- English

Conio

Questo SDK rende semplice integrare un portafoglio Bitcoin Conio nella propria app.

Cosa si può fare?

- Creare un portafoglio Bitcoin
- Ottenere le informazioni del portafoglio creato
- Comprare e Vendere Bitcoin
- Leggere il dettaglio storico delle operazioni di un utente
- Ottenere il prezzo storico ed attuale del Bitcoin

Come posso utilizzarlo?

- Bilancio del portafoglio
- Codice di recupero Bitcoin

- Operazioni sul mercato
  - Prezzo attuale del Bitcoin
  - Prezzo storico del Bitcoin
  - Riepilogo trading
  - Commissioni trading
  - Limiti di trading
  - Acquisto di Bitcoin
  - Vendita di Bitcoin

- Italiano

Conio

This SDK makes it simple to integrate a Bitcoin wallet Conio in your app.

What can you do?

- Create a Bitcoin wallet
- Show wallet details
- Show the historical and current Bitcoin price
- Send and receive Bitcoin
- Buy and Sell Bitcoin
- Show all the transactions made by the wallet

How does it work?

- Italiano: Questa pagina non è tradotta in italiano

Changelog

- iOS
- Android

iOS

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 `SwiftyRSA` 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

0.7.4 - 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_CRYPTO_PROOF,
  - CRYPTO_PROOF_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

- English

Installazione

Prerequisiti

- SDK supporta iOS 10+
- Autoconf installato
- Automake installato
- Libtool installato

È consigliato l'utilizzo del gestori di pacchetti MacOS Brew per l'installazione di `Autoconf`, `Automake` e `Libtool`.

```
# Install Brew
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
HEAD/install.sh)"

# Install Autoconf, Automake and Libtool
brew install autoconf automake libtool
```

Installazione con Cocoapods

L'SDK Conio è disponibile come Pod ed è possibile includerla nei progetti aggiungendo le seguenti righe al Podfile:

```
# The ConioSDK Core
pod 'ConioSDK', :git => 'git@bitbucket.org:squadrone/conio-swift-
sdk.git', :branch => 'master'

# BitcoinKit for encryption purposes
pod 'BitcoinKit', :git => 'https://github.com/Conio/BitcoinKit.git', :branch
=> 'keyconvert'
```

Eseguire il comando `pod install` nella cartella per ottenere l'SDK.

Possibili Errori nell'installazione

Se si dovesse verificare il seguente messaggio di errore:

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

Eseguire il comando:

`brew install autoconf && brew install automake`.

Se si dovesse verificare il seguente messaggio di errore:

```
Can't exec "/opt/local/bin/aclocal": No such file or directory
```

Disinstallare dal sistema MacPorts eseguendo:

`sudo port -fp uninstall --follow-dependents installed`

- Italiano

Installation

Prerequisites

- iOS 10+
- Autoconf installed
- Automake installed
- Libtool installed

Consider using MacOS package manager Brew to install `Autoconf`, `Automake` e `Libtool`.

```
# Install Brew
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
HEAD/install.sh)"

# Install Autoconf, Automake and Libtool
brew install autoconf automake libtool
```

Cocoapods install

Add this line to your podfile:

```
# The ConioSDK Core
pod 'ConioSDK', :git => 'git@bitbucket.org:squadrone/conio-swift-
sdk.git', :branch => 'master'

# BitcoinKit for encryption purposes
pod 'BitcoinKit', :git => 'https://github.com/Conio/BitcoinKit.git', :branch
=> 'keyconvert'
```

Then use the command: `pod install`

Troubleshooting

If you get the following error:

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

Try the following command:

`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`

- English

Installazione su Android

L'SDK si installa utilizzando il repository Maven di Artifactory. Per potersi autenticare al repository è necessario configurare le credenziali nel file **gradle.properties** come segue:

gradle.properties

```
artifactory_user={username}
artifactory_password={password}
```

A questo punto sarà possibile aggiungere l'indirizzo del repository nel **build.gradle** dell'applicazione:

app/build.gradle

```
repositories {

    ...

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

Dopo aver specificato l'indirizzo del repository dal quale verranno sincronizzati gli artefatti sarà possibile aggiungere il **Conio SDK** come dipendeza dell'applicazione:

app/build.gradle

```
dependencies {

    ...

    implementation 'com.conio:sdk2:[VERSION]'
}
```

Sincronizzando il progetto con Gradle sarà possibile utilizzare l'SDK.

- Italiano

Install on Android

You can install the SDK using Artifactory as Maven repository. To authenticate you have to put your credentials in the app **gradle.properties** file:

gradle.properties

```
artifactory_user={username}
artifactory_password={password}
```

Then in the app **build.gradle** file add the repository address:

app/build.gradle

```
repositories {

    ...

    maven {
        url "https://d314astu88ufzo.cloudfront.net/artifactory/gradle-release-
local"
        credentials(PasswordCredentials) {
            username "${artifactory_user}"
            password "${artifactory_password}"
        }
    }
}
```

Finally add **Conio SDK** as app dependency:

app/build.gradle

```
dependencies {

    ...

    implementation 'com.conio:sdk2:[VERSION]'
}
```

Then just sync Gradle files.

- English

Inizializzazione dell'SDK

L'oggetto Conio

Per usare l'SDK, occorre inizializzare l'oggetto `Conio` con una `ConioConfiguration`. La configurazione determinerà l'ambiente con il quale l'SDK interagirà.

È necessario inizializzare l'SDK con un ambiente personalizzato, specificando l'url del backend e la rete Bitcoin da utilizzare.

Di seguito le specifiche per inizializzare un oggetto di tipo `Conio`.

Parametri: **Conio**

- **configuration**: di tipo ConioConfiguration, la configurazione per inizializzare l'SDK;
- (Android) **context**: di tipo `Context`, il context dell'applicazione Android.

Parametri: ConioConfiguration

- **baseUrl**: di tipo `String`, url del backend;
- **bitcoinNetwork**: di tipo `BitcoinNetwork`, la rete Bitcoin. Può essere `.testnet`, `.mainnet`, `privateMainnet` o `privateTestnet`.

Codice

Android

```
import com.conio.sdk.Conio;
import com.conio.sdk.models.shared.BitcoinNetwork;
import com.conio.sdk.models.shared.ConioConfiguration;
import com.conio.sdk.providers.networking.NetworkEnvironment;

// Test configuration
ConioConfiguration testConfig = new ConioConfiguration("https://
example.test.com", BitcoinNetwork.TESTNET);
Conio conio = new Conio(testConfig, context);

// Production configuration
ConioConfiguration config = new ConioConfiguration("https://
example.production.com", BitcoinNetwork.MAINNET);
Conio conio = new Conio(config, context);
```

iOS

```
import ConioSDK

// Test configuration
let testConfig = ConioConfiguration(
    withBaseUrl: "https://example.test.com",
    bitcoinNetwork: .testnet
)
let conio = Conio(config: testConfig)

// Production configuration
let config = ConioConfiguration(
    withBaseUrl: "https://example.production.com",
    bitcoinNetwork: .testnet
)
let conio = Conio(config: config)
```

- Italiano

Initialize the SDK

Conio object

The `Conio` object needs to be initialized with a `ConioConfiguration`.

You can use the `test` configuration that will connect to the staging environment and the Bitcoin testnet blockchain. The `prod` environment instead will connect to the production server and the Bitcoin original blockchain.

You can also initialize the SDK with a custom environment, with the url of the backend and a Bitcoin blockchain.

Parameters

- **configuration**: configuration to initialize the SDK: ConioConfiguration type
- (Android) **context**: context will save in the Shared Preferences

Conio Configuration

- **identifier**: name of the configuration
- **bitcoinNetwork**: the Bitcoin network, either `.testnet` or `.mainnet`
- **networkEnvironment**: the environment (which backend): NetworkEnvironment type

Network Environment

- **name**: name of the environment
- **host**: the host

Code

Android

```
import com.conio.sdk.Conio;
import com.conio.sdk.models.shared.BitcoinNetwork;
import com.conio.sdk.models.shared.ConioConfiguration;
import com.conio.sdk.providers.networking.NetworkEnvironment;

// Test configuration
```

```
Conio conio = new Conio(ConioConfiguration.test, getApplicationContext());

// Production configuration
Conio conio = new Conio(ConioConfiguration.prod, getApplicationContext());
```

iOS

```
import ConioSDK

// Test configuration
let conio = Conio(configuration: ConioConfiguration.test)

// Production configuration
let conio = Conio(configuration: ConioConfiguration.prod)
```

- English: This page isn't translated to English.

Operazioni

Introduzione

Una volta inizializzato l'oggetto Conio, i servizi offerti dal SDK sono raggruppati in 3 categorie:

1. Servizi dell'utente (`conio.userService.*`);
2. Servizi del wallet (`conio.walletService.*`);
3. Servizi di mercato (`conio.exchangeService.*`).

Ogni servizio è un metodo il cui valore di ritorno è un implementazione dell'interfaccia `ServiceConsumer<O>`.

ServiceConsumer

L'interfaccia `ServiceConsumer<O>` (generica in `O`, il tipo che rappresenta il risultato del servizio stesso) dichiara le modalità con cui i risultati dei servizi possono essere fruiti, infatti espone i metodi:

- `asCallbeck`, che richiede come parametro una callback che verrà invocata con il risultato del servizio;
- (Android) `asFlow`, che restituisce un oggetto di tipo Flow, più adatto al paradigma di programmazione reattiva;
- (iOS) `asPublisher`, TODO.

Code

ANDROID (JAVA)

```
conio.walletService.currentBitcoinAddress().asCallback(result ->
result.analysis(
        address -> { /* ... */ },
        error -> { /* ... */ }
));
```

ANDROID (KOTLIN)

```
runBlocking {
    conio.walletService.currentBitcoinAddress().asFlow().first().analysis(
            { address -> /* ... */ },
            { error -> /* ... */ }
```

```
    )
}
```

Eccezioni possibili

ConioError (iOS)

Questo errore raggruppa tutte le possibili risposte di errore direttamente legate alle operazioni.

```
// General operation error with name and/or description
case onOperation(String)
// Decoding data error
case unableToDecodeData
// Cryptographic operation error
case onCryptography
// Secure storage operation error
case onStorage
// OAuth flow error: unable to retrieve and/or refresh access token
case unauthorized
// TBD
case appImprovementAcceptanceNotAccepted
// TBD
case clientSupportAcceptanceNotAccepted
// Ask operation already paid
case askAlreadyPaid
// Bid operation already paid
case bidAlreadyPaid
// Bid operation is expired
case bidExpired
// TBD
case bidIsInError
// Bid operation is not yet paid
case bidNotYetPaid
case bithustlerServiceCouldNotCreateSeller
// TBD
case cardsLimitsExceeded
case cardsServiceCouldNotCreatePayer
case duplicateEmailAddress
case dustAsk
case dustTransaction
// Fiat amount is under the minumum level limit
case fiatAmountTooLow
case inconsistentState
case inconsistentTransaction
case invalidIban
case invalidMessageSignature
// Used payment method is not valid
case invalidPaymentMethod
case invalidToken
case invalidTokenPayload
// Crypto proof used for operation is invalid
```

```
case invalidCryptoProof
case multipleSellMethods
case noSuch3DSecure
case noSuchSellMethod
// TBD
case noSuchSeller
// TBD
case noSuchWallet
case noSuchWithdrawalFeesInfo
case notEnoughBtcAmount
case tradeExpired
// Bid operation exceeded user purchase max limits
case tradingLimitsExceeded
case unavailableBtcSubsystem
// Ask operation is in an error status
case unrecoverableAsk
// Bid operation is in an error status
case unrecoverableBid
// Payment method used in not supported
case unsupportedPaymentMethod
case walletAlreadyCreatedWithDifferentKeys
case walletAlreadyOwnedByAnotherUser
// Unknown error with description
case unknown(String)
// Conio SDK version is outdated
case outdatedSdk
// Server is under maintenance
case underMaintenance
```

ConioException (Android)

Questo errore raggruppa (sia come namespace che come supertipo) tutte le possibili risposte di errore direttamente legate alle operazioni.

```
sealed class ConioException : Exception {
    // General operation error with name and/or description
    class OnOperation : ConioException
    // Decoding data error
    class UnableToDecodeData : ConioException
    // Cryptographic operation error
    class OnCryptography : ConioException
    // Secure storage operation error
    class OnStorage : ConioException
    // OAuth flow error: unable to retrieve and/or refresh access token
    class Unauthorized : ConioException
    // Conio SDK version is outdated
    class OutdatedSdk : ConioException
    // Server is under maintenance
    class UnderMaintenance : ConioException


    class AppImprovementAcceptanceNotAccepted : ConioException

    class ClientSupportAcceptanceNotAccepted : ConioException
```

```
    // Ask operation already paid
    class AskAlreadyPaid : ConioException
    // Bid operation already paid
    class BidAlreadyPaid : ConioException
    // Bid operation is expired
    class BidExpired : ConioException

    class BidIsInError : ConioException
    // Bid operation is not yet paid
    class BidNotYetPaid : ConioException

    class BithustlerServiceCouldNotCreateSeller : ConioException

    class CardsLimitsExceeded : ConioException

    class CardsServiceCouldNotCreatePayer : ConioException

    class DuplicateEmailAddress : ConioException

    class DustAsk : ConioException

    class DustTransaction : ConioException
    // Fiat amount is under the minumum level limit
    class FiatAmountTooLow : ConioException

    class InconsistentState : ConioException

    class InconsistentTransaction : ConioException

    class InvalidIban : ConioException

    class InvalidMessageSignature : ConioException
    // Used payment method is not valid
    class InvalidPaymentMethod : ConioException

    class InvalidToken : ConioException

    class InvalidTokenPayload : ConioException
    // Crypto proof used for operation is invalid
    class InvalidCryptoProof : ConioException

    class MultipleSellMethods : ConioException

    class NoSuch3DSecure : ConioException

    class NoSuchSellMethod : ConioException

    class NoSuchSeller : ConioException

    class NoSuchWallet : ConioException

    class NoSuchWithdrawalFeesInfo : ConioException

    class NotEnoughBtcAmount : ConioException

    class TradeExpired : ConioException
    // Bid operation exceeded user purchase max limits
```

```
    class TradingLimitsExceeded : ConioException

    class UnavailableBtcSubsystem : ConioException
    // Ask operation is in an error status
    class UnrecoverableAsk : ConioException
    // Bid operation is in an error status
    class UnrecoverableBid : ConioException
    // Payment method used in not supported
    class UnsupportedPaymentMethod : ConioException

    class WalletAlreadyCreatedWithDifferentKeys : ConioException

    class WalletAlreadyOwnedByAnotherUser : ConioException
    // Unknown error with description
    class Unknown : ConioException
}
```

Ad esempio, prendiamo l'operazione `conio.walletService.withdrawalFees`: se un utente ha 1 bitcoin nel portafoglio e richiede le mining fees per un invio da 50 bitcoin, riceverà un `NO_SUCH_WITHDRAWAL_FEES_INFO`.

Code

ANDROID

```
WithdrawalFeesParams params = new WithdrawalFeesParams(
    "mkHS9ne12qx9pS9VojpwU5xtRd4T7X7ZUt",
    100000000,
    TransactionSpeedType.SPEED_FIVE
);

conio.walletService.withdrawalFees(params).asCallback(result ->
result.analysis(
        fees -> { /* ... */ },
        error -> {
            ConioException conioException = (ConioException) error;
            if (conioException.getConioError() ==
ConioError.NO_SUCH_WITHDRAWAL_FEES_INFO) {
                /* Handle NO_SUCH_WITHDRAWAL_FEES_INFO error */
            }
        }
));
```

Non autorizzato

Questo errore viene generato quando non si è autorizzati a utilizzare un metodo per uno dei seguenti motivi:

- utilizzo di un metodo che richiede autenticazione senza una sessione valida;
- si sta provando ad effettura una login con credenziali errate.

Assicurarsi di avere una sessione valida, autenticandosi nuovamente tramite una login o una sign-up.

Codice

ANDROID

```
UserLogin user = new UserLogin("username", "wrong_password");

conio.userService.login(user).asCallback(result -> result.analysis(
    success -> { /* ... */ },
    error -> {
        if (error instanceof ConioException.Unauthorized) {
            /* Handle the error */
        }
    }
));
```

SDK obsoleto

Questo errore viene generato quando l'utente tenta di utilizzare una versione obsoleta dell'SDK.

Consigliamo di gestire questo errore per notificare all'utente di aggiornare l'applicazione.

Code

ANDROID

```
LegalAcceptancesParams params = new LegalAcceptancesParams(Language.ITALIAN);

conio.userService.getLegalAcceptances(params).asCallback(result ->
result.analysis(
    acceptances -> { /* ... */ },
    error -> {
        if (error instanceof ConioException.OutdatedSdk) {
            /* Handle the error */
        }
    }
));
```

IOS

```
let params = LegalAcceptancesParams(language: .italian)

conio.userService.getLegalAcceptances(params: params).asCallback { result in
    switch result {
    case .success:
        // success
    case .failure(let error):
        if case .outdatedSdk = error {
```

```
            print("Please update the SDK")
        }
    }
}
```

- English: This page isn't translated to English.

Crypto Request

Alcune funzionalità del SDK Conio sono protette da un meccanismo chiamato **Crypto Request**, che aggiunge un livello di sicurezza ulteriore all'invio di alcuni parametri, tramite una firma crittografica.

Le richieste che sfruttano questo meccanismo sono riconoscibili dalla presenza della proprietà `cryptoRequest`, di tipo `[Name]CryptoRequest`, presente nella funzione di costruzione (costruttore o metodo factory) del oggetto da passare come parametro all'operazione. In particolare, le funzionalità protette da questo meccanismo sono:

1. `userService.signup`, registrazione dell'utente (`SignupCryptoRequest`);
2. `userService.login`, autenticazione dell'utente (`LoginCryptoRequest`);
3. `exchangeService.purchase`, acquisto di Bitcoin (`BidCryptoRequest`);
4. `exchangeService.sell`, vendita di Bitcoin (`AskCryptoRequest`).

La costruzione di ogni proprietà di tipo `[Name]CryptoRequest` necessita di un parametro `cryptoProof`, un **array di byte**, ottenuto tramite firma `RSA` del hash `SHA256` della concatenazione (con separatore "|") ordinata delle altre proprietà del tipo `[Name]CryptoRequest` (come descritto per ogni tipo `[Name]CryptoRequest` nel apposito paragrafo).

```
NFC=<implementazione algoritmo di conversione stringa - array di byte>
SHA256=<implementazione algoritmo di hashing SHA256>
RSA_SIGN=<implementazione algoritmo di firma RSA>

CRYPTO_PROOF = RSA_SIGN(SHA256(NFC(DATA_TO_SIGN)))
```

Creazione SignupCryptoRequest

Proprietà

- **proofID**: di tipo `String`, identificativo della Crypto Request;
- **userID**: di tipo `String`, identificativo esterno del utente;
- **userLevel**: di tipo `String`, livello del utente che ne stabilisce i limiti di compravendita;
- **proofExpiration**: di tipo `long`, istante temporale dopo il quale la Crypto Request non è più considerata valida;

- **@Opzionale iban**: di tipo `String`, iban del conto bancario associato all'utente, utilizzato come metodo di pagamento per le operazioni di vendita;
- **email**: di tipo `String`, email dell'utente;
- **firstName**: di tipo `String`, nome dell'utente;
- **lastName**: di tipo `String`, cognome dell'utente;

DATA_TO_SIGN

```
DATA_TO_SIGN="<proofID>|SIGNUP|<userID>|<userLevel>|<proofExpiration>|<email>|
<firstName>|<lastName>"

or

DATA_TO_SIGN="<proofID>|SIGNUP|<userID>|<userLevel>|<proofExpiration>|<iban>|
<email>|<firstName>|<lastName>"
```

Nota: il campo **iban** è opzionale, pertanto, se non lo si inserisce nella `SignupCryptoReqeust`, va rimosso anche dalla stringa `DATA_TO_SIGN` (insieme al separatore "|")

Creazione LoginCryptoRequest

Proprietà

- **userID**: di tipo `String`, identificativo esterno del utente;
- **proofExpiration**: di tipo `long`, istante temporale dopo il quale la Crypto Request non è più considerata valida;

DATA_TO_SIGN

```
DATA_TO_SIGN="<userID>|LOGIN|<proofExpiration>"
```

Creazione AskCryptoRequest

Proprietà

- **proofID**: di tipo `String`, identificativo della Crypto Request;
- **askID**: di tipo `String`, identificativo della `CreatedAsk` che si vuole finalizzare;
- **userID**: di tipo `String`, identificativo esterno del utente;
- **proofExpiration**: di tipo `long`, istante temporale dopo il quale la Crypto Request non è più considerata valida;

DATA_TO_SIGN

```
DATA_TO_SIGN="<proofID>|PAY_FOR_ASK|<askID>|<userID>|<proofExpiration>"
```

Creazione BidCryptoRequest

Proprietà

- **proofID**: di tipo `String`, identificativo della Crypto Request;
- **bidID**: di tipo `String`, identificativo dell `CreatedBid` che si vuole finalizzare;
- **userID**: di tipo `String`, identificativo esterno del utente;
- **proofExpiration**: di tipo `long`, istante temporale dopo il quale la Crypto Request non è più considerata valida;

DATA_TO_SIGN

```
DATA_TO_SIGN="<proofID>|PAY_FOR_BID_WT|<bidID>|<userID>|<proofExpiration>"
```

- English

Operazioni sull'utente

Recupero dei termini di servizio

Questa operazione consente di recuperare le `LegalAcceptances`, ovvero le condizioni che l'utente potrà/dovrà accettare in fase di signup (scelte che, durante l'operazione signup, dovranno essere descritte tramite la classe `Acceptances`). L'oggetto `LegalAcceptances` recuperato conterrà gli url per mostrare le pagine dei Termini di Servizio e Privacy Policies di Conio e il dettaglio delle *acceptances* (`AcceptanceDetail`) che l'utente dovrà o meno accettare.

Metodo

`userService.getLegalAcceptances`

Parametri

Un oggetto di tipo `LegalAcceptancesParams` contenente la lingua di riferimento per ottenere le acceptances e gli url delle pagine web da mostrare all'utente.

Risposta

Un oggetto di tipo `LegalAcceptances` contenente la lista degli `AcceptanceDetail`, lo url dei *Termini di Servizio* e quello delle *Privacy Policies*.

Codice

Android

```
LegalAcceptancesParams params = new LegalAcceptancesParams(Language.ITALIAN);

conio.userService.getLegalAcceptances(params)
    .asCallback(result -> result.analysis(
        acceptances -> { /* Handle LegalAcceptances */ },
        error -> { /* Handle error */ }
    ));
```

iOS

```
let params = LegalAcceptancesParams(language: .italian)

conio.userService.signup(params: params).asCallback { result in
    switch result {
    case .success(let acceptances):
        // LegalAcceptances
    case .failure(let error):
        // Operation Error
    }
}
```

Autenticazione

Per poter operare con il portafoglio Conio occorre essere autenticati. Se è la prima volta che l'utente usa il servizio ci si può autenticare con il metodo `userService.signup`, altrimenti con il metodo `userService.login`.

Signup

L'operazione di signup permette di creare un nuovo utente Conio.

Metodo

`userService.signup`

Parametri

Un oggetto di tipo `SignupParams`, costruito tramite il metodo `SignupParams.createCryptoSignup` con:

- ***acceptances***: di tipo `Acceptances` con l'esito della conferma ai termini di servizio da parte dell'utente, recuperati tramite le `LegalAcceptances`;
- ***credentials***: di tipo `ConioCredentials` con username e password dell'utente;
- ***cryptoRequest***: di tipo `SignupCryptoRequest`, che specifica ulteriori parametri comprovati da una firma, come descritto in creazione della SignupCryptoRequest.

Risposta

Un oggetto di tipo `Success` che indica che l'utente è stato autenticato.

Errori

- ConioError:
  - `APP_IMPROVEMENT_ACCEPTANCE_NOT_ACCEPTED` Acceptance obbligatoria;
  - `CLIENT_SUPPORT_ACCEPTANCE_NOT_ACCEPTED` Acceptance obbligatoria;
  - `CRYPTO_PROOF_EXPIRED` La crypto proof è scaduta;
  - `INVALID_CRYPTO_PROOF` La crypto proof non è correttamente firmata;
  - `DUPLICATE_EMAIL_ADDRESS` Indirizzo email duplicato;
  - `INVALID_IBAN` IBAN non valido;
  - `WALLET_ALREADY_OWNED_BY_ANOTHER_USER` Il wallet è già utilizzato da un altro utente;
  - `CARDS_SERVICE_COULD_NOT_CREATE_PAYER` Errore interno del sottosistema di pagamento.

Codice

Android

```
// vedi "Creazione SignupCryptoRequest"
SignupCryptoRequest cryptoRequest = new SignupCryptoRequest(...);
ConioCredentials credentials = new ConioCredentials("username", "password");
Acceptances acceptances = new Acceptances(Arrays.asList(
    new Acceptance(AcceptanceType.CLIENT_SUPPORT, true),
    new Acceptance(AcceptanceType.APP_IMPROVEMENT, true)
));

SignupParams params = SignupParams.createCryptoSignup(acceptances,
credentials, cryptoRequest);

conio.userService.signup(params)
    .asCallback(result -> result.analysis(
        success -> { /* Handle success */ },
        error -> { /* Handle error */ }
    ));
```

iOS

```
// vedi "Creazione SignupCryptoRequest"

let credentials = ConioCredentials(username: "username", password: "password")

 var acceptancesList = [Acceptance]()
        acceptancesList.append(.init(type: .appImprovement, isAccepted: true))
        acceptancesList.append(.init(type: .clientSupport, isAccepted: true))
let acceptances = Acceptances(acceptances: acceptancesList)
let cryptoRequest = SignupCryptoRequest.init(proofID: "", cryptoProof:
Data(), proofExpiration: 0, externalUserID: "", userLevel: "", iban: "",
email: "", firstName: "", lastName: "")
```

```
let signupParams = SignupParams.createCryptoSignup(credentials: credentials,
acceptances: acceptances, cryptoRequest: cryptoRequest)

conio.userService.signup(params: params).asCallback { result in
    switch result {
    case .success:
        // Handle Succes
    case .failure(let error):
        // Operation Error
    }
}
```

Login

L'operazione di login permette di autenticarsi a Conio. È **raccomandabile** eseguire questa operazione ad ogni avvio dell'applicazione, similmente a come avviene per altri servizi terzi.

Metodo

`userService.login`

Parametri

Un oggetto di tipo `LoginParams`, costruito tramite il metodo `LoginParams.createCryptoLogin` con:

- **credentials**: di tipo `ConioCredentials` con username e password dell'utente
- **cryptoRequest**: di tipo `LoginCryptoRequest`, che specifica ulteriori parametri comprovati da una firma, come descritto in creazione della LoginCryptoRequest.

Risposta

Un oggetto di tipo `Success` che indica che l'utente è stato autenticato.

Errori

- Non autorizzato

Codice

Android

```
LoginCryptoRequest cryptoRequest = new LoginCryptoRequest(...);
ConioCredentials credentials = new ConioCredentials("username", "password");
```

```
LoginParams params = LoginParams.createCryptoLogin(credentials,
cryptoRequest);

conio.userService.login(params)
    .asCallback(result -> result.analysis(
        success -> { /* Handle success */ },
        error -> { /* Handle error */ }
    ));
```

iOS

```
let params = LoginParams(username: "lemonade", password: "secretword",
loginCryptoRequest: <LoginCryptoRequest>)
conio.userService.login(params: params).asCallback { result in
    switch result {
    case .success:
        // success
    case .failure(let error):
        // Operation Error
    }
}
```

Logout

Consente di disconnettere l'utenza Conio.

Metodo

`userService.logout`

Risposta

Un oggetto di tipo `Success` che indica che l'utente è stato disconnesso.

Codice

Android

```
conio.userService.logout()
    .asCallback(result -> result.analysis(
        success -> { /* Handle success */ },
        error -> { /* Handle error */ }
    ));
```

iOS

```
conio.userService.logout().asCallback { result in
    switch result {
```

```
    case .success:
        // success
    case .failure(let error):
        // Operation Error
    }
}
```

Change Email

Consente di modificare l'email associata all'utenza Conio.

Metodo

`userService.changeEmail`

Parametri

Un oggetto di tipo `ChangeEmailParams`.

- ***newEmail***: di tipo `String`, è il nuovo valore utilizzato per modificare l'attuale email dell'utente.

Risposta

Android

Un oggetto di tipo `Success` che indica se l'email dell'utente è stata modificata correttamente.

iOS

Un oggetto di tipo `Void` che indica se l'email dell'utente è stata modificata correttamente.

Codice

Android

```
ChangeEmailParams params = new ChangeEmailParams("newEmail@conio.com");

conio.userService.changeEmail(params).asCallback(result -> result.analysis(
        activityList -> { /* Success */ },
        error -> { /* ... */ }
));
```

iOS

```
let params = ChangeEmailParams(newEmail: "newEmail@conio.com")
conio.userService.changeEmail(with: params).asCallback { result in
    switch result {
    case .failure(let error):
        /* ... */
    case .success:
        /* Success */
    }
}
```

- Italiano

User operations

Terms and conditions

Using this operation you can retrieve the `Acceptances` (terms and conditions), T&C URL and Privacy Policy URL that the user has to accept during the signup.

Parameters

An object `LegalAcceptancesParams` with the language you want.

Returns

A `LegalAcceptances` object containing `Acceptances`, the URL `Termini di Servizio` and the URL `Privacy Policies`.

Acceptances localization

An `Acceptance` has 2 localization keys: one for the title and one for the content.

Code

Android

```
LegalAcceptancesParams params =
    new LegalAcceptancesParams(Language.ITALIAN);
conio.userService.getLegalAcceptances(params, result -> {
    result.analysis(acceptances -> {
        // LegalAcceptances
    }, error -> {
        // Exception
    });
});
```

iOS

```
let params = LegalAcceptancesParams(language: .italian)

conio.userService.getLegalAcceptances(params: params) { result in
    result.analysis(ifSuccess: { legalAcceptances in
        // LegalAcceptances
    }, ifFailure: { error in
```

```
        // ServiceError
    })
}
```

Signup

To use the wallet the user has to be authenticated. If it's the first time you can authenticate using the signup method, otherwise you have to use the login method.

Parameters

An `Account` struct containing:

- ***login***: `Login` on iOS or `UserLogin` on Android: username and password of the user.
- ***acceptances***: `Acceptances` containing booleans about the user consent to T&C
- ***cryptoRequest***: create a `CryptoRequest` : Crypto Request Creation

Crypto Request Creation

To generate a `Crypto Request` , you have to sign the string: `dataString` (create one by following the example below), using the function sha256 and the private key. The following lines of code are just an example. The actual implementation of the signing algorithm to include in the `CryptoRequest` is up to the client.

Java Example

```
    String proofId = UUID.randomUUID().toString();
    long proofExpiration =
        new Date()
        .tenMinutesFromNow()
        .millis();

    String userLevel = "A smart level"; // Es. "Advanced" to get adavanced
limits
    String userId = login.username;
    String iban = "IBAN"; // It should be a real iban
    String email = "user@email.com";
    String firstName = "Mario";
    String lastName = "Rossi";

    String[] data = {
        proofId,
        "SIGNUP",
        userId,
        userLevel,
        String.valueOf(proofExpiration),
        iban,
        email,
```

```
        firstName,
        lastName
    };

    String dataString = join("|", data);

    PrivateKey privateKey = new PrivateKey("key.pem");
    RsaSigner rsa = new RsaSigner(privateKey);

    String signature =
        rsa
        .sign("sha256", dataString)
        .toLowercase();

    byte[] cryptoProof = fromHexToBytes(signature);
```

Swift Example

```
    let proofID = UUID().uuidString
    let proofExpiration: UInt64 = UInt64(Date())
    let userLevel = "A smart level" // Es. "Advanced" to get adavanced limits
    let userID = login.username
    let iban = "IBAN" // It should be a real iban
    let email = "user@email.com"
    let firstName = "Mario"
    let lastName = "Rossi"

    let data = [
        proofID,
        "SIGNUP",
        userID,
        userLevel,
        String(proofExpiration),
        iban,
        firstName,
        lastName
    ]

    let dataString = data.joined(separator: "|")

    let cryptoProof = Crypto.sign(
        privateKey: privateKey,
        digestType: .sha256
    )

    let cryptoRequest = CryptoRequest(
        proofID: proofID,
        cryptoProof: cryptoProof.data,
        proofExpiration: proofExpiration,
        userID: userID,
        userLevel: userLevel,
        iban: iban,
        email: email,
        firstName: firstName,
        lastName: lastName
    )
```

Returns

An object `Acceptances` confirming which T&C the user approved during the signup.

Errori

- `INVALID_IBAN`
- `CRYPTO_PROOF_EXPIRED`
- `INVALID_CRYPTO_PROOF` Crypto proof was signed incorrectly
- `CARDS_SERVICE_COULD_NOT_CREATE_PAYER` Internal error of the payment system
- `DUPLICATE_EMAIL_ADDRESS`
- `WALLET_ALREADY_OWNED_BY_ANOTHER_USER`
- `CLIENT_SUPPORT_ACCEPTANCE_NOT_ACCEPTED` Required acceptance
- `APP_IMPROVEMENT_ACCEPTANCE_NOT_ACCEPTED` Required acceptance

Code

Android

```
UserLogin login = new UserLogin("lemonade", "secretword");

// Build the acceptances list with the user choices result
Acceptance appImprovement
    = new Acceptance(AcceptanceType.APP_IMPROVEMENT, true);
Acceptance clientSupport
    = new Acceptance(AcceptanceType.CLIENT_SUPPORT, true);

ArrayList<Acceptance> acceptanceList = new ArrayList<>();
acceptanceList.add(appImprovement);
acceptanceList.add(clientSupport);

Acceptances acceptances = new Acceptances(acceptanceList);

// Your crypto request implementation
CryptoRequest cryptoRequest = buildCryptoRequest();

Account account = new Account(login, acceptances, cryptoRequest);
conio.userService.signup(account, result -> {
    result.analysis(acceptances -> {
        // Acceptances
    }, error -> {
        // Exception
    });
});
```

iOS

```
let login = Login(username: "lemonade", password: "secretword")

// Your crypto request implementation
let cryptoRequest = buildCryptoRequest()

// Build the acceptances list with the user choices result
let appImprovement =
    Acceptance(type: .appImprovement, isAccepted: true)
let clientSupport =
    Acceptance(type: .clientSupport, isAccepted: true)

let acceptancesList = [appImprovement, clientSupport]
let acceptances = Acceptances(acceptances: acceptancesList)

let account = Account(
    login: login,
    acceptances: acceptances,
    cryptoRequest: cryptoRequest
)

conio.userService.signup(with: account) { result in
    result.analysis(ifSuccess: { acceptances in
        // Acceptances
    }, ifFailure: { error in
        // ServiceError
    })
}
```

Login

Using the login operation you can authenticate to Conio. It is **recommended** to perform this operation every time the app is started.

Parameters

An object, called `Login` on iOS or `UserLogin` on Android, containing:

- **username**
- **password**

Returns

An `Acceptances` object with the T&C that the user accepted on signup.

Code

Android

```
UserLogin login = new UserLogin("lemonade", "secretword");
conio.userService.login(login, result -> {
    result.analysis(acceptances -> {
        // Acceptances
    }, error -> {
        // Exception
    });
});
```

iOS

```
let login = Login(username: "lemonade",password: "secretword")
conio.userService.login(with: login) { result in
    result.analysis(ifSuccess: { acceptances in
        // Acceptances
    }, ifFailure: { error in
        // ServiceError
    })
}
```

Logout

Disconnect from Conio.

Returns

A `boolean` with the result of the operation.

Code

Android

```
conio.userService.logout(result -> {
    result.analysis(success -> {
        // Boolean
    }, error -> {
        // Exception
    });
});
```

iOS

```
conio.userService.logout { result in
    result.analysis(ifSuccess: { success in
```

```
        // Boolean
    }, ifFailure: { error in
        // ServiceError
    })
}
```

Change Email

Using this operation you can update Conio user email.

Method

`userService.changeEmail`

Parameters

A `ChangeEmailParams` object type.

- ***newEmail***: `String` type, it is the new value used to update actual user email.

Returns

Android

A `Success` object type if the operation finish with success.

iOS

A `Void` object type if the operation finish with success.

Code

Android

```
ChangeEmailParams params = new ChangeEmailParams("newEmail@conio.com");

conio.userService.changeEmail(params).asCallback(result -> result.analysis(
        activityList -> { /* Success */ },
        error -> { /* ... */ }
));
```

iOS

```
let params = ChangeEmailParams(newEmail: "newEmail@conio.com")
conio.userService.changeEmail(with: params).asCallback { result in
    switch result {
```

```
    case .failure(let error):
        /* ... */
    case .success:
        /* Success */
    }
}
```

- English

Operazioni sul mercato

Prezzo attuale del Bitcoin

È possibile recuperare il miglior prezzo di acquisto e di vendita attuale del bitcoin, specificando la valuta nel quale lo si vuole ottenere. Inoltre, l'SDK offre la possibilità di convertire un ammontare in bitcoin nella valuta specificata.

Metodo

`exchangeService.currentPrice`

Parametri

Un oggetto di tipo `CurrentPriceParams` contenente:

- **currency**: di tipo `Currency`, la valuta in cui si vuole ottenere il prezzo;
- **@Opzionale cryptoAmount**: di tipo `long`, l'ammontare in **satoshi** (1 bitcoin = 100.000.000 satoshi) che si vuole convertire nella valuta indicata.

Risposta

Un `CurrentPrice` contenente:

- **buyFiatAmount**: di tipo `Decimal` (iOS) / `BigDecimal` (Android), il prezzo di acquisto, calcolato nella valuta indicata tramite il campo **currency**;
- **sellFiatAmount**: di tipo `Decimal` (iOS) / `BigDecimal` (Android), il prezzo di vendita, calcolato nella valuta indicata tramite il campo **currency**.

Codice

Android

```
// Example 1: get current price
CurrentPriceParams params = new CurrentPriceParams(Currency.EUR);

// Example 2: get current price of a specified amount
CurrentPriceParams params = new CurrentPriceParams(Currency.EUR, 100000000);

conio.exchangeService.currentPrice(params)
```

```
    .asCallback(result -> result.analysis(
        currentPrice -> { /* Handle CurrentPrice */ },
        error -> { /* ... */ }
    ));
```

iOS

```
// Recupero del prezzo attuale
let params = CurrentPriceParams(currency: .eur)

// Conversione di 50.000.000 satoshi (0,5 BTC) in euro
let params = CurrentPriceParams(currency: .eur, satoshiAmount: 50_000_000)

let consumer = conio.exchangeService.currentPrice(params: params)
consumer.asCallback { result in
    switch result {
    case .success(let prices):
        // CurrentPrice
    case .failure(let error):
        // Operation Error
    }
}
```

Prezzo storico del Bitcoin

È possibile recuperare il prezzo storico del Bitcoin selezionando una finestra temporale di riferimento.

Metodo

`exchangeService.historicalPrices`

Parametri

Un oggetto di tipo `HistoricalPricesParams` contenente:

- **currency**: di tipo `Currency`, la valuta in cui si vuole ottenere il prezzo;
- **timeFrame**: di tipo `TimeFrame`, la finestra temporale di riferimento;
- **@Default(24h) interval**: di tipo `long`, l'intervallo che si vuole porre tra i prezzi restituiti;

Risposta

Un `HistoricalPrices` contenente:

- **prices**: di tipo `List<PricePoint>`, la lista dei prezzi del bitcoin nella finestra temporale specificata;

- **analytics**: di tipo `PriceAnalytics`, contenente:
  - **deltaFiat**: di tipo `Decimal` (iOS) / `BigDecimal` (Android), la variazione in valuta del prezzo del Bitcoin dall'inizio del periodo di riferimento;
  - **deltaPercentage**: la variazione in percentuale del prezzo del Bitcoin dall'inizio del periodo di riferimento;
  - **trend**: di tipo `PriceTrend`, un enumerato che rappresenta se il prezzo del Bitcoin, dall'inizio del periodo di riferimento, è cresciuto, è diminuito o è rimasto stagnante;

Codice

Android

```
// Example 1: get last month prices with default interval (1 day)
HistoricalPricesParams params = new HistoricalPricesParams(
    Currency.EUR,
    TimeFrame.lastMonth()
);

// Example 2: get prices from 16th April 2018 to 16th April 2019 with 1 week
interval
HistoricalPricesParams params = new HistoricalPricesParams(
    Currency.EUR,
    new TimeFrame(1523885446000L, 1563465540000L),
    604800000
);

conio.exchangeService.historicalPrices(params)
    .asCallback(result -> result.analysis(
        prices -> { /* Handle HistoricalPrices */ },
        error -> { /* ... */ }
    ));
```

iOS

```
// Prezzo dal 16 aprile 2019 al 16 aprile 2018
// Intervallo standard: 1 giorno
let params = HistoricalPriceParams(currency: .eur, startTimestamp:
1523885446000, endTimestamp: 1563465540000)

// Prezzo dal 16 aprile 2019 al 16 aprile 2018
// Intervallo selezionato: 1 settimana
let params = HistoricalPriceParams(currency: .eur, startTimestamp:
1523885446000, endTimestamp: 1563465540000, interval: 604800000)

let consumer = conio.exchangeService.historicalPrices(params: params)
consumer.asCallback { result in
    switch result {
    case .success(let prices):
        // HistoricalPrices
    case .failure(let error):
        // Operation Error
```

```
    }
}
```

Recupero informazioni di trading

Recupero delle informazioni riassuntive delle operazioni di compravendita eseguite dall'utente.

Metodo

`exchangeService.tradingInfo`

Parametri

Un oggetto di tipo `TradingInfoParams`, contenente:

- **currency**: di tipo `Currency`, la valuta sulla quale si vuole ottenere la risposta;

Risposta

- **weightedBidBalance**: di tipo `Decimal` (iOS) / `BigDecimal` (Android), controvalore investito, calcolato come la media pesata del valore (in valuta fiat) degli acquisti moltiplicato per il bilancio attuale;
- **currency**: di tipo `Currency`, la valuta di riferimento della risposta;
- **bidSummary**: di tipo `TradingSummary`, contenente un riepilogo delle operazioni di acquisto;
- **askSummary**: di tipo `TradingSummary`, contenente un riepilogo delle operazioni di vendita;

Le proprietà di tipo `TradingSummary` contengono:

- **operationsCount**: di tipo `intero`, il numero totale di operazioni;
- **totalFiatAmount**: di tipo `Decimal` (iOS) / `BigDecimal` (Android), l'ammontare totale delle operazioni.

Codice

Android

```
TradingInfoParams params = new TradingInfoParams(Currency.EUR);

conio.exchangeService.tradingInfo(params)
    .asCallback(result -> result.analysis(
```

```
        info -> { /* Handle TradingInfo */ },
        error -> { /* ... */ }
    ));
```

```
let consumer = conio.exchangeService.tradingInfo()
consumer.asCallback { result in
    switch result {
    case .success(let info):
        // Handle TradingInfo
    case .failure(let error):
        // Operation Error
    }
}
```

Recupero commissioni di trading

Per recuperare le informazioni delle commissioni sulle operazioni di compravendita.

Metodo

`exchangeService.tradingFees`

Parametri

Un oggetto di tipo `TradingFeesParams`, contenente:

- **currency**: di tipo `Currency`, la valuta sulla quale si vuole ottenere la risposta;

Risposta

Un oggetto di tipo `TradingFees`, contenente:

- **currency**: di tipo `Currency`, la valuta di riferimento della risposta;
- **bidServiceFees**: di tipo `List<ServiceFee>`, contenente la lista delle fascie di commissioni per le operazioni di acquisto;
- **askServiceFees**: di tipo `List<ServiceFee>`, contenente la lista delle fascie di commissioni per le operazioni di vendita.

Le proprietà di tipo `ServiceFee` contengono:

- **rangeFrom**: di tipo `Decimal` (iOS) / `BigDecimal` (Android), il valore (in valuta fiat) dal quale viene applicata;
- **@Opzionale percentage**: di tipo `double`, la percentuale di commissione rispetto al valore del operazione, nulla se la `ServiceFee` rappresenta una commissione assoluta;

- **@Opzionale fiatAmount**: di tipo `Decimal` (iOS) / `BigDecimal` (Android), la commisione assuluta applicata su ogni operazione, nulla se la `ServiceFee` rappresenta una commissione in percentuale.

Codice

Android

```
TradingFeesParams params = new TradingFeesParams(Currency.EUR);

conio.exchangeService.tradingFees(params)
    .asCallback(result -> result.analysis(
        fees -> { /* Handle TradingFees */ },
        error -> { /* ... */ }
    ));
```

```
let consumer = conio.exchangeService.tradingFees()
consumer.asCallback { result in
    switch result {
    case .success(let fees):
        // Handle TradingFees
    case .failure(let error):
        // Operation Error
    }
}
```

Recupero limiti di trading

Per recuperare i limiti di compravendita associati ad un utente, assegnati in fase di signup tramite il campo **userLevel**.

Metodo

`exchangeService.tradingLimits`

Risposta

Un oggetto di tipo `AllTradingLimits`, contenente:

- **buyLimits**: di tipo `TradingLimits`, contenenti informazioni sui limiti di acquisto;
- **sellLimits**: di tipo `TradingLimits`, contenenti informazioni sui limiti di vendita.

Le proprietà di tipo `TradingLimits` contengono:

- **minFiatAmount**: di tipo `Decimal` (iOS) / `BigDecimal` (Android), il limite minimo attualmente a disposizione;

- **maxFiatAmount**: di tipo `Decimal` (iOS) / `BigDecimal` (Android), il limite massimo attualmente a disposizione;
- **allLimits**: di tipo `List<Limit>`, una lista di limiti, ciascuno conterrà la rispettiva tipologia (`DAILY`, `MONTHLY`, `YEARLY`) ed il valore per ciascuno di essi;
- **currentLimits**: di tipo `List<Limit>`, il valore residuo per ciascuno dei limiti contenuti nell'oggetto `allLimits` del punto precedente.

Codice

Android

```
conio.exchangeService.tradingLimits()
    .asCallback(result -> result.analysis(
        limits -> { /* Handle AllTradingLimits */ },
        error -> { /* ... */ }
    ));
```

iOS

```
let consumer = conio.exchangeService.tradingLimits()
consumer.asCallback { result in
    switch result {
    case .success(let fees):
        // Handle AllTradingLimits
    case .failure(let error):
        // Operation Error
    }
}
```

Acquisto di Bitcoin

Per poter acquistare dei Bitcoin è necessario effettuare due operazioni. La prima è quella di creazione di una `Bid`, ovvero di una richiesta di acquisto di una determinata somma di Bitcoin ad un certo prezzo. All'interno della `Bid` si troveranno le `WiretransferInfo` che dovranno essere usate dal client per effettuare il pagamento. Infine si dovrà utilizzare la seconda operazione verso Conio per comunicare l'avvenuto pagamento della `Bid` allegando anche una `BidCryptoRequest`, generata client side, per testimoniare la legittimità dell'operazione.

Creazione della Bid

Una `Bid` si crea specificando la **valuta** che si intende utilizzare per la transazione e l'importo, o in satoshi o in valuta corrente. Ad esempio, sarà quindi possibile richiedere una `Bid` per l'acquisto di 150€ di Bitcoin o una `Bid` per l'acquisto di 100.000.000 satoshi.

Una volta inviata la richiesta, si otterrà una `CreatedBid` contenente, tra le altre informazioni un `bidId`. Con questo identificativo sarà possibile **aggiornare** la richiesta di Bid per rimandarne la scadenza e per ottenere le informazioni sul tasso di cambio più aggiornate. Questo scenario è utile nei casi in cui tra la richiesta della Bid e l'effettiva azione dell'utente passi del tempo che renderebbe il tasso di cambio obsoleto.

Metodo

`exchangeService.createOrRefreshBid`

Parametri

Un oggetto di tipo `CreateOrRefreshBidParams`, costruibile tramite i metodi factory `CreateOrRefreshBidParams.fromFiat` o `CreateOrRefreshBidParams.fromCrypto` che richiedono:

- **currency**: di tipo `Currency`, la valuta dell'operazione;
- **amount**: di tipo `long` per `.fromCrypto` o `Decimal` (iOS) / `BigDecimal` (Android) per `.fromFiat`, l'ammontare, a seconda del metodo usato, in satoshi o nella valuta scelta che si vuole acquistare;
- **@Opzionale bidId**: di tipo `String`, l'id della bid, da valorizzare solo in caso di refresh della bid stessa.

Risposta

Un oggetto di tipo `CreatedBid` che contiene:

- **id**: di tipo `String`, l'id utile al refresh o alla finalizzazione della bid;
- **currency**: di tipo `Currency`, la valuta dell'operazione;
- **cryptoAmount**: di tipo `long`, l'ammontare in satoshi della richiesta d'acquisto;
- **fiatAmount**: di tipo `Decimal` (iOS) / `BigDecimal` (Android), l'ammontare in valuta corrente della richiesta d'acquisto al netto delle commissioni;
- **grossFiatAmount**: di tipo `Decimal` (iOS) / `BigDecimal` (Android), l'ammontare in valuta corrente della richiesta d'acquisto comprensivo delle commissioni;
- **serviceFee**: di tipo `Decimal` (iOS) / `BigDecimal` (Android), le commissioni di servizio per la transazione, espresse nella **currency** di riferimento
- **expiration**: di tipo `long`, il timestamp di scadenza della richiesta di pagamento. Se la bid scade sarà necessario aggiornarla per proseguire
- **wireTransferInfo**: di tipo `WireTransferInfo`, le informazioni necessarie per procedere al pagamento della Bid tramite bonifico.

Codice

Android

```
// Example 1: €200 bid
CreateOrRefreshBidParams params =
CreateOrRefreshBidParams.fromFiat(Currency.EUR, 20000);
// Example 2: 1.000.000.000 satoshi bid
CreateOrRefreshBidParams params =
CreateOrRefreshBidParams.fromCrypto(Currency.EUR, 1000000000);

conio.exchangeService.createOrRefreshBid(params)
    .asCallback(result -> result.analysis(
        bid -> { /* Handle CreatedBid */ },
        error -> { /* ... */ }
    ));
```

iOS

```
// Richiesta d'acquisto per 50€
let params = CreateOrRefreshBidParams(currency: .eur, fiatAmount: 50.0)

// Richiesta d'acquisto per 1.000.000 satoshi
 let params = CreateOrRefreshBidParams(currency: .eur, satoshi: 1000000)

// Aggiornamento di una richiesta d'acquisto per 100€
let params = CreateOrRefreshBidParams(bidID: "bididentifier", currency: .eur,
fiatAmount: 100.0)

let consumer = conio.exchangeService.createOrRefreshBid(params: params)
consumer.asCallback { result in
    switch result {
    case .success(let bid):
        // Handle CreatedBid
    case .failure(let error):
        // Operation Error
    }
}
```

Utilizzo della Bid (pagamento)

Una volta effettuato il pagamento tramite bonifico si dovrà usare l'operazione `purchase` per comunicare a Conio l'avvenuto pagamento. Questa operazione richiederà una `BidCryptoRequest`.

Metodo

`exchangeService.purchase`

Parametri

Un oggetto di tipo `PurchaseParams` contenente:

- **bidId**: d tipo `String`, l'id della `Bid` da pagare
- **cryptoRequest**: di tipo `BidCryptoRequest`, configurabile come descritto nell'apposita sezione

Risposta

Un oggetto di tipo `Success`, che conferma l'avvenuta operazione.

Errori

- INVALID_CRYPTO_PROOF La crypto proof non è valida
- INVALID_PAYMENT_METHOD Il metodo di pagamento non è valido
- UNSUPPORTED_PAYMENT_METHOD Il metodo di pagamento non è supportato
- TRADING_LIMITS_EXCEEDED La bid viola i limiti massimi di acquisto dell'utente
- TRADE_EXPIRED La bid è scaduta
- BID_ALREADY_PAID La bid è già stata pagata
- BID_NOT_YET_PAID La bid non è ancora stata pagata
- UNRECOVERABLE_BID La bid è in errore
- FIAT_AMOUNT_TOO_LOW L'importo in Fiat è inferiore al limite minimo

Codice

Android

```
PurchaseParams params = new PurchaseParams("bidId", bidCryptoRequest);

conio.exchangeService.purchase(params)
    .asCallback(result -> result.analysis(
        success -> { /* Handle Success */ },
        error -> { /* ... */ }
    ));
```

iOS

```
let params = PurchaseParams(bidId: "bidId", cryptoRequest: bidCryptoRequest)
let consumer = conio.exchangeService.purchase(params: params)
consumer.asCallback { result in
    switch result {
    case .success:
```

```
        // Handle Success
    case .failure(let error):
        // Operation Error
    }
}
```

Vendita di Bitcoin

Per poter vendere dei Bitcoin è necessario effettuare due operazioni. La prima è quella di creazione di una `Ask`, ovvero di una richiesta di vendita di una determinata somma di Bitcoin ad un certo prezzo. Si procede poi con il pagamento di tale `Ask`, passando l'`askId` e allegando anche una `AskCryptoRequest`, generata client side, per testimoniare la legittimità dell'operazione. L'SDK firmerà la transazione che sposterà i Bitcoin dal wallet dell'utente, restituendo alla fine l'id della `Ask` completata.

Creazione della Ask

Per richiedere una `Ask` si dovrà procedere analogamente a quanto visto per la Bid. Sarà quindi possibile richiedere una `CreatedAsk` per la vendita di 150€ o una per la vendita di 100.000.000 satoshi.

Una volta inviata la richiesta, si otterrà una `CreatedAsk` contenente, tra le altre informazioni un `askId`. Con questo identificativo sarà possibile **aggiornare** la richiesta di Ask per rimandarne la scadenza e per ottenere le informazioni sul tasso di cambio più aggiornate. Questo scenario è utile nei casi in cui tra la richiesta della Ask e l'effettiva azione dell'utente passi del tempo che renderebbe il tasso di cambio obsoleto.

Metodo

`exchangeService.createOrRefreshAsk`

Parametri

Un oggetto di tipo `CreateOrRefreshAskParams`, costruibile tramite i metodi factory `CreateOrRefreshAskParams.fromFiat` o `CreateOrRefreshAskParams.fromCrypto` che richiedono:

- **currency**: di tipo `Currency`, la valuta dell'operazione;
- **amount**: di tipo `long` per `.fromCrypto` o `Decimal` (iOS) / `BigDecimal` (Android) per `.fromFiat`, l'ammontare, a seconda del metodo usato, in satoshi o nella valuta scelta che si vuole vendere;

- **@Opzionale askId**: di tipo `String`, l'id della ask, da valorizzare solo in caso di refresh della ask stessa.

Risposta

Un oggetto di tipo `CreatedAsk` che contiene:

- **askId**: di tipo `String`, l'id utile al refresh o alla finalizzazione della ask;
- **currency**: di tipo `Currency`, la valuta dell'operazione;
- **cryptoAmount**: di tipo `long`, l'ammontare in satoshi della richiesta d'acquisto
- **fiatAmount**: di tipo `Decimal` (iOS) / `BigDecimal` (Android), l'ammontare in valuta corrente della richiesta d'acquisto;
- **serviceFee**: di tipo `Decimal` (iOS) / `BigDecimal` (Android), le commissioni di servizio per la transazione, espresse nella **currency** di riferimento;
- **miningFee**: di tipo `long`, le commissioni per scrivere la transazione in blockchain, espresse in satoshi;
- **expiration**: di tipo `long`, lo Unix Timestamp di scadenza della richiesta di pagamento. Se la bid scade sarà necessario aggiornarla per proseguire

Errori

- TRADING_LIMITS_EXCEEDED L'utente ha 0 Eur di limiti residui
- NOT_ENOUGH_BTC_AMOUNT solo se non ha btc L'utente non ha alcun bitcoin
- NO_SUCH_SELLER Errore interno del sottosistema di vendita
- NO_SUCH_WALLET Errore interno del sottosistema di wallet

Android

```
// Example 1: €200 ask
CreateOrRefreshAskParams params =
CreateOrRefreshAskParams.fromFiat(Currency.EUR, BigDecimal("200"));
// Example 2: 1.000.000.000 satoshi ask
CreateOrRefreshAskParams params =
CreateOrRefreshAskParams.fromCrypto(Currency.EUR, 1000000000);

conio.exchangeService.createOrRefreshAsk(params)
    .asCallback(result -> result.analysis(
        ask -> { /* Handle CreatedAsk */ },
        error -> { /* ... */ }
    ));
```

iOS

```
// Richiesta di vendita per 50€
let params =
    CreateOrRefreshAskParams(currency: .eur, fiatAmount: 50.0)

// Richiesta di vendita per 100000000 satoshi
let params =
    CreateOrRefreshAskParams(currency: .eur, satoshi: 100000000)

// Aggiornamento del valore di una Ask esistente
let params =
    CreateOrRefreshAskParams(askID: "id", currency: .eur, fiatAmount: 100.0)


conio.exchangeService.createOrRefreshAsk(params: params).asCallback { result
in
    switch result {
    case .success(let createdAsk):
        // CreatedBid
    case .failure(let error):
        // Operation Error
    }
}
```

Utilizzo della Ask

Ottenuta la `Ask` da utilizzare è possibile procedere con la finalizzazione della vendita. Per effettuare questa operazione bisognerà passare l'ID della `CreatedAsk` alla `Sell` operation, insieme alla `AskCryptoRequest`.

Metodo

`exchangeService.sell`

Parametri

Un oggetto di tipo `SellParams` contenente:

- **askId**: di tipo `String`, l'id della `Ask`
- **cryptoRequest**: di tipo `AskCryptoRequest`, configurabile come descritto nell'apposita sezione

Risposta

Un oggetto di tipo `Success` che conferma l'avvenuta operazione.

Errori

- TRADING_LIMITS_EXCEEDED La ask viola i limiti massimi di acquisto dell'utente
- TRADE_EXPIRED La ask è scaduta
- UNRECOVERABLE_ASK La ask è in errore
- ASK_ALREADY_PAID La ask è già stata pagata
- NOT_ENOUGH_BTC_AMOUNT_E Bitcoin disponibili non sufficienti
- DUST_ASK Importo in Bitcoin troppo piccolo
- FIAT_AMOUNT_TOO_LOW Importo in Eur troppo basso

Codice

Android

```
SellParams params = new SellParams("askId", askCryptoRequest);

conio.exchangeService.sell(params)
    .asCallback(result -> result.analysis(
        success -> { /* Handle Success */ },
        error -> { /* ... */ }
    ));
```

iOS

```
let params = SellParams(askID: askID)

conio.exchangeService.createOrRefreshAsk(params: params).asCallback { result
in
    switch result {
    case .success:
        // Handle Succes
    case .failure(let error):
        // Operation Error
    }
}
```

- Italiano

Exchange operations

Current Bitcoin Price

You can get the current buy or sell Bitcoin price. The SDK can also give you the Bitcoin equivalent for a set amount of currency.

Parameters

An object of type `CurrentPriceParams`:

- **currency**: which fiat currency (EUR) you want to have the exchange rate for.
- **@Optional amount**: the amount of Fiat currency (EUR) that you want to know the equivalent in BTC

Returns

A `CurrentPrice` object:

- **buyPrice**: Buy exchange rate
- **sellPrice**: Sell exchange rate
- **timestamp**: price timestamp

Code

Android

```
// Current price
CurrentPriceParams params = new CurrentPriceParams(Currency.EUR);

// Conversion of 50.000.000 satoshi (0,5 BTC) in euro
CurrentPriceParams params = new CurrentPriceParams(Currency.EUR, 50_000_000L)

conio.exchangeService.currentPrice(params, result->{
  result.analysis(price-> {
    // CurrentPrice
  }, error-> {
     // Exception
  });
});
```

iOS

```
// Current price
let params = CurrentPriceParams(currency: .eur)

// Conversion of 50.000.000 satoshi (0,5 BTC) in euro
let params = CurrentPriceParams(currency: .eur, satoshiAmount: 50_000_000)

conio.exchangeService.currentPrice(params: params) { result in
 result.analysis(ifSuccess: { prices in
      // CurrentPrice
    }, ifFailure: { error in
      // ServiceError
    })
});
```

Bitcoin hystorical price

You can get the Bitcoin hystorical price during a set amount of time.

Parameters

An object `HistoricalPriceParams`:

- **currency**: which fiat currency you want to have the exchange rate for.
- **startTimestamp**: Initial timestamp.
- **endTimestamp**: Final timespamp.
- **@Optional interval**: Time between each price (week, day, hour...)

Returns

Object `HistoricalPrices`:

- Contains a list of `CurrentPrice`
- An object `PriceAnalytics` containing:
- **deltaFiat**: absolute exchange rate change during the selected period.
- **deltaPercentage**: percentage change of the exchange rate during the selected period.
- **trend**: enum: price increased, decreased or stayed the same.

Code

Android

```
// Price from April 16, 2019 to April 16, 2018
// Standard Interval: 1 day
HistoricalPriceParams params = new HistoricalPriceParams(
  Currency.EUR,
  1523885446000L,
  1563465540000L
);

// Price from April 16, 2019 to April 16, 2018
// Selected Interval: 1 week
HistoricalPriceParams params = new HistoricalPriceParams(
  Currency.EUR,
  1523885446000L,
  1563465540000L,
  604800000
);

conio.exchangeService.historicalPrices(params, result->{
  result.analysis(prices-> {
    // HistoricalPrices
  }, error-> {
    // Exception
  });
});
```

iOS

```
// Price from April 16, 2019 to April 16, 2018
// Standard Interval: 1 day
let params = HistoricalPriceParams(currency: .eur, startTimestamp:
1523885446000, endTimestamp: 1563465540000)

// Price from April 16, 2019 to April 16, 2018
// Selected Interval: 1 week
let params = HistoricalPriceParams(currency: .eur, startTimestamp:
1523885446000, endTimestamp: 1563465540000, interval: 604800000)

conio.exchangeService.historicalPrices(params: params) { result in
 result.analysis(ifSuccess: { prices in
      // HistoricalPrices
    }, ifFailure: { error in
      // ServiceError
    })
});
```

Trading limits

Request user trading limits (assigned at signup)

Returns

An object `AllTradingLimits (Android)` or `Limits (iOS)` containing:

- Two objects: `TradingLimits`, one for buying limits and one for selling limits. Inside of it we get:
  - **currentLimit**: current limit
  - **limits**: a list containing each limit (daily, monthly, yearly) and their maximum values.
  - **currentLimitsByType**: current limit for each of the limits.
- **minimumBuyAmount**: minimum amount in fiat currency (EUR) required to buy Bitcoin
- **minimumSellAmount**: minimum amount in fiat currency (EUR) required to sell Bitcoin

Codice

Android

```
conio.exchangeService.tradingLimits(result -> {
  result.analysis(limits -> {
    // TradingLimits
  }, error -> {
    // Exception
  });
});
```

iOS

```
conio.exchangeService.tradingLimits { result in
 result.analysis(ifSuccess: { limits in
      // TradingLimits
    }, ifFailure: { error in
      // ServiceError
    })
});
```

Buy Bitcoin

In order to buy Bitcoin you will have to perform 2 operations. The first one creates a `Bid` (a request to buy some BTC at some price). Inside the created `Bid` you will find the `WiretransferInfo` that you will use to make the wire transfer necessary to pay for the Bitcoin. Once the wire transfer is completed you can perfom the second operation that will inform Conio that you payed the `Bid` by sending over a `CryptoProof`, required to proof the validity of the transaction.

Bid creation

You can create a `RequestBid` with a **currency** (BTC or EUR) and an amount, expressed either in satoshi, or in Fiat currency. For example you can create a `RequestBid` in Euro to buy an amount in Bitcoin for the equivalent of 20€, or a `RequestBid` in Euro to buy 100.000.000 satoshi.

If the request will be successful you will get a `Bid` containing an `ID`. With this identifier you will be able to **update** the Bid to get fresh info about it. This will be necessary if the user takes some time (more than 2 minutes) from the Bid request to the actual payment.

Parameters

- **(Optional) id**: Bid id, insert only if you need to refresh the bid
- **(one of) satoshi**: amount of Bitcoin that the user wants to buy
- **(one of) fiatAmount**: amount of Fiat currency the user wants to spend to buy an equivalent Bitcoin amount
- **currency**: Fiat currency used to buy (EUR)

The SDK will allow you to insert only one of **satoshi** or **fiatAmount**. You should never input both of them at the same time.

Returns

An object `CreatedBid` containing:

- **id**: identifier required to refresh or finalize a bid
- **currency**: Fiat currency used to buy (EUR)
- **satoshi**: Satoshi amount of the request
- **fiatAmount**: Fiat amount (EUR) of the request
- **serviceFees**: Fees for the transaction in the selected **currency**
- **expiration**: Bid expiration timestamp. If expired please refresh the Bid.
- **wiretransferInfo**: necessary info to pay for the bid

Code

Android

```
// Buy request for 100€
CreateOrRefreshBidParams params =
  new CreateOrRefreshBidParams(Currency.EUR, 100d);
```

```
// Buy request for 1.000.000 satoshi
CreateOrRefreshBidParams params =
  new CreateOrRefreshBidParams(Currency.EUR, 1000000001);

// Bid refresh for 100€
CreateOrRefreshBidParams params =
  new CreateOrRefreshBidParams(
    "bididentifier",
    Currency.EUR,
    100d
  );

conio.exchangeService.createOrRefreshBid(params, result -> {
  result.analysis(bid -> {
    // CreatedBid
  }, error -> {
    // Exception
  });
});
```

iOS

```
// Buy request for 50€
let params = CreateOrRefreshBidParams(currency: .eur, fiatAmount: 50.0)

// Buy request for 1.000.000 satoshi
 let params = CreateOrRefreshBidParams(currency: .eur, satoshi: 1000000)

// Bid refresh for 100€
let params = CreateOrRefreshBidParams(bidID: "bididentifier", currency: .eur,
fiatAmount: 100.0)

conio.exchangeService.createOrRefreshBid(params: params) { result in
 result.analysis(ifSuccess: { createdBid in
        // CreatedBid
    }, ifFailure: { error in
        // ServiceError
    })
});
```

Bid Payment

Once you have payed the `Bid` you can use the `Purchase` operation to receive the Bitcoin. You will have to submit a `CryptoProof`, that you can create in the same way as the one created during the signup. The only difference is the following DATA to concatenate (exactly in this order):

```
[proofID, "PAY_FOR_BID_WT", bidID, userID, Expiration]
```

Parameters

An object `PurchaseParams` containing:

- **bidId**: `Bid` identifier referring to the bid you want to finalize
- **cryptoRequest**: a `BidCryptoRequest`

Returns

An object `PurchaseResult` containing:

- **bidId**: `Bid` identifier

Errors

- INVALID_CRYPTO_PROOF Crypto proof is not valid
- INVALID_PAYMENT_METHOD Payment method is not valid
- UNSUPPORTED_PAYMENT_METHOD Payment method is not supported
- TRADING_LIMITS_EXCEEDED Bid exceed the maximum buy limit of the user
- TRADE_EXPIRED Bid is expired
- BID_ALREADY_PAID Bid was already paid
- BID_NOT_YET_PAID Bid has not been paid yet
- UNRECOVERABLE_BID Bid is in an error state
- FIAT_AMOUNT_TOO_LOW Fiat amount is lower than minimum limit

Code

Android

```
BidCryptoRequest bidCryptoRequest =
  createCryptoRequest() // Your implementation

PurchaseParams params =
  new PurchaseParams("bidId", bidCryptoRequest, card);

conio.exchangeService.purchase(params, result -> {
  result.analysis(bid -> {
    // PurchaseResult
  }, error -> {
    // Exception
  });
});
```

iOS

```
let cryptoRequest = createCryptoRequest() // Your implementation

let params = PurchaseParams(bidID: "bidID", paymentCard: card, cryptoRequest:
cryptoRequest)

conio.exchangeService.purchase(params: params) { result in
 result.analysis(ifSuccess: { bid in
    // PurchaseResult
    }, ifFailure: { error in
    // ServiceError
    })
});
```

Sell Bitcoin

In order to buy Bitcoin you will have to perform 2 operations. The first one creates a `Ask` (a request to sell some BTC at some price). The second one will pay said `Ask`, by using the ask identifier. The SDK will sign the Bitcoin transaction that moves the bitcoins from the user wallet, returning the id of said completed Ask.

Ask Creation

You can create a `CreatedAsk` in Euro to sell an amount in Bitcoin for the equivalent of 50€, or a `CreatedAsk` in Euro to sell 100.000.000 satoshi.

The request will return an `Ask` containing an `ID`. With this identifier you will be able to **update** the Ask to get fresh info about it. This will be necessary if the user takes some time (more than 2 minutes) from the Ask request to the actual sell.

Parameters

- **(Optional) id**: Ask id, insert only if you need to refresh the Ask
- **(one of) satoshi**: bitcoin amount the user wants to sell
- **(one of) fiatAmount**: amount of Fiat currency the user wants to receive when selling Bitcoin
- **currency**: Fiat currency to receive (EUR)

The SDK will allow you to insert only one of **satoshi** or **fiatAmount**. You should never input both of them at the same time.

Returns

An object `CreatedAsk` containing:

- **id**: identifier required to refresh or finalize a ask
- **currency**: Fiat currency to receive (EUR)
- **satoshi**: Satoshi amount of the request
- **fiatAmount**: Fiat amount (EUR) of the request
- **serviceFees**: Fees for the transaction in the selected **currency**
- **expiration**: Ask expiration timestamp. If expired please refresh the Ask.
- **minerFees**: Bitcoin network fees, used to pay for the inclusion of the transaction in the blockchain.

Errors

- TRADING_LIMITS_EXCEEDED
- NOT_ENOUGH_BTC_AMOUNT
- NO_SUCH_SELLER (Internal selling error)
- NO_SUCH_WALLET (Internal wallet error)

iOS

```
// Sell request for 50€
let params =
  CreateOrRefreshAskParams(currency: .eur, fiatAmount: 50.0)

// Sell request for 100000000 satoshi
let params =
  CreateOrRefreshAskParams(currency: .eur, satoshi: 100000000)

// Refresh ask
let params =
  CreateOrRefreshAskParams(askID: "id", currency: .eur, fiatAmount: 100.0)

conio.exchangeService.createOrRefreshAsk(params: params) { result in
  result.analysis(ifSuccess: { createdAsk in
    // CreatedBid
  }, ifFailure: { error in
    // ServiceError
  })
}
```

Android

```
// Sell request for 50€
CreateOrRefreshAskParams params =
  new CreateOrRefreshAskParams(Currency.EUR, 50d);

// Refresh ask
CreateOrRefreshAskParams params =
  new CreateOrRefreshAskParams("id", Currency.EUR, 50d);

conio.exchangeService.createOrRefreshAsk(params: params) { result in
 result.analysis(ifSuccess: { createdAsk in
        // CreatedAsk
    }, ifFailure: { error in
        // Exception
    })
});
```

Finalize Ask

To finalize the sell you just need to input the ID of the `CreatedAsk` in the `Sell` operation.

Parameters

An object `SellParams` containing:

- **askId**: `Ask` identifier

Returns

An object `SellResult` containing:

- **askId**: `Ask` identifier

Errors

- TRADING_LIMITS_EXCEEDED
- TRADE_EXPIRED
- UNRECOVERABLE_ASK
- ASK_ALREADY_PAID
- NOT_ENOUGH_BTC_AMOUNT_E
- DUST_ASK (Bitcoin amount is too low)
- FIAT_AMOUNT_TOO_LOW

Code

iOS

```
let params = SellParams(askID: askID)

conio.exchangeService.sell(params: params) { result in
  result.analysis(ifSuccess: { sellResult in
    // SellResult
  }, ifFailure: { error in
    // ServiceError
  })
}
```

Android

```
SellParams sellParams = new SellParams("askId");

conio.exchangeService.sell(params: params) { result in
 result.analysis(ifSuccess: { sellResult in
        // SellResult
    }, ifFailure: { error in
        // Exception
    })
});
```