script checks how many decimals the USDC token uses which is usually 6. const decimals = await contract.methods.decimals().call()

// 9. The script checks the balance of the sender's address. const balance = await contract.methods.balanceOf(fromAddress).call() console.log('USDC balance:', balance / (10 ** decimals))

// 10. The script calculates the actual amount in the smallest unit (10 USDC would be 10000000 in its smallest unit). let value = amount * (10 ** decimals)

// 11. The script encodes the transfer function call for the USDC contract. let data = contract.methods.transfer(toAddress, value).encodeABI()

// 12. The script creates the transaction object. const transaction = { 'to': tokenAddress, 'gas': Web3.utils.toHex(100000), 'data': data };

// 13. The script signs the transaction with the sender's private key. const signedTx = await web3.eth.accounts.signTransaction(transaction, privatekey)

// 14. The script broadcasts the signed transaction to the Ethereum network. return web3.eth.sendSignedTransaction(signedTx.rawTransaction, function (error, hash) { if (!error) { console.log("Tx Hash: ", hash); } else { console.log("Error sending Tx:", error) } });

}

// 15. The script calls the main function and prints the receipt once it's done. main().then(receipt => console.log("Tx Receipt:", receipt))

Customize Code

Customize the code by replacing the following variables:

- Under comment 2, replace
-
- with the INFURA endpoint.
- Under comment 4, replace
-
- with the token contract address:
- 0x07865c6E87B9F70255377e024ace6630C1Eaa37F

- .
- Under comment 7:* Replace
-
  -
-
  - with the origin wallet address.
-
  - Replace
-
  -
-
  - with the destination wallet address.
-
  - Replace
-
  -
-
  - with the origin wallet’s private key.
-
  - Replace
-
  -
-
  - with the number of tokens to send:
-
  - 10
-
  -).

Save the Javascript file.

5. Execute JavaScript code to initiate USDC transfer

In Terminal, enter the following command to execute the code and transfer USDC.

// Type node .js to run code node send10.js

Check the status of your USDC transaction by visiting the Sepolia testnet block explorer :

Paste the transactionHash value from your terminal log into the search bar. Under “Status,” “Success” means the transaction has successfully settled.

Congratulations! Completing this quickstart guide for USDC has equipped you with the essential skills of programmable money.

The script provided in this tutorial can be adapted to enable real-world transactions and empower your own app development. Now, you can apply this newfound knowledge to create frictionless financial experiences, empowering users with seamless, secure global transactions. Updatedabout 23 hours ago * Table of Contents * * Overview of USDC and its token contract * * Before You Get Started * * 1. Set up a Circle Mint Sandbox Account to access to testnet USDC * * 2. Obtain testnet USDC and native gas token * * 3. Install Web3.js library to enable API calls * * 4. Customize sample JavaScript code by inputting variables * * * Create Sample File * * * Customize Code * * 5. Execute JavaScript code to initiate USDC transfer