Web3 Services SDKs

Suggest Edits

The Circle Web3 Services server-side SDKs provide simple, programmatic access to Circle Web3 Services APIs. You can use the SDKs to simplify application development for apps that interact with Circle Web3 Services, such as embedding secure wallets in your applications and interacting with smart contracts on the Web3 Services platform.

Prerequisites

To successfully use Circle SDKs to interact with Circle Web3 Services APIs, you must:

- Have an active Circle Web3 Services account. You can create a new account
- using the Web3 Services Console
- .
- After you create an account, generate an API Key
- to use in your requests to Circle APIs.
- Create an entity secret to use in requests with the Smart Contract Platform SDK and Developer-Controlled Programmable Wallets SDK. See the steps in Generate an entity secret
- to generate your entity secret.

For instructions about using your API key and entity secret in requests, see the section for the specific programming language you use.

Supported Programming Languages

Currently, the Web3 Services SDK is available only for Node.js . Support for additional languages will be added when available.

Available Web3 Services SDKs

The Web3 Services SDK includes the following application-specific SDKs:

- User-Controlled Programmable Wallets SDK
- Developer-Controlled Programmable Wallets SDK
- Smart Contract Platform SDK

For details about the API, see the relevant section of the Web3 Services API Reference:

- User-Controlled Programmable Wallets API Reference
- Developer-Controlled Programmable Wallets API Reference
- Smart Contract Platform API Reference

To learn more about these features and concepts, see Programmable Wallets and Smart Contract Platform in the Developer documentation.

Installing the SDKs

The three available SDKs are independent separate installations, but you can use them together for application development. You can install all the SDKs using a single command or run an install command for each SDK separately. See the section for the programming language you use for installation instructions for that SDK.

Currently, the SDK supports only Node.js. We are working on adding support for more programming languages in the near future. To indicate your language preference, please complete the Web3 Services server-side SDK programming languages survey accessible in the developer console.

Node.js

Use the commands in this section to install the Web3 Services SDKs for Node.js.

Install all available SDKs

Install using npm :

Shell npm install @circle-fin/smart-contract-platform @circle-fin/user-controlled-wallets @circle-fin/developer-controlled-wallets --save Install using yarn :

Shell yarn add @circle-fin/smart-contract-platform @circle-fin/user-controlled-wallets @circle-fin/developer-controlled-wallets Install the User-Controlled Programmable Wallets SDK

Use the following commands to install the SDK. View the package information on the npm site.

Install using npm :

Shell npm install @circle-fin/user-controlled-wallets --save Install using yarn :

Shell yarn add @circle-fin/user-controlled-wallets Install the Developer-Controlled Programmable Wallets SDK

Use the following commands to install the SDK. View the package information on the npm site.

Install using npm :

Shell npm install @circle-fin/developer-controlled-wallets --save Install using yarn :

Shell yarn add @circle-fin/developer-controlled-wallets Install the Smart Contract Platform SDK

Use the following commands to install the SDKs. View the package information on the npm site.

Install using npm :

Shell npm install @circle-fin/smart-contract-platform --save Install using yarn :

Shell yarn add @circle-fin/smart-contract-platform

Using the Web3 Services SDKs

To start using the SDKs, you first need to configure a client.

User-Controlled Programmable Wallets Client

To interact with the User-Controlled Wallets, import the factory initiateUserControlledWalletsClient from the SDK, and then initialize the client using your API key.

The following code examples demonstrate how to import the client and configure it to use your API key:

JavaScript const { initiateUserControlledWalletsClient } = require('@circle-fin/user-controlled-wallets') const client = initiateUserControlledWalletsClient({ apiKey: '' }) Alternatively:

JavaScript import { initiateUserControlledWalletsClient } from '@circle-fin/user-controlled-wallets' const client = initiateUserControlledWalletsClient({ apiKey: '', }) Interact with the client

The following code sample demonstrates how to create a transaction using the client.

JavaScript const response = await client.createTransaction({ userToken: 'dummy-user-token', amount: ['0.01'], destinationAddress: '0xa51c9c604b79a0fadbfed35dd576ca1bce71da0a', tokenId: '738c8a6d-8896-46d1-b2cb-083600c1c69b', walletId: 'a635d679-4207-4e37-b12e-766afb9b3892', fee: { type: 'level', config: { feeLevel: 'HIGH', }, }, }) console.log(response.data?.challengeId)

Developer-Controlled Programmable Wallets Client

To interact with the Developer-Controlled Wallets, import the factory initiateDeveloperControlledWalletsClient from the SDK, and then initialize the client using your API key and entity secret.

The following code examples demonstrate how to import the client and configure it to use your API key and entity secret:

JavaScript const { initiateDeveloperControlledWalletsClient } = require('@circle-fin/developer-controlled-wallets') const client = initiateDeveloperControlledWalletsClient({ apiKey: '', entitySecret: '', }) Alternatively:

JavaScript import { initiateDeveloperControlledWalletsClient } from '@circle-fin/developer-controlled-wallets' const client = initiateDeveloperControlledWalletsClient({ apiKey: '', entitySecret: '', }) Interact with the client

The following code sample demonstrates how to create a transaction using the client.

JavaScript const walletSetResponse = await client.createWalletSet({ name: 'WalletSet 1', }) console.log('Created WalletSet', walletSetResponse.data?.walletSet) const walletsResponse = await client.createWallets({ blockchains: ['MATIC-MUMBAI'], count: 2, walletSetId: walletSetResponse.data?.walletSet?.id ?? '', }) console.log('Created Wallets', walletsResponse.data?.wallets)

Smart Contract Platform Client

To interact with the Smart Contract Platform, import the factory initiateSmartContractPlatformClient from the SDK, and then initialize the client using your API key and entity secret.

The following code examples demonstrate how to import the client and configure it to use your API key and entity secret:

JavaScript const { initiateSmartContractPlatformClient } = require('@circle-fin/smart-contract-platform') const client = initiateSmartContractPlatformClient({ apiKey: '', entitySecret: '', }) Alternatively:

JavaScript import { initiateSmartContractPlatformClient } from '@circle-fin/smart-contract-platform' const client = initiateSmartContractPlatformClient({ apiKey: '', entitySecret: '', }) Interact with the client

The following code sample demonstrates how to create a simple, smart contract using the client.

JavaScript const response = await client.deployContract({ name: 'First Contract', description: 'My first hello world contract', walletId: '004735f6-d9fc-44f8-933c-672cdf3d240d', abiJson: "[\n\t{\n\t\t'inputs': [],\n\t\t'stateMutability': 'nonpayable',\n\t\t'type': 'constructor'\n\t},\n\t...", bytecode: '0x608060405234801562000011576000080fd5b50604051806040...', constructorParameters: ['TICK', 10000], feeLevel: 'MEDIUM', })console.log(response.data)

Client Configuration options

The client for each SDK accepts the following configuration parameters:

Option Required Description apiKey [x] The API Key used to authenticate against Circle APIs. entitySecret [x] Your configured entity secret. Required for Developer-Controlled Programmable Wallets and Smart Contract Platform SDKs only. storage [ ] Optional custom storage solution for persisting data. We will fall back to InMemoryStorage if none was provided. Updated22 days ago

What's Next Jump into one of three quickstarts. *Create Your First User-Controlled Wallet * Create Your First Developer-Controlled Wallet * Deploy a Smart Contract From a Template * Table of Contents * * Prerequisites * * * Supported Programming Languages * * Available Web3 Services SDKs * * Installing the SDKs * * * Node.js * * Using the Web3 Services SDKs * * * User-Controlled Programmable Wallets Client * * * Developer-Controlled Programmable Wallets Client * * * Smart Contract Platform Client * * * Client Configuration options