



# 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

[2.1.1](#) - 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
- TSK-4505: Trading Buy Service
- TSK-4506: Trading Sell Service
- TSK-4509: Trading Price Service
- TSK-4510: Swap Service

## Changed

- TSK-4502: User Service
- TSK-4598: Wallet Service

## 0.7.0 - 07-06-2022

### Added

- Model `CryptoSellParams` to replace `SellParams`
- `ConioError.onNetwork` to wrap network communication errors

### Changed

- `SellParams` deprecated
- Factory init used to create/refresh an ask with all user available amount

### Fixed

- Always throw `ConioError.unauthorized` on session expired

## 0.6.10 - 09-02-2022

### Changed

- Update legal acceptances

## 0.6.9 - 08-02-2022

### Changed

- Error mapping as `unauthorized` on 401 status code

## 0.6.8 - 28-01-2022

### Added

- Transaction speedup
- Reset password flow APIs
- KYC create applicant params public init

- KYC trigger check params public init
- User data handling APIs
- User permissions map update

## Changed

- Updated model: withdrawal transaction, available fee

## Fixed

- Signup B2B wallet encrypt with hashed password
- Bad cancellables store in operations
- Missing password hash on B2B signup

## 0.6.2 - 17-11-2021

### Changed

- Wallet service `walletPDFActivities`, `limit` in `PDFActivitiesParams` now optional

### Added

- User service `getLegalAcceptances`, new `preContractualInfoUrl` param in `LegalAcceptances` response

## 0.6.0 - 02-11-2021

### Added

- User service `changeEmail`

## 0.5.0 - 11-10-2021

### Changed

- Data serialization and mapping
- Code refactor and optimizations

## 0.4.0 - 13-09-2021

### Fixed

- Wrong mapping for `rangeFrom` property in `ServiceFee`

## Changed

- Update `rangeFrom` type from `UInt64?` to `FiatAmount?` in `ServiceFee`

## 0.3.3 - 07-09-2021

## Changed

- Rename `tradedFiat` to `weightedBidBalance` in `TradingInfo.swift` as per docs specifications

## 0.3.2 - 20-07-2021

## Added

- Bitcoin network `privateMainnet` and `privateTestnet`

## 0.3.1 - 19-07-2021

## Fixed

- Avoid using app bundle identifier during keychain init

## 0.3.0 - 14-07-2021

## Changed

- Added missing filters params in `ActivitiesParams` to correctly get wallet activities
- Refactor on SDK errors: `ConioError` is now the only error type throwable (check [operation](#) section)

## 0.2.0 - 06-07-2021

## Changed

- SDK configuration object `ConioConfiguration` has no default value and must be explicitly initialized

## Fixed

- Fix wrong privacy policies url mapping in `GetLegalAcceptancesOperation`
- Avoid build error on Xcode 12.4 in `OpenAPIConioBuilder`

## 0.1.6 - 25-06-2021

### Changed

- Explicit fees represented as intervals
- `WiretransferPayeeInfo` in `CreatedBid` has now two dedicated properties representing standard and custom wire transfer payee info
- `CreatedBid` now contains net cost amount `fiatAmount` and gross amount `grossFiatAmount`
- All fiat amounts are now represented as `Decimal`

## 0.1.5 - 15-06-2021

### Changed

- `Models` update
- `Bid`, `Ask` e `Transaction` properties linked to amount/balance now are declared with type `UInt64`

### Added

- `ConioError` entity to map operation errors

## 0.1.4 - 10-06-2021

### Changed

- `Models` update
- `Bid`, `Ask`, `WalletBalances` e `SimpleActivity` properties now have public control access
- `Bid`, `Ask`, `WalletBalances` e `SimpleActivity` properties linked to amount/balance now are declared with type `UInt64`

### Removed

- Removed `SwiftRSA` from dependencies included in `ConioSDK`

## 0.1.3 - 03-06-2021

### Fixed

- Correzione errore signup operation



0.1.0 - 12-04-2021

**Added**

- Rilascio versione 0.1.0

Android

0.8.11 - 7-06-2022

**Added**

- `ConioException.OnNetwork` to wrap network communication errors

**Fixed**

- Always throw `ConioException.Unauthorized` on session expired

0.8.3 - 24-03-2022

**Fixed**

- Initialization error caused by unusable KeyStore keys

0.8.0 - 9-03-2022

**Changed**

- Minimum Android version supported to *Android 6.0* (Android Sdk Version: 23)
- Improved performance
- `SellParams` deprecated

**Added**

- Factory method `CreateOrRefreshAskParams.withAll` to request an Ask with the maximum sellable amount
- Model `CryptoChangeEmailParams` to replace `ChangeEmailParams`
- Model `CryptoSellParams` to replace `SellParams`

0.7.18 - 8-02-2022

**Changed**

- Solved retro-compatibility with OkHttp 3.x

- Removed appsync dependency

### 0.7.16 - 4-02-2022

#### Changed

- Downgraded OkHttp to 3.14.9

### 0.7.15 - 3-02-2022

#### Changed

- Improved concurrency on service layer
- Updated OkHttp to 4.9.0

### 0.7.13 - 24-01-2022

#### Fixed

- Compatibility issue below Api level 26

### 0.7.9 - 26-11-2021

#### Changed

- Legal text copies on the `LegalAcceptances` model

### 0.7.8 - 17-11-2021

#### Changed

- Wallet service `activityListPdf`, `limit` in `ActivityListPdfParams` now nullable

#### Added

- User service `getLegalAcceptances`, new `preContractualInfoUrl` param in `LegalAcceptances` response

### 0.7.4 - 02-11-2021

#### Added

- User service `changeEmail`

## 0.7.2 - 20-10-2021

### Added

- API to get activities in PDF format

## 0.7.0 - 11-10-2021

### Changed

- Data serialization and mapping
- Code refactor and optimizations

## 0.6.2 - 03-08-2021

### Fixed

- Security issue

## 0.6.1 - 29-07-2021

### Changed

- Refactor on SDK errors: `ConioException` as the operations result error type

## 0.6.0 - 28-07-2021

### Changed

- Refactor on SDK errors: `ConioException` is now the only error type throwable (check [operation](#) section)

## 0.5.4 - 26-07-2021

### Fixed

- Made `cro`, `iban` and `chargedAt` fields of `Ask` class optional
- Made `paidAt` field of `Ask` class non-optional

## 0.5.3 - 20-07-2021

### Added

- Bitcoin network `privateMainnet` and `privateTestnet`

## 0.5.1 - 14-07-2021

### Fixed

- Fix factory methods of `TimeFrame` class

## 0.5.0 - 06-07-2021

### Changed

- SDK configuration object `ConioConfiguration` has no default value and must be explicitly initialized

## 0.4.8 - 25-06-2021

### Changed

- Explicit fees represented as intervals
- `WiretransferPayeeInfo` in `CreatedBid` has now two dedicated properties representing standard and custom wire transfer payee info
- `CreatedBid` now contains net cost amount `fiatAmount` and gross amount `grossFiatAmount`
- All fiat amounts are now represented as `BigDecimal`

### Removed

- Removed `type` property from `ServiceFee` entity
- Renamed `id` property of model entities:
  - `CreatedAsk.id` -> `CreatedAsk.askId`
  - `CreatedBid.id` -> `CreatedBid.bidId`
  - `SimpleActivity.id` -> `SimpleActivity.activityId`
  - `ActivityDetails.id` -> `ActivityDetails.activityId`

### Added

- `ConioError`:
  - `INVALID_CRYPTOPROOF`,
  - `CRYPTOPROOF_EXPIRED`

#### 0.4.7 - 01-06-2021

##### **Added**

- Aggiunta di `weightedBidBalance` alle `TradingInfo` : controvalore investito

##### **modified**

- Modifica alle `TradingFees` : supporto fasce di commissioni

#### 0.4.2 - 13-04-2021

##### **Added**

- Rilascio versione 0.4.2

#### 0.4.1 - 12-04-2021

##### **Added**

- Rilascio versione 0.4.1



# 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
```

try the following command using [Brew](#):

```
brew install autoconf && brew install automake
```

If you get the following error: Can't exec "/opt/local/bin/aclocal": No such file or directory Uninstall MacPorts with: `sudo port -fp uninstall --follow-dependents` installed



# 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:sdk-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)
```

```

        .asPublisher()
        .sink { result in
            switch result {
            case .success:
                // ...
            case .failure(let error):
                // ...
            }
        }
        .store(in: ...)
    // ...

```

Each API is returned as `ServiceConsumer` and can be consumed in three different ways:

- `asPublisher()` , used to handle the result in a declarative way leveraging on [Combine](#);
- `asCallback()` , used to handle the result in closure/lambda style as self-contained block;
- `run()` , used to execute the API without handling the result.

```

// asPublisher()
let cancellable = userService
    .logout()
    .asPublisher()
    .sink { result in
        // ...
    }

// asCallback()
userService
    .logout()
    .asCallback { result in
        // ...
    }

// run()
userService
    .logout()
    .run()

```

# 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](#)