Quickstart: Circle Paymaster

Copy page

Build a smart wallet that pays fees in USDC

Before you begin, select the version of the Circle Paymaster that you want to build with.

Paymaster v0.7 Paymaster v0.8

This guide walks you through the process of:

1. Setting up a smart account and checking its USDC balance
2. Configuring the Circle Paymaster v0.7 to pay for gas with USDC
3. Connecting to a bundler and submitting a user operation

This quickstart shows you how you can integrate Circle Paymaster into your app to simplify network fee management for your users.

Note: Throughout this tutorial, each snippet lists any new imports above the relevant code. You should add the new imports to the top of the file and merge with other imports from the same module. You can add the rest of the code inline.

Prerequisites

Before you start building the sample app to pay for gas fees in USDC, ensure that **Node.js** and **npm** are installed. You can download and install **Node.js** directly, or use a version manager like **nvm**. The npm binary comes with Node.js.

Part 1: Set up a smart account

CIRCLE | DOCS

›

1.1. Set up your development environment

Create a new project, set the package type to `module`, and install the necessary dependencies.

Shell

```
npm init
npm pkg set type="module"
npm install --save viem @circle-fin/modular-wallets-core dotenv
```

Create a new `.env` file.

Shell

```
touch .env
```

Edit the `.env` file and add the following variables, replacing `{YOUR_PRIVATE_KEY}` and `{RECIPIENT_ADDRESS}` with your own values:

```
OWNER_PRIVATE_KEY={YOUR_PRIVATE_KEY}
RECIPIENT_ADDRESS={RECIPIENT_ADDRESS}
PAYMASTER_V07_ADDRESS=0x31BE08D380A21fc740883c0BC434FcFc88740b58
USDC_ADDRESS=0x75faf114eafb1BDbe2F0316DF893fd58CE46AA4d # Arbitrum Sepolia
```

The `RECIPIENT_ADDRESS` is the destination address for the example USDC transfer.

1.2. Initialize clients and smart account

Create a file called `index.js` and add the following code to set up the necessary clients and account:

```
import { privateKeyToAccount } from "viem/accounts";
import { toCircleSmartAccount } from "@circle-fin/modular-wallets-core";

const chain = arbitrumSepolia;
const usdcAddress = process.env.USDC_ADDRESS;
const ownerPrivateKey = process.env.OWNER_PRIVATE_KEY;

const client = createPublicClient({ chain, transport: http() });
const owner = privateKeyToAccount(ownerPrivateKey);
const account = await toCircleSmartAccount({ client, owner });
```

1.3. Check the USDC balance

Check the smart account's USDC balance using the following code:

JavaScript

```
import { erc20Abi } from "viem";
const usdc = getContract({ client, address: usdcAddress, abi: erc20Abi });
const usdcBalance = await usdc.read.balanceOf([account.address]);

if (usdcBalance < 1000000) {
  console.log(
    `Fund ${account.address} with USDC on ${client.chain.name} using https://f
  );
  process.exit();
}
```

Part 2: Configure the Paymaster

The Circle Paymaster requires an allowance to spend USDC on behalf of the smart account.

2.1. Implement the permit

CIRCLE | DOCS

›

Create a new file called `permit.js` with the following code to sign EIP-2612 permits:

CIRCLE | DOCS

```
// Adapted from https://github.com/vacekj/wagmi-permit/blob/main/src/permit.ts
export async function eip2612Permit({
  token,
  chain,
  ownerAddress,
  spenderAddress,
  value,
}) {
  return {
    types: {
      // Required for compatibility with Circle PW Sign Typed Data API
      EIP712Domain: [
        { name: "name", type: "string" },
        { name: "version", type: "string" },
        { name: "chainId", type: "uint256" },
        { name: "verifyingContract", type: "address" },
      ],
      Permit: [
        { name: "owner", type: "address" },
        { name: "spender", type: "address" },
        { name: "value", type: "uint256" },
        { name: "nonce", type: "uint256" },
        { name: "deadline", type: "uint256" },
      ],
    },
    primaryType: "Permit",
    domain: {
      name: await token.read.name(),
      version: await token.read.version(),
      chainId: chain.id,
      verifyingContract: token.address,
    },
    message: {
      // Convert bigint fields to string to match EIP-712 JSON schema expectat
      owner: ownerAddress,
      spender: spenderAddress,
      value: value.toString(),
      nonce: (await token.read.nonces([ownerAddress])).toString(),
      // The paymaster cannot access block.timestamp due to 4337 opcode
```

CIRCLE | DOCS

›

```
export const eip2612Abi = [
  ...erc20Abi,
  {
    inputs: [
      {
        internalType: "address",
        name: "owner",
        type: "address",
      },
    ],
    stateMutability: "view",
    type: "function",
    name: "nonces",
    outputs: [
      {
        internalType: "uint256",
        name: "",
        type: "uint256",
      },
    ],
  },
  {
    inputs: [],
    name: "version",
    outputs: [{ internalType: "string", name: "", type: "string" }],
    stateMutability: "view",
    type: "function",
  },
];

export async function signPermit({
  tokenAddress,
  client,
  account,
  spenderAddress,
  permitAmount,
}) {
```

```
  const permitData = await eip2612Permit({
    token,
    chain: client.chain,
    ownerAddress: account.address,
    spenderAddress,
    value: permitAmount,
  });

  const wrappedPermitSignature = await account.signTypedData(permitData);

  const isValid = await client.verifyTypedData({
    ...permitData,
    address: account.address,
    signature: wrappedPermitSignature,
  });

  if (!isValid) {
    throw new Error(
      `Invalid permit signature for ${account.address}: ${wrappedPermitSignatu
    );
  }

  const { signature } = parseErc6492Signature(wrappedPermitSignature);
  return signature;
}
```

2.2. Set up Circle Paymaster

In the `index.js` file, use the Circle permit implementation to build paymaster data:

CIRCLE | DOCS

›

```
const paymasterAddress = process.env.PAYMASTER_V07_ADDRESS;

const paymaster = {
  async getPaymasterData(parameters) {
    const permitAmount = 10000000n;
    const permitSignature = await signPermit({
      tokenAddress: usdcAddress,
      account,
      client,
      spenderAddress: paymasterAddress,
      permitAmount: permitAmount,
    });

    const paymasterData = encodePacked(
      ["uint8", "address", "uint256", "bytes"],
      [0, usdcAddress, permitAmount, permitSignature],
    );

    return {
      paymaster: paymasterAddress,
      paymasterData,
      paymasterVerificationGasLimit: 200000n,
      paymasterPostOpGasLimit: 15000n,
      isFinal: true,
    };
  },
};
```

Part 3: Submit a user operation

Once the paymaster is configured, you can connect to a bundler and submit a user operation to transfer USDC.

3.1. Connect to the bundler

```
import { hexToBigInt } from "viem";

const bundlerClient = createBundlerClient({
  account,
  client,
  paymaster,
  userOperation: {
    estimateFeesPerGas: async ({ account, bundlerClient, userOperation }) => {
      const { standard: fees } = await bundlerClient.request({
        method: "pimlico_getUserOperationGasPrice",
      });
      const maxFeePerGas = hexToBigInt(fees.maxFeePerGas);
      const maxPriorityFeePerGas = hexToBigInt(fees.maxPriorityFeePerGas);
      return { maxFeePerGas, maxPriorityFeePerGas };
    },
  },
  transport: http(`https://public.pimlico.io/v2/${client.chain.id}/rpc`),
});
```

3.2. Submit the user operation

Finally, submit a user operation to transfer USDC, using the paymaster to pay for the network fee in USDC. In `index.js` add the following code:

CIRCLE | DOCS

```
  account,
  calls: [
    {
      to: usdc.address,
      abi: usdc.abi,
      functionName: "transfer",
      args: [recipientAddress, 10000n],
    },
  ],
});
console.log("UserOperation hash", hash);

const receipt = await bundlerClient.waitForUserOperationReceipt({ hash });
console.log("Transaction hash", receipt.receipt.transactionHash);

// We need to manually exit the process, since viem leaves some promises on th
// event loop for features we're not using.
process.exit();
```

Next steps

The above example demonstrates how to pay for a transaction using only USDC. You can review the transaction in an **explorer** to verify the details and see the USDC transfers that occurred during the transaction. Remember that you need to use the transaction hash from the bundler, not the user operation hash in the explorer. You can also view more details about the user operation by searching for the user operation hash in a **user op explorer** on the appropriate network.

Was this page helpful? Yes No

CIRCLE | DOCS

›

Paymaster Addresses and Events

Next ›

CIRCLE | DOCS

Legal

Developer Terms

Service Terms

Acceptable Use

Privacy

Privacy Policy

Cookie Policy

Your Privacy Choices

Powered by mintlify