



# Conio

platform [iOS](#)

platform [Android](#)

spm [compatible](#)

artifactory [v0.5.0](#)

Conio SDK provides a set of Android and iOS native APIs for Conio services to let you create native applications with Crypto Wallets and Crypto Trading functionalities.

## Overview

---

- Installation
  - [iOS](#)
  - [Android](#)
- Configuration
  - [iOS](#)
  - [Android](#)
- Features
  - [User Service](#)
  - [Trading Info Service](#)
  - [Wallet Service](#)
  - [BTC Transaction Management Service](#)
  - [Trading Buy Service](#)
  - [Trading Sell Service](#)
  - [Trading Price Service](#)
  - [Swap Service](#)
  - [Transfer Service](#)
  - [Activities Service](#)

## Old

---

Old [docs](#)



# Changelog

- [iOS](#)
- [Android](#)

## iOS

[2.1.2](#) - 22-10-2024

### Changed

- Update ConioSDK

[2.1.1](#) - 04-10-2024

### Changed

- TSK-6182: Update Fetch Historical Prices

[2.1.0](#) - 18-09-2024

### Changed

- TSK-4511: Transfer Service

[2.0.1](#) - 07-08-2024

### Changed

- Update ConioSDK

[2.0.0](#) - 13-06-2024

### Added

- TSK-4503: Trading Info Service
- TSK-4504: Btc Transaction Management Service



# iOS Installation

## Prerequisites

---

- iOS 13+
- Swift 5.9

## Swift Package Manger

---

### Via Xcode

1. In Xcode, install Conio B2B SDK by navigating to *File > Add Packages*
2. In the prompt that appears, insert the repository:

```
git@bitbucket.org:squadrone/conio-sdk-b2b-ios.git
```

or

```
https://bitbucket.org/squadrone/conio-sdk-b2b-ios.git
```

### Via `Package.swift`

Simply add the following lines to `dependencies` of your `Package.swift` manifest:

```
dependencies: [  
    .package(url: "git@bitbucket.org:squadrone/conio-sdk-b2b-ios.git")  
    // ...  
],
```

Note: in order to correctly fetch package you will need to have access to project repository.

## Troubleshooting

---

If you get the following error:

```
autoreconf: failed to run aclocal: No such file or directory
```

# Android Installation

## Prerequisites

---

- Min Android SDK: 23 (Android 6.0 “Marshmallow”)

## Installation

---

The Conio Android SDK is located in a private Maven repository on *JBfrog Artifactory*, so it is necessary to configure the authentication as follow.

- Add the Artifactory credentials provided by Conio to your global `gradle.properties`

```
artifactory_user=<username provided by Conio>
artifactory_password=<password provided by Conio>
```

- Add the Conio Artifactory repository to your `build.gradle`

```
repositories {
    // ...
    maven {
        url "https://artifactory.conio.com/artifactory/gradle-release-local"
        credentials(PasswordCredentials) {
            username "${artifactory_user}"
            password "${artifactory_password}"
        }
    }
}
```

- Add the Conio SDK dependency

```
dependencies {
    // ...
    implementation 'com.conio:sdm-b2b:[VERSION]'
}
```





# iOS Configuration

`ConioB2BSDK` is divided into multiple services, each one providing a different set of APIs.

Each service is independent and can be initialized through a `ServiceConfiguration` configuration using its own factory.

```
let conioConfig = ConioConfiguration.makeTestConfiguration(baseUrl: ...)
let userService =
    UserServiceFactory().makeServiceUsingConfiguration(conioConfig)

// User Service ready to be used
userService
    .login(with: ...)
    .asPublisher()
    .sink { ... }
// ...
```

Otherwise, `ConioB2BServiceFactory` factory leverages on a single `ServiceFactory` to make the requested `Service`.

```
let conioConfig = ConioConfiguration.makeTestConfiguration(baseUrl: ...)
let userService =
    ConioB2BServiceFactory.makeServiceUsingFactory(UserServiceFactory(),
    serviceConfiguration: conioConfig)
let walletService =
    ConioB2BServiceFactory.makeServiceUsingFactory(WalletServiceFactory(),
    serviceConfiguration: conioConfig)
let activitiesService =
    ConioB2BServiceFactory.makeServiceUsingFactory(ActivitiesServiceFactory(),
    serviceConfiguration: conioConfig)
// ...
```

## Usage

The single `Service` API is initialized with its specific `Params` parameters (if necessary) and the output can be read through its `OperationResult` result.

```
// ...
let params = LoginParams
    .make(
        username: ...,
        password: ...,
        cryptoRequest: ...
    )

userService
    .login(with: params)
```

# Android Configuration

To use the Conio SDK, you need to create an instance of the `Conio` class, providing an Android `Context` and a `ConioConfiguration`.

The `ConioConfiguration` allow you to specify the execution environment of the Conio SDK (e.g. test or production) and can be created with the url of the Conio Back-end and with the related Bitcoin Network.

```
val configuration = ConioConfiguration(  
    // required  
  
    baseUrl = "https://example.test.com",  
    bitcoinNetwork = BitcoinNetwork.Testnet, // or BitcoinNetwork.Mainnet for  
    production enviroment  
  
    // optional  
  
    // http headers added to each request, usefull for debug purpose  
    headers = mapOf("header_key" to "header_value"),  
)  
  
val conio = Conio(configuration, context)
```

# User Service

The `UserService` contains all the APIs used to manage a Conio user. It provides methods to manage a Conio user.

## APIs

---

### Login

- [User Login](#)

### Signup

- [User Signup](#)

### Logout

- [User Logout](#)

### Fetch Legal Acceptances

- [Fetch Legal Acceptances](#)

### Fetch User Permissions

- [Fetch Permissions](#)

### Accept New Legal Acceptances

- [Accept New Legal Acceptances](#)

# Trading Info Service

The `TradingInfoService` contains all the APIs used to manage a Conio user trading profile and information.

## APIs

---

### Fetch Trading Fees

- [Fetch Trading Fees](#)

### Fetch Trading Summary

- [Fetch Trading Summary](#)

### Fetch Trading Limits

- [Fetch Trading Limits](#)

### Fetch Trading Report

- [Fetch Trading Report](#)

# Wallet Service

The `WalletService` contains all the APIs that provides information about the user Wallets, such balance and mnemonic.

## APIs

---

### Balance

- [Fetch Balances](#)

### Mnemonic

- [Fetch Mnemonic](#)

# BTC Transaction Management Service

The `BtcTransactionManagementService` contains all the APIs responsible for managing Bitcoin transactions, including sending bitcoin, receiving bitcoin and speeding up transactions.

## APIs

---

### Receive

- [Fetch Address](#)

### Send

- [Send Bitcoin](#)

### Speed Up

- [Speed Up Transaction](#)

### Transaction Available Fees

- [Fetch Transaction Available Fees](#)

### Speed Up Transaction Available Fees

- [Fetch Speed Up Available Fees](#)

# Trading Buy Service

The `TradingBuyService` contains all the APIs designed to facilitate the purchase of cryptocurrencies through trading operations. It provides methods for creating, updating, fetching and finalizing bid quotations.

## APIs

---

### Create New Bid

- [Create Bid](#)

### Update Existing Bid

- [Update Bid](#)

### Fetch Existing Bid

- [Fetch Bid](#)

### Buy Cryptocurrency

- [Buy](#)

# Trading Sell Service

The `TradingSellService` contains all the APIs designed to facilitate the sale of cryptocurrencies through trading operations. It provides methods for creating, updating, fetching and finalizing ask quotations.

## APIs

---

### Create New Ask

- [Create Ask](#)

### Update Existing Ask

- [Update Ask](#)

### Fetch Existing Ask

- [Fetch Ask](#)

### Sell Cryptocurrency

- [Sell](#)



# Trading Price Service

The `TradingPriceService` contains all the APIs that provides cryptocurrencies trading price information. It provides methods for fetching current or historical crypto prices and tradable metadata, including cryptocurrency ids.

## APIs

### Fetch Current Cryptocurrency Price

- [Fetch Price](#)

### Fetch Historical Cryptocurrency Price

- [Fetch Historical Prices](#)

### Fetch All Current Cryptocurrencies Prices

- [Fetch All Prices](#)

### Fetch Tradable Cryptocurrencies Metadata

- [Fetch Tradable Crypto Metadata](#)

# Swap Service

The `SwapService` contains all the APIs designed to facilitate the cryptocurrency swap functionality. It provides methods for creating, updating, fetching and finalizing swap quotations between cryptos.

## APIs

---

### Create New Swap

- [Create Swap](#)

### Update Existing Swap

- [Update Swap](#)

### Fetch Existing Swap

- [Fetch Swap](#)

### Swap Cryptocurrency

- [Swap](#)

# Transfer Service

The `TransferService` contains all the APIs designed to facilitate the cryptocurrency amount transferring from an On-Chain Wallet to an Off-Chain Wallet of the same cryptocurrency and viceversa. It provides methods for creating, updating, fetching and finalizing transfer cryptocurrency between On-Chain and Off-Chain Wallet.

## APIs

---

### Create New Transfer

- [Create Transfer](#)

### Update Existing Transfer

- [Update Transfer](#)

### Fetch Existing Transfer

- [Fetch Transfer](#)

### Transfer Cryptocurrency Amount

- [Transfer](#)

# Activities Service

The `ActivitiesService` contains all the API that provides information about wallets transactions.

## APIs

---

### Activities

- [Fetch Activities](#)

### Single Activity

- [Fetch Activity](#)