MiniPay Code Library

Snippets of code that can be used to implement flows inside MiniPay

To use the code snippets below, install the following packages:

- yarn
- npm
- viem
- ethers

yarn

add @celo/abis @celo/identity viem@1 yarn

add @celo/abis @celo/identity ethers@5 * viem * ethers

npm

install @celo/abis @celo/identity viem@1 npm

install @celo/abis @celo/identity ethers@5

Check cUSD Balance of an address

- viem
- ethers

const

{ getContract , formatEther , createPublicClient , http }

=

require ("viem") ; const

{ celo }

=

require ("viem/chains") ; const

{ stableTokenABI }

=

require ("@celo/abis") ;

const

STABLE_TOKEN_ADDRESS

=

"0x765DE816845861e75A25fCA122bb6898B8B1282a" ;

async

function

checkCUSDBalance (publicClient , address)

{ let

StableTokenContract

=

getContract ({ abi : stableTokenABI , address :

```
STABLE_TOKEN_ADDRESS , publicClient , } ) ;
let balanceInBigNumber =
await
StableTokenContract . read . balanceOf ( [ address , ] ) ;
let balanceInWei = balanceInBigNumber . toString ( ) ;
let balanceInEthers =
formatEther ( balanceInWei ) ;
return balanceInEthers ; }
const publicClient =
createPublicClient ( { chain : celo , transport :
http ( ) , } ) ;
// Mainnet
let balance =
await
checkCUSDBalance ( publicClient , address ) ;
// In Ether unit const ethers =
require ( "ethers" ) ; const
{ stableTokenABI }
=
require ( "@celo/abis" ) ;
const
STABLE_TOKEN_ADDRESS
=
"0x765DE816845861e75A25fCA122bb6898B8B1282a" ;
const
{
Contract , utils , providers }
= ethers ; const
{ formatEther }
= utils ;
async
function
checkCUSDBalance ( provider , address )
{ const
StableTokenContract
=
new
```

```
Contract ( STABLE_TOKEN_ADDRESS , StableToken . abi , provider ) ;

let balanceInBigNumber =

await

StableTokenContract . balanceOf ( address ) ;

let balanceInWei = balanceInBigNumber . toString ( ) ;

let balanceInEthers =

formatEther ( balanceInWei ) ;

// Ether is a unit = 10 ** 18 wei

return balanceInEthers ; }

const provider =

new

providers . JsonRpcProvider ( "https://forno.celo.org" ) ;

// Mainnet

let balance =

await

checkCUSDBalance ( provider , address ) ;

// In Ether unit
```

Check If a transaction succeeded

- viem
- ethers

```
const

{ createPublicClient , http }

=

require ( "viem" ) ; const

{ celo }

=

require ( "viem/chains" ) ;

async

function

checkIfTransactionSucceeded ( publicClient , transactionHash )

{ let receipt =

await publicClient . getTransactionReceipt ( { hash : transactionHash , } ) ;

return receipt . status

===

"success" ; }

const publicClient =

createPublicClient ( { chain : celo , transport :

http ( ) , } ) ;
```

```
// Mainnet
let transactionStatus =
await
checkIfTransactionSucceeded ( publicClient , transactionHash ) ; async
function
checkIfTransactionSucceeded ( provider , transactionHash )
{ let receipt =
await provider . send ( "eth_getTransactionReceipt" ,
[ transactionHash , ] ) ;
return receipt . status
===
"0x1" ; }
const provider =
new
providers . JsonRpcProvider ( "https://forno.celo.org" ) ;
// Mainnet
let transactionStatus =
await
checkIfTransactionSucceeded ( provider , transactionHash ) ;
```

Estimate Gas for a transaction (in Celo)

- viem
- ethers

```
const
{ createPublicClient , http }
=
require ( "viem" ) ; const
{ celo }
=
require ( "viem/chains" ) ;
async
function
estimateGas ( publicClient , transaction , feeCurrency =
"" )
{ return
await publicClient . estimateGas ( { ... transaction , feeCurrency : feeCurrency ? feeCurrency :
"" , } ) ; }
const publicClient =
createPublicClient ( { chain : celo , transport :
```

```
http ( ) , } ) ;

let gasLimit =

await

estimateGas ( publicClient ,

{ account :

"0x8eb02597d85abc268bc4769e06a0d4cc603ab05f" , to :

"0x4f93fa058b03953c851efaa2e4fc5c34afdfab84" , value :

"0x1" , data :

"0x" , } ) ; const ethers =

require ( "ethers" ) ; const

{ providers }

= ethers ;

async

function

estimateGas ( provider , transaction , feeCurrency =

"" )

{ return

await provider . send ( "eth_estimateGas" ,

[ feeCurrency ?

{

... transaction , feeCurrency }

: transaction , ] ) ; }

const provider =

new

providers . JsonRpcProvider ( "https://forno.celo.org" ) ;

// Mainnet

// Estimate gas for an example transaction let gasLimit =

await

estimateGas ( provider ,

{ from :

"0x8eb02597d85abc268bc4769e06a0d4cc603ab05f" , to :

"0x4f93fa058b03953c851efaa2e4fc5c34afdfab84" , value :

"0x1" , data :

"0x" , } ) ;
```

Estimate Gas for a transaction (in cUSD)

- viem
- ethers

```
const
```

```
{ createPublicClient , http }
=
require ( "viem" ) ; const
{ celo }
=
require ( "viem/chains" ) ;
async
function
estimateGas ( publicClient , transaction , feeCurrency =
"" )
{ return
await publicClient . estimateGas ( { ... transaction , feeCurrency : feeCurrency ? feeCurrency :
"" , } ) ; }
const publicClient =
createPublicClient ( { chain : celo , transport :
http ( ) , } ) ;
const
STABLE_TOKEN_ADDRESS
=
"0x765DE816845861e75A25fCA122bb6898B8B1282a" ;
let gasLimit =
await
estimateGas ( publicClient , { account :
"0x8eb02597d85abc268bc4769e06a0d4cc603ab05f" , to :
"0x4f93fa058b03953c851efaa2e4fc5c34afdfab84" , value :
"0x1" , data :
"0x" , } , STABLE_TOKEN_ADDRESS ) ; const ethers =
require ( "ethers" ) ; const
{ providers }
= ethers ;
async
function
estimateGas ( provider , transaction , feeCurrency =
"" )
{ return
await provider . send ( "eth_estimateGas" ,
[ feeCurrency ?
```

{

... transaction , feeCurrency }

: transaction ,]) ; }

const provider =

new

providers . JsonRpcProvider ("https://forno.celo.org") ;

// Mainnet

const

STABLE_TOKEN_ADDRESS

=

"0x765DE816845861e75A25fCA122bb6898B8B1282a" ;

// Estimate gas for an example transaction let gasLimit =

await

estimateGas (provider , { from :

"0x8eb02597d85abc268bc4769e06a0d4cc603ab05f" , to :

"0x4f93fa058b03953c851efaa2e4fc5c34afdfab84" , value :

"0x1" , data :

"0x" , } , STABLE_TOKEN_ADDRESS) ;

Estimate Gas Price for a transaction (in Celo)

- viem
- ethers

const

{ createPublicClient , http }

=

require ("viem") ; const

{ celo }

=

require ("viem/chains") ;

async

function

estimateGasPrice (publicClient , feeCurrency =

"")

{ return

await publicClient . request ({ method :

"eth_gasPrice" , params : feeCurrency ?

[feeCurrency]

:

[ ] , }) ; }

```
const publicClient =
createPublicClient ( { chain : celo , transport :
http ( ) , } ) ;
let gasPrice =
await
estimateGasPrice ( publicClient ) ; const ethers =
require ( "ethers" ) ; const
{ providers }
= ethers ;
async
function
estimateGasPrice ( provider , feeCurrency =
"" )
{ let gasPriceinHex =
await provider . send ( "eth_gasPrice" , feeCurrency ?
[ feeCurrency ]
:
[ ] ) ;
return gasPriceinHex ; }
const provider =
new
providers . JsonRpcProvider ( "https://forno.celo.org" ) ;
// Mainnet
let gasPrice =
await
estimateGasPrice ( provider ) ;
```

Estimate Gas Price for a transaction (in cUSD)

- viem
- ethers

```
const
{ createPublicClient , http }
=
require ( "viem" ) ; const
{ celo }
=
require ( "viem/chains" ) ;
async
function
```

```
estimateGasPrice ( publicClient , feeCurrency =
"" )
{ return
await publicClient . request ( { method :
"eth_gasPrice" , params : feeCurrency ?
[ feeCurrency ]
:
[ ] , } ) ; }
const publicClient =
createPublicClient ( { chain : celo , transport :
http ( ) , } ) ;
const
STABLE_TOKEN_ADDRESS
=
"0x765DE816845861e75A25fCA122bb6898B8B1282a" ;
let gasPrice =
await
estimateGasPrice ( publicClient ,
STABLE_TOKEN_ADDRESS ) ; const ethers =
require ( "ethers" ) ; const
{ providers }
= ethers ;
async
function
estimateGasPrice ( provider , feeCurrency =
"" )
{ let gasPriceinHex =
await provider . send ( "eth_gasPrice" , feeCurrency ?
[ feeCurrency ]
:
[ ] ) ;
return gasPriceinHex ; }
const provider =
new
providers . JsonRpcProvider ( "https://forno.celo.org" ) ;
// Mainnet
const
```

STABLE_TOKEN_ADDRESS

=

"0x765DE816845861e75A25fCA122bb6898B8B1282a" ;

let gasPrice =

await

estimateGasPrice (provider ,

STABLE_TOKEN_ADDRESS) ;

Calculate cUSD to be spent for transaction fees

- viem
- ethers

const

{ createPublicClient , http , formatEther }

=

require ("viem") ; const

{ celo }

=

require ("viem/chains") ;

const publicClient =

createPublicClient ({ chain : celo , transport :

http ( ) , }) ;

const

STABLE_TOKEN_ADDRESS

=

"0x765DE816845861e75A25fCA122bb6898B8B1282a" ;

// estimateGas implemented above let gasLimit =

await

estimateGas (publicClient , { account :

"0x8eb02597d85abc268bc4769e06a0d4cc603ab05f" , to :

"0x4f93fa058b03953c851efaa2e4fc5c34afdfab84" , value :

"0x1" , data :

"0x" , } , STABLE_TOKEN_ADDRESS) ;

// estimateGasPrice implemented above let gasPrice =

await

estimateGasPrice (publicClient ,

STABLE_TOKEN_ADDRESS) ;

let transactionFeesInCUSD =

formatEther (gasLimit *

hexToBigInt (gasPrice)) ; const ethers =

```
require ( "ethers" ) ; const
{ providers , utils }
= ethers ; const
{ formatEther }
= utils ;
const provider =
new
providers . JsonRpcProvider ( "https://forno.celo.org" ) ;
const
STABLE_TOKEN_ADDRESS
=
"0x765DE816845861e75A25fCA122bb6898B8B1282a" ;
// estimateGas implemented above let gasLimit =
await
estimateGas ( provider , { from :
"0x8eb02597d85abc268bc4769e06a0d4cc603ab05f" , to :
"0x4f93fa058b03953c851efaa2e4fc5c34afdfab84" , value :
"0x1" , data :
"0x" , } , STABLE_TOKEN_ADDRESS ) ;
// estimateGasPrice implemented above let gasPrice =
await
estimateGasPrice ( provider ,
STABLE_TOKEN_ADDRESS ) ;
let transactionFeesInCUSD =
formatEther ( BigNumber . from ( gasLimit ) . mul ( BigNumber . from ( gasPrice ) ) . toString ( ) ) ;
```

Resolve Minipay phone numbers to Addresses

- viem
- ethers
- index.js
- SocialConnect.js

```
const
{ createPublicClient , http }
=
require ( "viem" ) ; const
{ celo }
=
require ( "viem/chains" ) ; const
```

```
{ privateKeyToAccount }

=

require ( "viem/accounts" ) ; const

{

SocialConnectIssuer

}

=

require ( "./SocialConnect.js" ) ;

let account =

privateKeyToAccount ( process . env . ISSUER_PRIVATE_KEY ) ;

let walletClient =

createWalletClient ( { account , transport :

http ( ) , chain , } ) ;

const issuer =

new

SocialConnectIssuer ( walletClient ,

{ authenticationMethod :

AuthenticationMethod . ENCRYPTION_KEY , rawKey : process . env . DEK_PRIVATE_KEY , } ) ;

await issuer . initialize ( ) ;

const identifierType =

IdentifierPrefix . PHONE_NUMBER ;

/* * Any phone number you want to lookup * * The below phone number is registered on the testnet issuer mentioned below.
/ const identifier =

"+911234567890" ;

/* * You can lookup under multiple issuers in one request. * * Below is the MiniPay issuer address on Mainnet. * * Note:
Remember to make your environment variable ENVIRONMENT=MAINNET / let issuerAddresses =

[ "0x7888612486844Bb9BE598668081c59A9f7367FBc" ] ;

// A testnet issuer we setup for you to lookup on testnet. // let issuerAddresses =
["0xDF7d8B197EB130cF68809730b0D41999A830c4d7"];

let results =

await issuer . lookup ( identifier , identifierType , issuerAddresses ) ; const

{ federatedAttestationsABI , odisPaymentsABI , stableTokenABI , }

=

require ( "@celo/abis" ) ;

const

{ getContract }

=

require ( "viem" ) ; const
```

```
{
OdisUtils
}
=
require ( "@celo/identity" ) ; const
{
OdisContextName
}
=
require ( "@celo/identity/lib/odis/query" ) ;
const
ONE_CENT_CUSD
=
parseEther ( "0.01" ) ;
const
SERVICE_CONTEXT
= process . env . ENVIRONMENT
===
"TESTNET" ?
OdisContextName . ALFAJORES :
OdisContextName . MAINNET ;
class
SocialConnectIssuer
{ walletClient ; authSigner ;
federatedAttestationsContractAddress ; federatedAttestationsContract ;
odisPaymentsContractAddress ; odisPaymentsContract ;
stableTokenContractAddress ; stableTokenContract ;
serviceContext ; initialized =
false ;
constructor ( walletClient , authSigner )
{ this . walletClient
= walletClient ; this . authSigner
= authSigner ; this . serviceContext
= OdisUtils . Query . getServiceContext ( SERVICE_CONTEXT ) ; }
async
initialize ( )
{ this . federatedAttestationsContractAddress
```

= await

getCoreContractAddress ("FederatedAttestations") ;

this . federatedAttestationsContract

=

getContract ({ address :

this . federatedAttestationsContractAddress , abi : federatedAttestationsABI ,

// Needed for lookup publicClient ,

// Needed for registeration and de-registration walletClient :

this . walletClient , }) ;

this . odisPaymentsContractAddress

=

await

getCoreContractAddress ("OdisPayments") ; this . odisPaymentsContract

=

getContract ({ address :

this . odisPaymentsContractAddress , abi : odisPaymentsABI , walletClient :

this . walletClient , }) ;

this . stableTokenContractAddress

=

await

getCoreContractAddress ("StableToken") ; this . stableTokenContract

=

getContract ({ address :

this . stableTokenContractAddress , abi : stableTokenABI , walletClient :

this . walletClient , }) ;

this . initialized

=

true ; }

async

getObfuscatedId

(plaintextId , identifierType)

{ // TODO look into client side blinding const

{ obfuscatedIdentifier }

= await

OdisUtils . Identifier . getObfuscatedIdentifier (plaintextId , identifierType , this . walletClient . account . address , this . authSigner , this . serviceContext) ; return obfuscatedIdentifier ; }

async

checkAndTopUpODISQuota

( )

{ const remainingQuota =

await

this . checkODISQuota ( ) ;

if

(remainingQuota <

1)

{ // TODO make threshold a constant let approvalTxHash = await

this . stableTokenContract . write . increaseAllowance ([this . odisPaymentsContractAddress , ONE_CENT_CUSD ,

// TODO we should increase by more]) ;

let approvalTxReceipt = await publicClient . waitForTransactionReceipt ({ hash : approvalTxHash , }) ;

let odisPaymentTxHash = await

this . odisPaymentsContract . write . payInCUSD ([this . walletClient . account , ONE_CENT_CUSD ,

// TODO we should increase by more]) ;

let odisPaymentReceipt = await publicClient . waitForTransactionReceipt ({ hash : odisPaymentTxHash , }) ; } }

async

getObfuscatedIdWithQuotaRetry (plaintextId , identifierType)

{ if

(this . initialized)

{ try

{ return

await

this .

getObfuscatedId

(plaintextId , identifierType) ; }

catch

{ await

this .

checkAndTopUpODISQuota

( ) ; return

this .

getObfuscatedId

(plaintextId , identifierType) ; } } throw

new

Error ("SocialConnect instance not initialized") ; }

async

registerOnChainIdentifier (plaintextId , identifierType , address)

{ if

(this . initialized)

{ const obfuscatedId =

await

this . getObfuscatedIdWithQuotaRetry (plaintextId , identifierType) ;

const hash = await

this . federatedAttestationsContract . write . registerAttestationAsIssuer ([// TODO check if there are better code patterns for sending txs obfuscatedId , address , NOW_TIMESTAMP ,]) ;

const receipt =

await publicClient . waitForTransactionReceipt ({ hash , }) ;

return receipt ; } throw

new

Error ("SocialConnect instance not initialized") ; }

async

deregisterOnChainIdentifier (plaintextId , identifierType , address)

{ if

(this . initialized)

{ const obfuscatedId =

await

this . getObfuscatedIdWithQuotaRetry (plaintextId , identifierType) ; const hash = await

this . federatedAttestationsContract . write . revokeAttestation ([obfuscatedId ,

this . walletClient . account . address , address]) ;

const receipt =

await publicClient . waitForTransactionReceipt ({ hash , }) ;

return receipt ; } throw

new

Error ("SocialConnect instance not initialized") ; }

async

checkODISQuota ( )

{ if

(this . initialized)

{ const

{ remainingQuota }

=

await

OdisUtils . Quota . getPnpQuotaStatus (this . walletClient . account . address , this . authSigner , this . serviceContext) ; console . log ("Remaining Quota" , remainingQuota) ; return remainingQuota ; } throw

new

Error ("SocialConnect instance not initialized") ; }

async

lookup (plaintextId , identifierType , issuerAddresses)

{ if

(this . initialized)

{ const obfuscatedId =

await

this . getObfuscatedIdWithQuotaRetry (plaintextId , identifierType) ;

const attestations = await

this . federatedAttestationsContract . read . lookupAttestations ([obfuscatedId , issuerAddresses]) ;

return

{ accounts : attestations [1] ,

// Viem returns data as is from contract not in JSON obfuscatedId , } ; } throw

new

Error ("SocialConnect instance not initialized") ; } } * index.js * SocialConnect.js

const ethers =

require ("ethers") ; const

{ providers , utils ,

Wallet

}

= ethers ; const

{ formatEther }

= utils ;

let wallet =

new

Wallet (process . env . ISSUER_PRIVATE_KEY , provider) ;

const issuer =

new

SocialConnectIssuer (wallet ,

{ authenticationMethod :

AuthenticationMethod . ENCRYPTION_KEY , rawKey : process . env . DEK_PRIVATE_KEY , }) ;

await issuer . initialize ( ) ;

const identifierType =

IdentifierPrefix . PHONE_NUMBER ;

/* * Any phone number you want to lookup * * The below phone number is registered on the testnet issuer mentioned below. / const identifier =

"+911234567890" ;

/* * You can lookup under multiple issuers in one request. * * Below is the MiniPay issuer address on Mainnet. * * Note: Remember to make your environment variable ENVIRONMENT=MAINNET / let issuerAddresses =

["0x7888612486844Bb9BE598668081c59A9f7367FBc"] ;

// A testnet issuer we setup for you to lookup on testnet. // let issuerAddresses = ["0xDF7d8B197EB130cF68809730b0D41999A830c4d7"];

let results =

await issuer . lookup (identifier , identifierType , issuerAddresses) ; const

{ federatedAttestationsABI , odisPaymentsABI , stableTokenABI , }

=

require ("@celo/abis") ;

const ethers =

require ("ethers") ; const

{

OdisUtils

}

=

require ("@celo/identity") ; const

{

OdisContextName

}

=

require ("@celo/identity/lib/odis/query") ;

const

{

Contract , utils }

= ethers ; const

{ parseEther }

= utils ;

const

ONE_CENT_CUSD

=

parseEther ("0.01") ;

const

SERVICE_CONTEXT

= process . env . ENVIRONMENT

===

"TESTNET" ?

OdisContextName . ALFAJORES :

OdisContextName . MAINNET ;

class

SocialConnectIssuer

{ wallet ; authSigner ;

federatedAttestationsContract ; odisPaymentsContract ; stableTokenContract ; serviceContext ;

initialized

false ;

constructor (wallet , authSigner)

{ this . wallet

= wallet ; this . authSigner

= authSigner ; this . serviceContext

= OdisUtils . Query . getServiceContext (SERVICE_CONTEXT) ; }

async

initialize ( )

{ this . federatedAttestationsContract

=

new

Contract (await

getCoreContractAddress ("FederatedAttestations") , federatedAttestationsABI , this . wallet) ;

this . odisPaymentsContract

=

new

Contract (await

getCoreContractAddress ("OdisPayments") , odisPaymentsABI , this . wallet) ;

this . stableTokenContract

=

new

Contract (await

getCoreContractAddress ("StableToken") , stableTokenABI , this . wallet) ;

this . initialize

=

true ; }

async

getObfuscatedId

(plaintextId , identifierType)

{ // TODO look into client side blinding const

{ obfuscatedIdentifier }

= await

OdisUtils . Identifier . getObfuscatedIdentifier (plaintextId , identifierType , this . wallet . address , this . authSigner , this . serviceContext) ; return obfuscatedIdentifier ; }

async

checkAndTopUpODISQuota

( )

{ const remainingQuota =

await

this . checkODISQuota ( ) ;

if

(remainingQuota <

1)

{ // TODO make threshold a constant const approvalTxReceipt =

(await

this . stableTokenContract . increaseAllowance (this . odisPaymentsContract . address , ONE_CENT_CUSD

// TODO we should increase by more)) . wait ( ) ;

const odisPaymentTxReceipt =

(await

this . odisPaymentsContract . payInCUSD (this . wallet . address , ONE_CENT_CUSD

// TODO we should increase by more)) . wait ( ) ; } }

async

getObfuscatedIdWithQuotaRetry (plaintextId , identifierType)

{ if

(this . initialized)

{ try

{ return

await

this .

getObfuscatedId

(plaintextId , identifierType) ; }

catch

{ await

this .

checkAndTopUpODISQuota

( ) ; return

this .

getObfuscatedId

(plaintextId , identifierType) ; } } throw

new

Error ("SocialConnect instance not initialized") ; }

async

registerOnChainIdentifier (plaintextId , identifierType , address)

{ if

(this . initialized)

{ const obfuscatedId =

await

this . getObfuscatedIdWithQuotaRetry (plaintextId , identifierType) ;

const tx = await

this . federatedAttestationsContract . registerAttestationAsIssuer (// TODO check if there are better code patterns for sending txs obfuscatedId , address , NOW_TIMESTAMP) ;

const receipt =

await tx . wait ( ) ; return receipt ; } throw

new

Error ("SocialConnect instance not initialized") ; }

async

deregisterOnChainIdentifier (plaintextId , identifierType , address)

{ if

(this . initialized)

{ const obfuscatedId =

await

this . getObfuscatedIdWithQuotaRetry (plaintextId , identifierType) ; const tx = await

this . federatedAttestationsContract . revokeAttestation (obfuscatedId , this . wallet . address , address) ; const receipt =

await tx . wait ( ) ; return receipt ; } throw

new

Error ("SocialConnect instance not initialized") ; }

async

checkODISQuota ( )

{ if

(this . initialized)

{ const

{ remainingQuota }

=

await

OdisUtils . Quota . getPnpQuotaStatus (this . wallet . address , this . authSigner , this . serviceContext) ; console . log ("Remaining Quota" , remainingQuota) ; return remainingQuota ; } throw

new

Error ("SocialConnect instance not initialized") ; }

async

lookup (plaintextId , identifierType , issuerAddresses)

{ if

(this . initialized)

{ const obfuscatedId =

await

this . getObfuscatedIdWithQuotaRetry (plaintextId , identifierType) ; const attestations = await

this . federatedAttestationsContract . lookupAttestations (obfuscatedId , issuerAddresses) ;

return

{ accounts : attestations . accounts ,

// TODO typesafety obfuscatedId , } ; } throw

new

Error ("SocialConnect instance not initialized") ; } }Edit this page Previous Enabling Testnet in MiniPay Next Celo Bridges