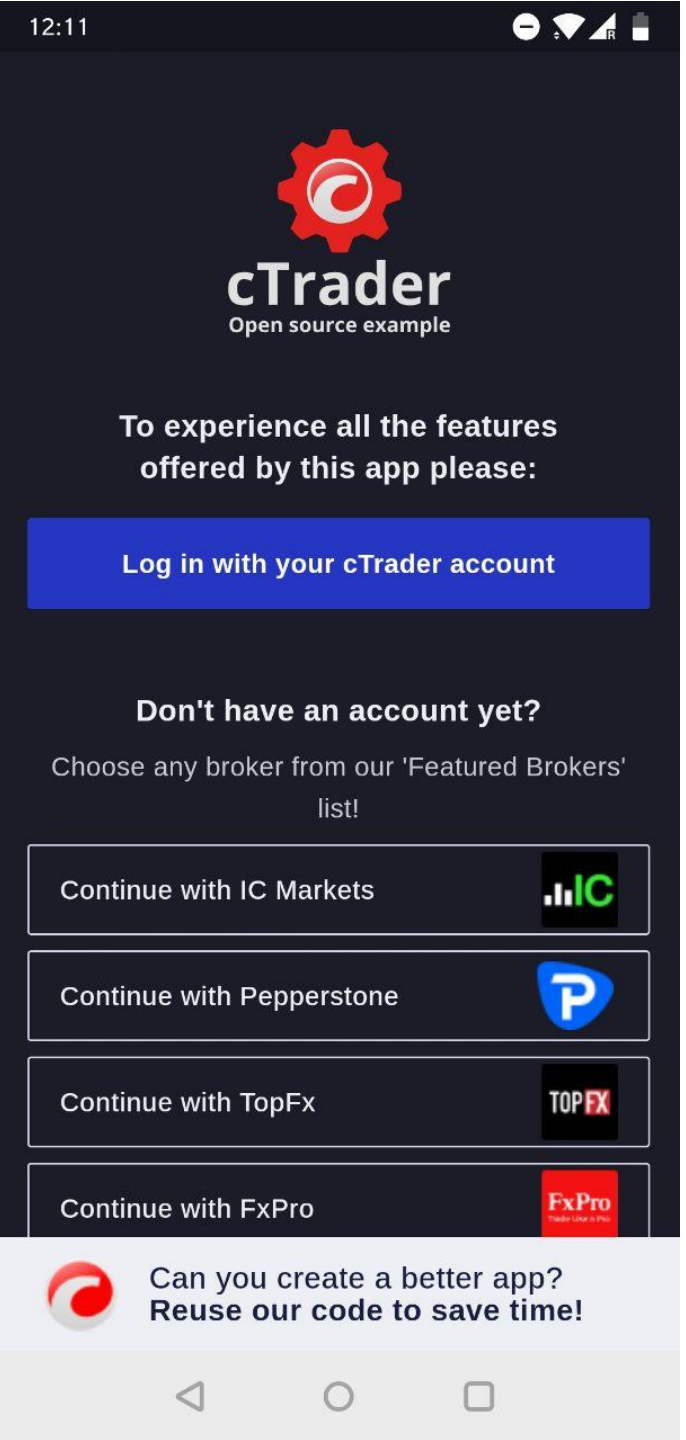


Authentication screen

1. Will have cTrader Open API Example app's logo
2. Below there will be promo text with short explanation
3. In the middle there will be a CTA button: "Log in with your cTrader account"
4. Below there will be a promo text for case when end user doesn't have cTrader account.
5. Below there will be chosen logos of cTrader partners; add appropriate links below:
6. IC: https://www.icmarkets.com/open-new-live-account/?camp=3263
7. Pep: https://track.pepperstonepartners.com/visit/?bta=35600&nci=5399
8. Top: https://signup.topfx.com.sc/Registration/Main/Account?dest=live&isSpecAts=true&camp=7087
9. FxPro: https://direct.fxpro.com.cy/partner/8033201
10. If user will tap on "Login" button (#3) – the next url will be activated through webview: https://id.ctrader.com/my/settings/openapi/grantingaccess/?client_id=[client_id]&redirect_uri=[redirect_uri]&scope=trading&product=web , while `scope=trading` provides access to trading.
11. Get `client id` from https://openapi.ctrader.com/apps > "view" credentials and `redirectUrl` from app's details, as it was inserted during registration.
12. If login is successful and access to some accounts provided, user will be redirected to 'redirect url', while `authorization code` will be added to that url as param, like in given example: https://my-redirect-uri.com/?code={authorization_code-code-will-be-here}
13. Authorization token must be exchanged in 60 secs to `access token` through following url: https://openapi.ctrader.com/apps/token?grant_type=authorization_code&code={authorization_code}&redirect_uri={redirect_uri}&client_id={your_app_client_id}&client_secret={your_app_client_secret} , with inserting the authorization code from #12, 'redirect url' and `client id` from #11 and `secret code` from "view" credentials, will appear below `client id`.

Authentication screen

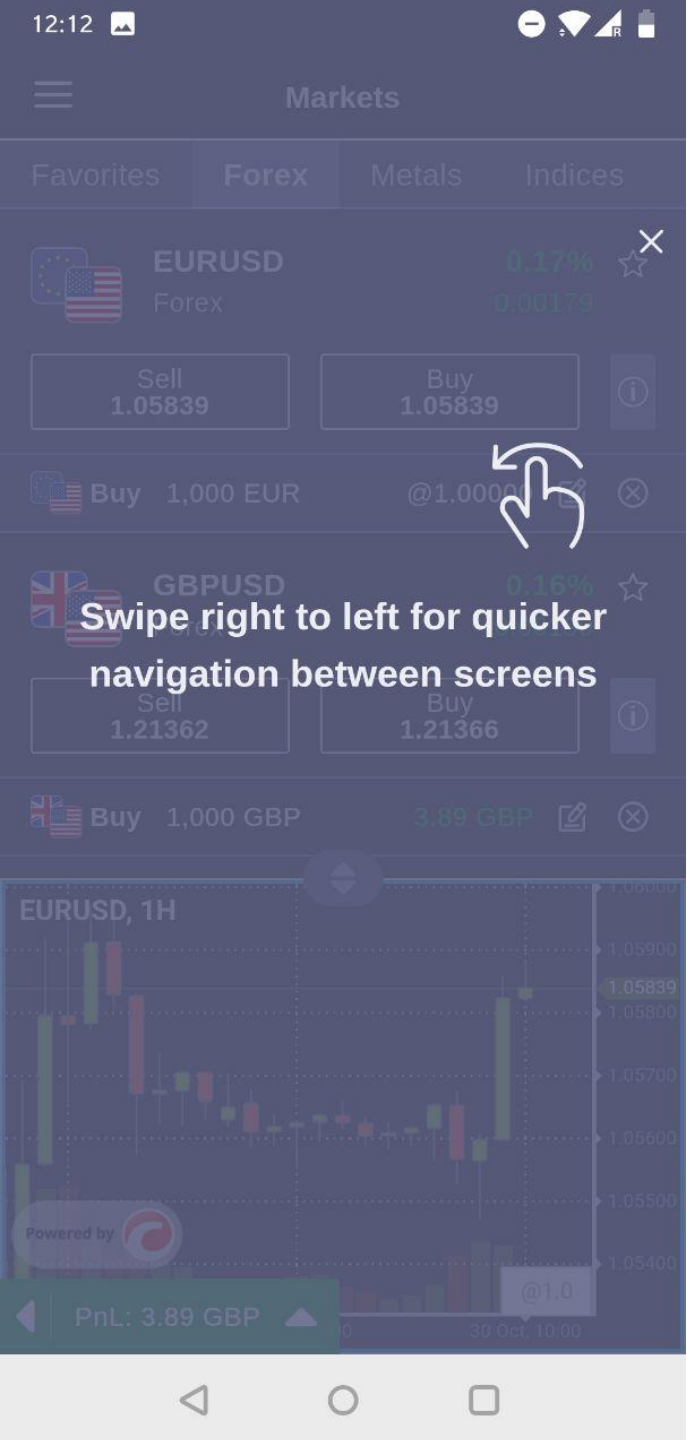
14. In response to #10 (prev. page) an access token will be sent. Will be provided with params: `**accessToken**` (access token string will be used for authentication on API messages), `**refreshToken**` (refresh token will be used for updating access token when it expired), `**expiresIn**` (the expiry time interval of access token in seconds), `**tokenType**` (the token type, REST standard which is usually Bearer), `**errorCode**` (in case of any error you will find the code here; "null" for success), `**description**` (the string description of error; "null" for success).
15. Keep `refreshToken` on client; if `access token` expires: using `refresh token` get a new **pair** of access &refresh tokens.
16. **App must establish a connection with Open API in order to allow trading for end user. Send `ProtoOAApplicationAuthReq` with `client id` and `client secret` (see previous page #13).**
17. General: note that in case no messages are sent to Open API ("subscribed" services, like `spotEvent` are not counted as "sent"; must send new requests) – application should use `ProtoHeartbeatEvent` in order to keep connection active.
18. Note, that connection establishing request must be sent to both `live` and `demo` endpoints: it is the easiest way to get an info regarding end user's accounts (live or demo; see #7). (!)`Live` account won't work on `demo` endpoint and vice versa.
19. Response will confirm successful connection to Open API endpoint/s.
20. When `access token` was accepted for given end user – user can access his trading data through the app. In order to get list of given user accounts (that he provided access to in #12 of previous page), send `ProtoOAGetAccountListByAccessTokenReq` (through both `live` and `demo` connections). All further requests-responses must keep #18 rule: for `live` accounts >> send requests to `live` endpoint and for `demo` - to `demo` endpoint.
21. `ctidTraderAccount` is the most important field, as it is a mandatory field in almost any other request related to trading.
22. Use `brokerTitleShort` field of `ProtoOACtidTraderAccount` entity to get broker's name of given account.

Loading trading data

1. **General: Please note, that sometimes names of fields/requests will appear without `ProtoOA` addition. If it wasn't mentioned otherwise – it will always meant to have "ProtoOA" in the beginning.**
2. As soon as `access token` was accepted for given user and at least one `ProtoOACtidTraderAccount` (that provides app with `ctidTraderAccountId`) was returned in `GetAccountListByAccessTokenRes` - user can be redirected to inner screens. **Please note** (might be useful): in REST protocols `ctidTraderAccountId` will appear as `accountId`.
3. To get data for any account > send `AccountAuthReq` with approp. `ctidTraderAccountId`. Response will confirm connection.
4. Send the `ProtoOATraderReq` for `ctidTraderAccountId` (if more than one `ctidTraderAccountId` returned – in any request before logging end user in - use the highest number account as a default) - in response you'll get data about trading conditions of given account. Later there will be usage of that data. If user has more than one account – send request for each of accounts he granted access to.
5. Please note, that if some account (**or all given user's accounts**) has `accessRights=3 (see below) – you'll not get a response for `TraderReq`; expect: "errorCode":"UNKNOWN_ERROR", "description":"Authentication failed" or RET_ACCOUNT_DISABLED (or can be a similar message). Then please show the following popup: header – "No access", body - "Your account has no access to live trading. Please contact your broker for details.". User **won't be allowed** to use the app.
6. Check `ProtoOATrader.accessRights` value. If user has 1 account and `access rights`>0, do the following:
7. For `access rights`=1, show user popup: header - "Limited Access"; body - "Your account has a limited access. You can't create new orders, but only close existing orders. Please contact your broker for more details." Below there will be 2 lines: "Your account…. `traderLogin`" and "Your broker…`brokerTitleShort`" (from prev. p. #22). On tap on OK button will go further. Later, show that popup on any "initiating new order" action tap.
8. For `access rights`=2, show the popup: header – "Limited Access"; body - "Your account has a "view only" access. Please contact your broker for more details.". Below show 2 lines with details from #7. On tap on OK button will go further. Later, show that popup **on any** tap on "Buy" or "Sell" buttons and "close" and "edit" icons.

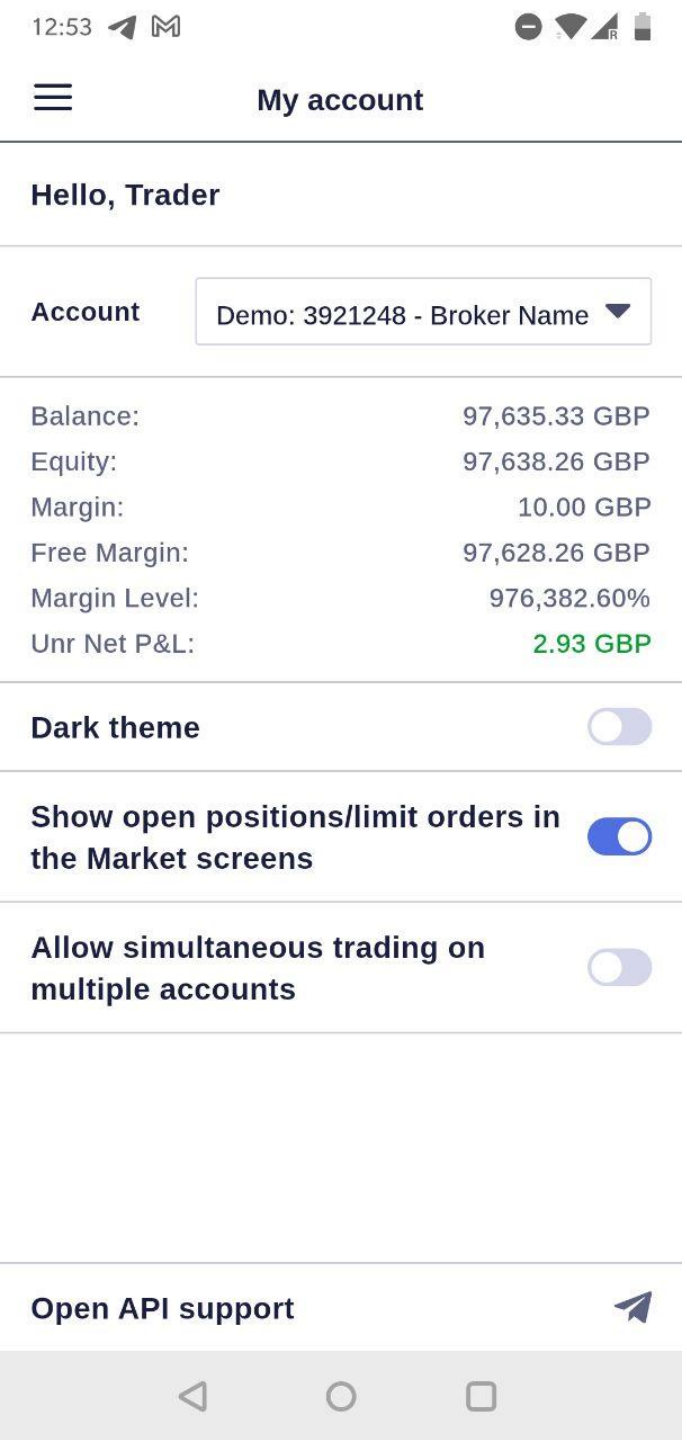
9. In (rare) case, when `access rights`<3, but `ProtoOATrader.accountType`=2 (SPREAD_BETTING), please show the next popup: header - "Spread Betting Account", body - "Your account is of SPREAD BETTING type. This app allows trading activities only for HEDGED or NETTED types of accounts. In order to continue, please open an appropriate account with one of our partners". Button "Ok" will redirect user to >> https://www.spotware.com/featured-ctrader-brokers . This account won't be able to use the app.

10. If user has more than 1 account, check all `ctidTraderAccountId` for `access rights`>=0. If there is at least one account with `access rights`=0 – use that account (always start with the highest `ctidTraderAccountId` if more than one have equal `access rights`), but keep all ctid 'access rights' in app's cache.
11. If none `access rights`=0, check one by one (from highest to lowest `ctidTraderAccountId`) what is the lowest possible `access rights`. If some accounts have the same `access rights` – go to the highest `ctidTraderAccountId`. Show user popups according to rules from #5-#8 from previous page and go to app's inner screens or return him to `authentication screen`.
12. If some account (for user **with more than one** account) with `access rights`<3 (according to aforementioned on previous page rules) logged into the app **for the 1st time**, show end user the following popup: "You've successfully logged in".
13. Below the header show the next 2 lines: "Your account…. `traderLogin`" and "Your broker…`brokerTitleShort`".
14. For user with 2+ accounts with `accessRights`<3 – show additional sentence below #12 details: " You can manage your accounts in app's "My account" tab".
15. In case there is at least one available for continued usage account, send `ProtoOAReconsileReq` with chosen `ctidTraderAccountId`. In response app will get all given account's positions (as list of `ProtoOAPosition` entities, if any) and pending orders (as list of `ProtoOAOrder` entities, if any). Later there will be usage of that data. Please note, that it is a valid case when no open positions or pending orders will return for certain account.
16. Despite app's home page will be "Markets", some data that will appear on "My account" screen must be available at any moment, so will start with it first.



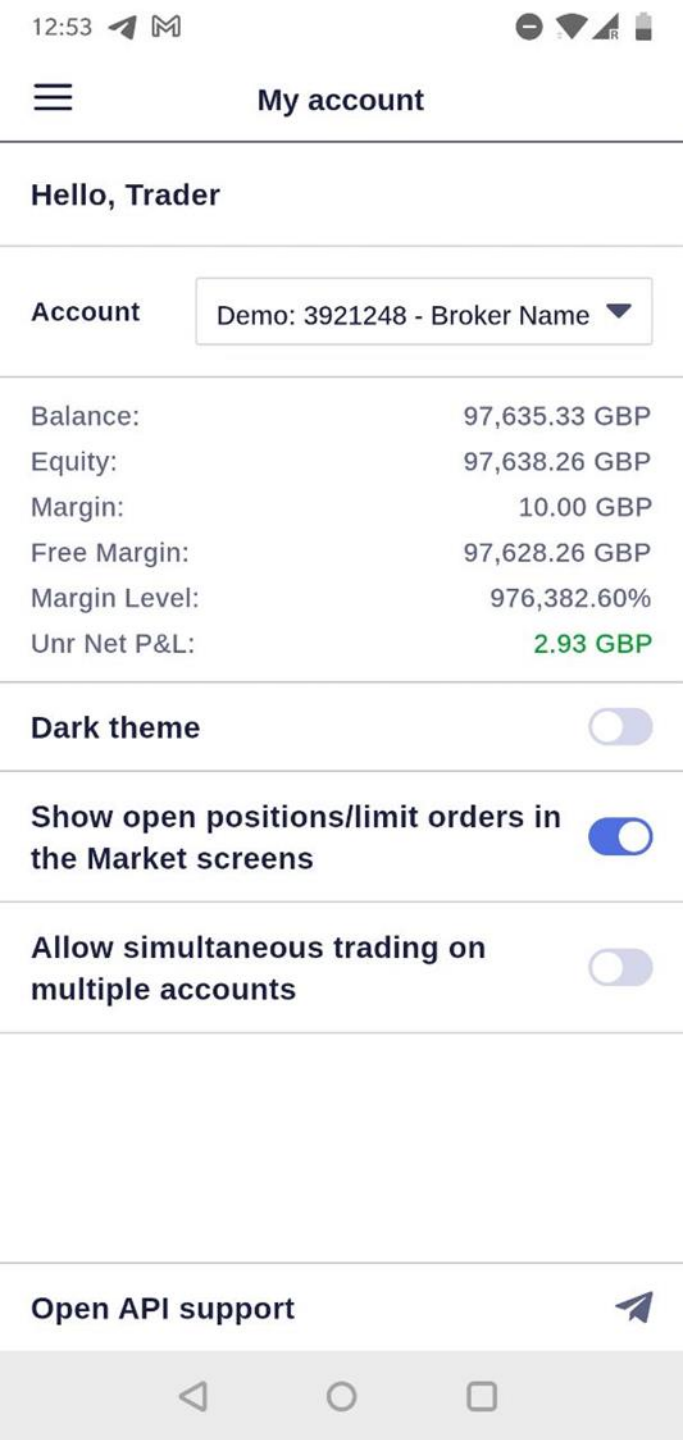
App's shortcuts and tooltips

1. In order to ease some actions, 3 shortcuts will be added to the app:
   a) Quick skipping between the pages by sideways swipe. User will be able to jump to the next page (Markets >> My Account >> My Activity) simply by swiping move.
   b) Opening full screen chart with menu by changing device's view from portrait to landscape. *Of course given device should have this option allowed in device's menu.
   c) Quick access to app's menu by swiping from left side of the screen to the right.
2. Since these options are helpful for end user and adding to positive user experience, user will get appropriate tooltips on app's 1st session.
3. As user granted access to some of the accounts, `access token` is issued and user lands on "Markets" screen – cover it with partial transparency grey background, when the following text and appropriate icon will appear: "Swipe right to left for quicker navigation between the screens". "X" icon will appear in upper right corner.
4. Tooltip will disappear either after tapping on "X" or swiping sideways.
5. After 2 seconds (if user tapped "X") or next time user will reach Markets screen (if swiped sideways and reached another screen) – show tooltip about chart's landscape mode. Cover with grey semitransparent background chart's area with: "Rotate phone to get full screen chart with menu", appropriate icon of suggested action and "X" icon in the upper right corner. Again, tapping "X" or rotating phone to landscape will delete the tooltip.
6. After 2 more seconds or when user get again to Markets screen – show 3rd tooltip. Full screen covered with grey background, "Swipe from the left edge to right for quicker access to app's menu", appropriate icon and "X" icon in the upper right corner.
7. Again, either tapping "X" or swiping from screen's left edge inside will delete the tooltip.

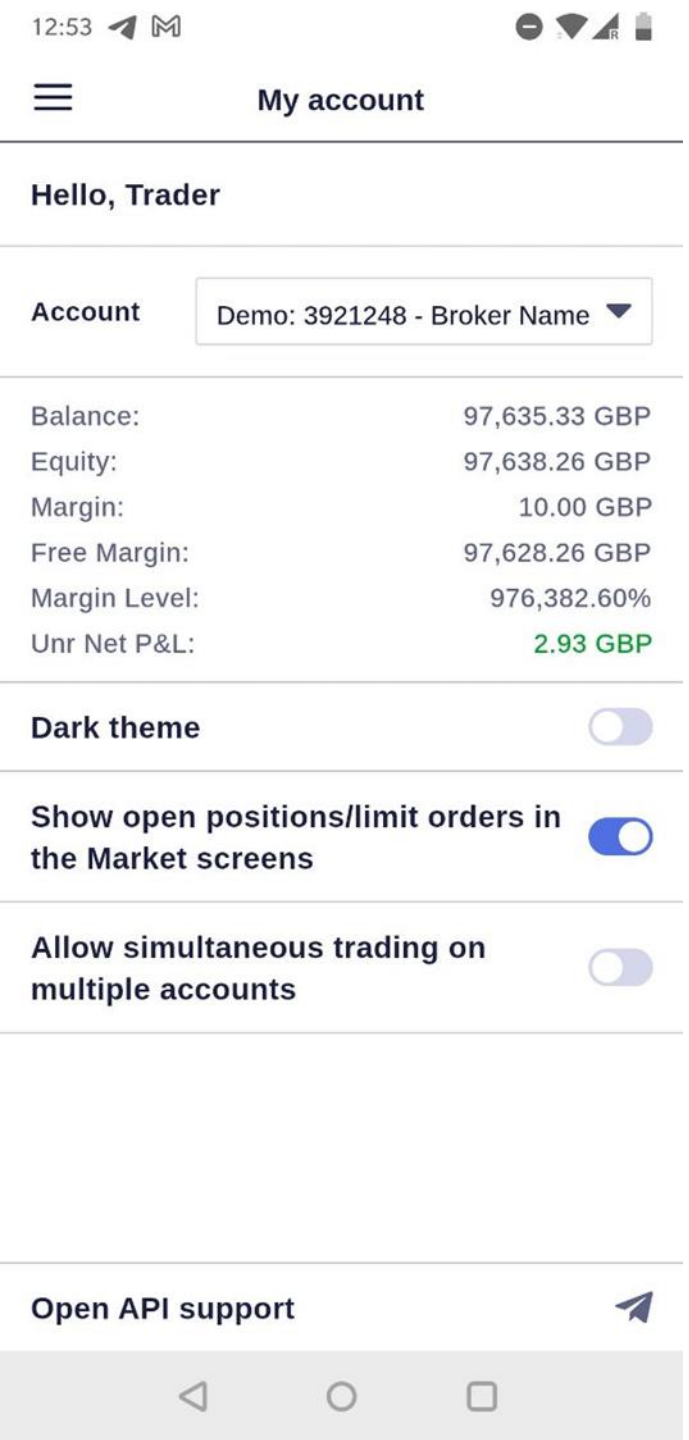


My account

1. Screen's header will be "My account"
2. Add: "Hello, Trader" below screen's header. Use it also in "App's Menu" (p. 12).
3. `Account choice` menu. In case some of these accounts has `access rights`>0, and user wants to switch accounts - show popups from ***Loading Trading Data*** #5 - #8 and switch account on tap on popup's OK. If some account got earlier "errorCode":"UNKNOWN_ERROR", "description":"Authentication failed" or RET_ACCOUNT_DISABLED (likely `accessRights`=3), exclude it from accounts list here (if needed, don't process its data during login process after getting "Authentication failed"). E.g. the list should act like it has (-1) accounts. If there is more than 1 account with "Authentication failed" – deduct all from accounts list on "My account" page.
4. In case some of user's accounts have failed authentication – on **first only** tap on `simultaneous trading` show user the next popup: header – "No access" and body - "[X] of your accounts has no access to live trading. Please contact your broker for details." Below list the account/s with failed authentication in same format as #7.
5. If user has only one account (`accessRights`<3) – that will be **the only choice** in accounts list.
6. Each account/s will appear with the following details: `account type` (`GetAccountListByAccessTokenRes` > `CtidTraderAccount.isLive` field. If true, `account type`="Live", if false > `account type="Demo") : `traderLogin` – `brokerTitleShort`.
7. So, example of user's account might be as following: [Live:1234567 – FX Pro ltd] or [Demo:987654 – Raw Trading ltd].



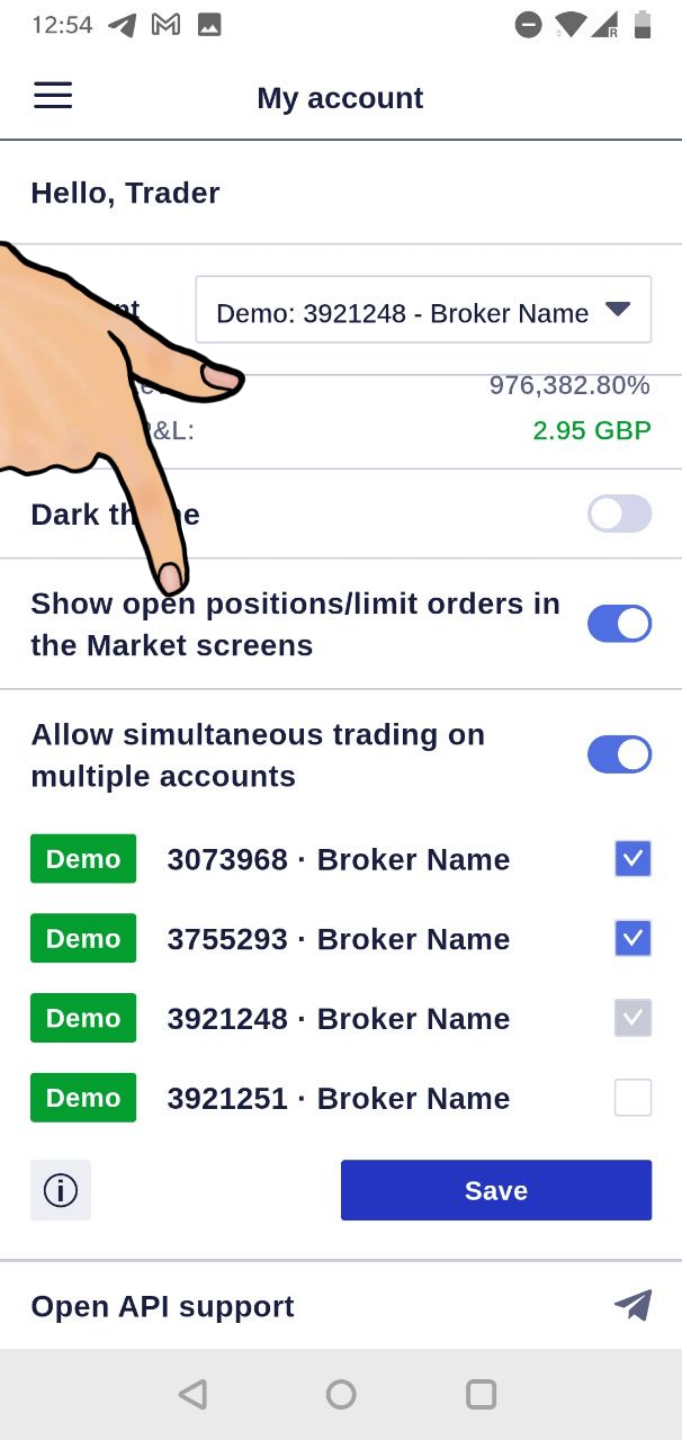
8. Below account/accounts choice, account's details will appear. These values are important, so its better to calculate them right at login and maintain live calculation during session.
9. All stats except "Margin level" (its % type of data) provide data in account deposit currency.
10. Send `ProtoOAAssetListReq` and by value of 'ProtoOATrader.depositAssetId ' find the appropriate asset. Get `Asset.name`, show it on left side (USD on example screen).
11. Insert 1 space between the digits and `depositAsset`, from here and on, anywhere `depositCurrency` is added to some numeric value.
12. Balance: will be provided in appropriate `ProtoOATrader.balance` field on app's launch.
13. After app's launch balance will be updated by `ProtoOAExecutionEvent` events. Changes in `balance` will be provided by:
14. Closed positions: `ProtoOAExecutionEvent` (of `executionType=3, 11) that it's 'Deal' entity includes 'ClosePositionDetail.balance' field. When new 'executionEvent' is accepted – check 'deal' entities that have 'closePositionDetail'=value (some 'deals' have 'close position detail'=empty) > get 'balance' with the highest `balanceVersion` > update the balance".
15. Through `ProtoOADepositWithdrawal` entities of `executionEvent` (executionType=10) >> `DepositWithdrawal.balance` fields. As with `closed positions`, check the `balanceVersion` field's value to know whether it’s the updated version. Update `balance`, if needed.
16. Save the last `balanceVersion` value in app's cache, so then a new version is accepted – verify that its number is higher for `balance` update, if yes - rewrite `balanceVersion`, too.
17. All 'balance' values must be divided by $10^{('moneyDigits')}$ (`ProtoOATrader.moneyDigits`).



18. Send `GetPositionUnrealizedPnLReq` (for appropriate `ctidTraderAccountId` - in response (repeated `PositionUnrealizedPnl` entities) gross and net unrealized Pnl per position will be returned. Cumulate `netUnrealizedPnl` of all positions – show it in "Unr Net P&L" field.
19. Send this request once a second in order to maintain adequate state of trader's account.
20. In 'ProtoOATraderRes' get 'managerBonus', 'ibBonus' and 'nonWithdrawableBonus', cumulate the values (even if all 3=0). Calculate as "0" if field=null or not returned. Then divide amount by $10^{('moneyDigits')}$ e.g. for USD ('moneyDigits'=2) total amount of bonuses will divided by 100.
21. Calculate ['balance' (value) + 'Unr. Net PnL' (value)]/ $10^{('moneyDigits')}$.
22. If result of #3 > #4, then Equity=[result of #4]*2.
23. If result of #3 < #4, then Equity=[result of #3]+[result of #4].
24. In order to calculate "Margin" properly – get `Trader.totalMarginCalculationType` and `Position.usedMargin' of all positions returned in `ReconsileReq'. Check next page for margin calculation details. Add 'Trader.depositAssetId' left of result.
25. **Note that** `executionEvent` with `executionType`=12 has values in `BonusDepositWithdraw` entity; it updates account's bonuses. It won't update `balance`, **but can** update `equity`. Get fields `managerDelta`+`ibDelta` to get new values, then recalculate `equity` (see #20).
26. In case some position will be closed through 'ProtoOAExecutionEvent' (event's `Deal` will have value in `closePositionDetail` field, like #14) – **recalculate** total `margin`.
27. Later other types/data provided of `executionEvent` will be discussed in details.
28. Calculate "Free margin" value as='equity' – 'margin'.
29. Calculate "Margin level" value as='equity' / 'margin' *100

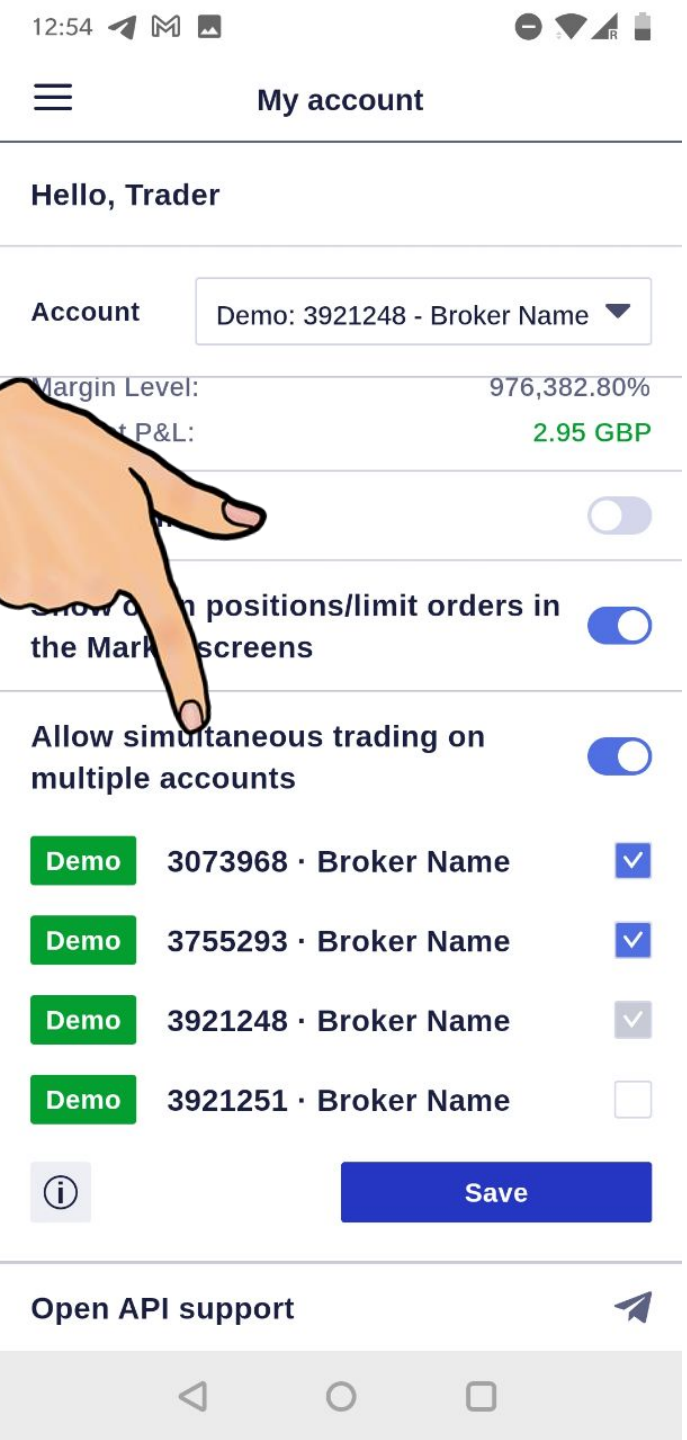
Exact Margin Calculation

1. There are 3 types of calculation that might be used for required margin: `MAX` (ENUM=0), `NET` (ENUM=1) and `SUM` (ENUM=2).
2. For each calculation all positions' `usedMargin` values should be considered.
3. For further explanation, let's assume there are 3 open positions for given account: P1=EURUSD Buy 10,000, P2=EURUSD Sell 20,000 and P3=GBPUSD Buy 30,000. `usedMargin` is P1=10$, P2=20$ and P3=45$.
4. Margin calculation type `SUM` - is generating all `usedMargin` values of all positions, e.g. in our example it will be 10+20+45=75$.
5. Margin calculation type of `MAX` and `NET` requires dividing margin per asset to 2 `tradeSide`: buy and sell.
6. So, the margin for GBPUSD is 45$ (since its only 1 position), then for EURUSD we have 20$ for sell position/s and 10$ for buy position/s.
7. If calculating `NET` margin (the difference between Sell and Buy margins for given symbol) the total margin amount will be: 45$ for GBPUSD and (20-10)=10$ for EURUSD, i.e. total `NET` margin will be 55$.
8. If calculating `MAX` margin (the biggest of Sell and Buy margins for given symbol), the biggest value of Buy/Sell side should be taken into consideration; in case of EURUSD it will be Sell side – 20$. Then, the total `MAX` margin will be 45$+20$ = 65$.
9. Suppose given trader opened 2 more positions: P4=EURUSD Sell 25,000 (`usedMargin`=25$) and P5=GBPUSD Sell 20,000 (`usedMargin`=30$).
10. Then, total `SUM` margin will be 10$+20$+45$+25$+30$=130$.
11. Total `NET` margin will be (25$+20$-10$)+(45$-30$)=50$
12. Total `MAX` margin will be 25$+20$+45$=90$
13. In rare cases, `Trader.isLimitedRisk` field will be TRUE, which can effect margin calculation type (LR traders will be discussed later, p. 44).
14. If `Trader.limitedRiskMarginCalculationStrategy`>0, expected margin will be calculated by different formula, which is also will be provided later (p. 44-45).

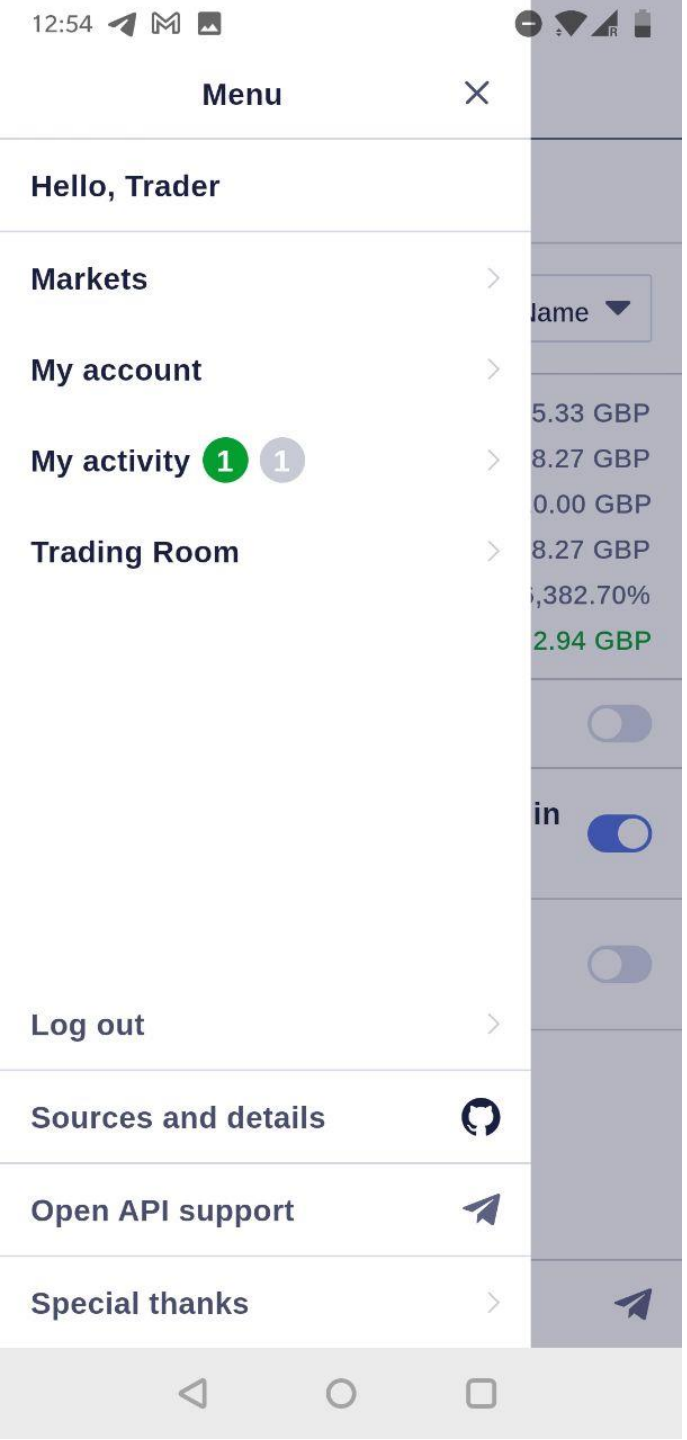


My account: advanced settings

1. Below account stats there is additional choices for end user. Currently there are 3 options.
2. "Dark theme" will change the color of app's background. Default – dark theme.
3. "Show open positions or/and pending orders in Markets screen" – is on "ON" as default. Will show/hide positions/orders for given symbol below 'symbol area' (see p. 30).
4. "Allow simultaneous trading on multiple accounts" is available for owners of 2+ accounts **with `access rights`=0**. Default: "OFF". Is visible for users with 1 account `accessRights=0`, but on tap show popup - header: "Limited trading access", body: "In order to activate the option of Simultaneous Trading, you should have at least 2 accounts with full trading access. You can open a new trading account with any broker from our 'Featured Brokers' list!" with "Cancel" and "Ok" buttons. "Cancel" closes popup, "OK" will open link.
5. If user (2+ full trading accounts) taps radio button: list of all accounts that appear in `accounts choice menu` AND have `accessRights`=0 is opened below. Default: currently chosen account is tapped and disabled.
6. A tooltip with appropriate explanation will appear below on the left side and "Save" button will be below on the right side of `simultaneous trading area`.
7. Tooltip text: "Any orders created on active account will be sent to other accounts that you've chosen in this list. In case of rejection or cancelation you'll get notified with popups."
8. Tapped "Save" button keeps the choice, collapsing 'accounts choice area'. If more than 1 account was chosen, after tap on "Save" will collapse the list and add a new line "Opened for [no. of chosen accounts] accounts" below. Edit icon will reopen accounts list.
9. If 1 account was selected – please show this popup >> H: Incorrect choice", B: "In order to activate the option of simultaneous trading, please choose at least 2 accounts."



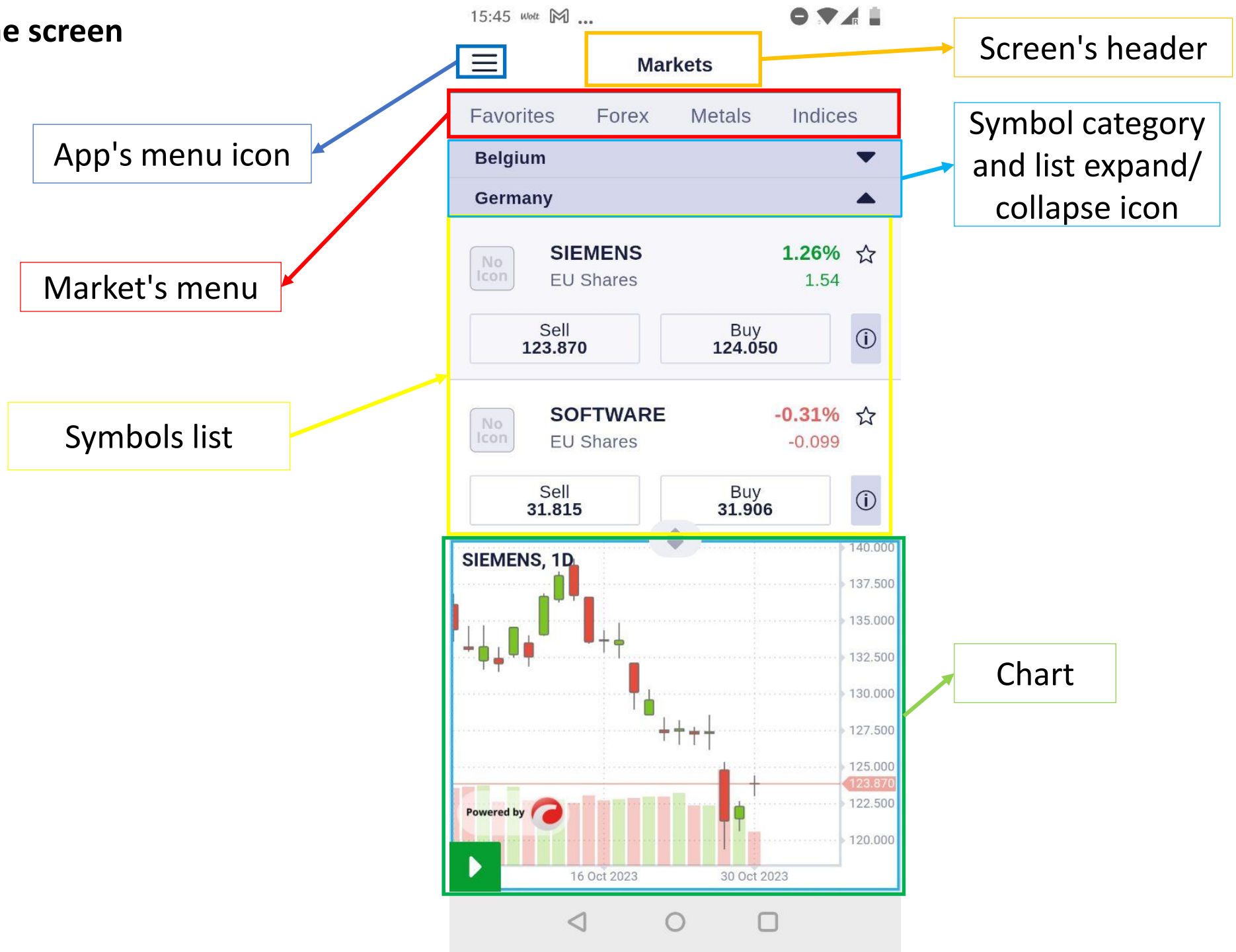
9. If user with 2+ account with `accessRights`=0, but also has accounts with `accessRights`=1 or 2 AND one of these is chosen – on tap on "simultaneous trading" do **not** open accounts list, but show a popup: header "Please change account", body: "This account is not suitable for Simultaneous Trading. Please choose account with full access to trading".
10. If ST is activated for some accounts with `accessRights`=0, but user is switches to account with `accessRights=1 or 2 and then taps on "edit" on ST – show the popup from #9.
11. If user opened a list of accounts in simultaneous trading and then tapped on `accounts choice menu` - collapse list in simultaneous trading and turn off ST button w\o saving any.
12. If active, tapping radio button again will **immediately** disable simultaneous trading.
13. After #12 – check whether user has any linked orders (more details p. 75-77) or/and positions. If no – disable "Simultaneous trading" option, if yes – show the next popup: header – "Linked orders", body: "You still have linked positions or pending orders from previous simultaneous trading activity. If you disable this option – all links will be deleted. Are you sure you want to continue?"
14. Show buttons Yes and No; on "Yes" – erase links between positions/pending orders and disable simultaneous trading, on "No" – close the popup and leave ST option active.
15. If user disables one of previously chosen accounts – check whether there are any linked orders/positions. If no – make the change on "Save" tap, if yes – show the next popup: header – "Linked orders", body: "This account still has linked positions or pending orders from previous simultaneous trading activity. If you disable this option – all links will be deleted. Are you sure you want to continue?"
16. Show buttons Yes and No; on "Yes" – unlink this account's activity from related ST orders/positions, on "No" – close the popup w\o any other action.



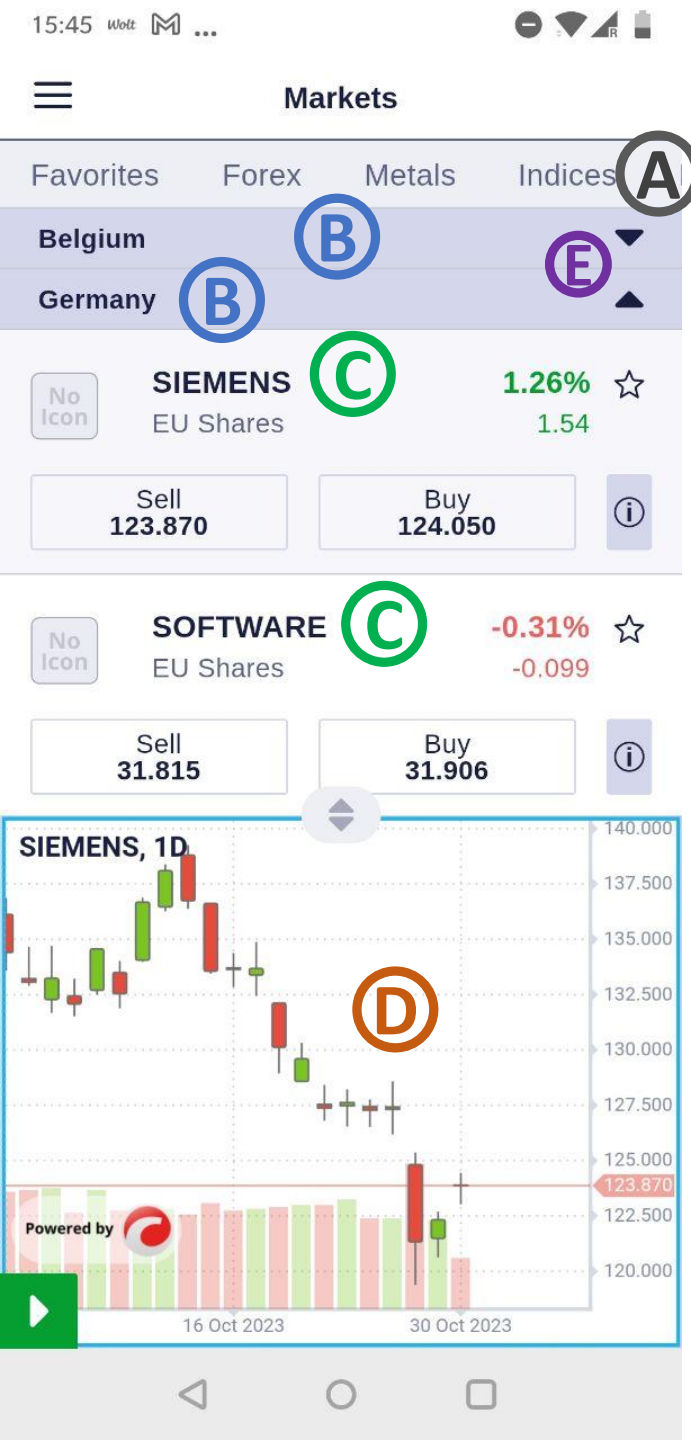
App's menu

1. In app's menu, the following options will be available:
2. Untapable upper option will provide some greeting see "My account" #2.
3. "Markets" option will lead to 'markets' screen on tap.
4. "My account" option will lead to 'My account' screen on tap.
5. "My activity" screen will lead to 'My activity' (default: positions) on tap. If there are open positions, their number will appear in green/red circle (depending on total Net PnL); same with pending orders – number in grey circle. Support number up to 3 digits in each circle.
6. Trading Room – will open 3rd party technical analysis signals in webview.
7. "Log out" must delete current access token, so user will have to authenticate again next session.
8. Sources and Details – will lead to official cTrader github resource, where files of this app will be stored (and available for downloads).
9. Open API support: please open the link - https://t.me/ctrader_open_api_support
10. Special thanks – will contain references to flutter creators, that their code was used in development of this app.

Structure of the home screen



Structure of the home screen (Markets)



1. Each screen in the app will have a header.
2. Header will include screen's name and app's menu icon (will open app's menu; see p. 12).
3. On Markets screen, below the header `Markets menu` will appear. (A)
4. 'Markets menu` will be scrollable sideways: options will be swiped left or right while the order will be according to `assetClass.sortingNumber`, while the smallest will be on the left edge of the swipe. Tapped option will be the active one, while the 1st choice from the left that is not `favorites` will be default choice on app's launch.
5. Below `Markets menu` there will be `symbols list` and it will include (optionally) `symbol categories` (B) and (mandatory)`symbols` (C).
6. Each `symbols list` will have expand/collapse icon on its right side (E). Tapping will hide all symbols belonging to given category. Tapping 2nd time will open the list of given `symbol category`. On each type the icon will appropriately change direction.
7. `Symbols list` will be scrollable between its boundaries: `markets menu` on top, `chart` in the bottom).
8. The order of `symbol categories` (if there will be more than 1) and symbols in each list will also be arranged by `symbolCategory.sortingNumber` and `LightSymbol.sortingNumber`.
9. Below the `symbols list` area the `chart` (D) will be placed (more details **will be provided later**).
10. `Chart` area will be "sticky" to the bottom of Markets page.

Requests

`ProtoOASymbolsListReq`
`ProtoOASymbolCategoryListReq`
`ProtoOAAssetClassListReq`

Response

`ProtoOASymbolsListRes`
`ProtoOASymbolCategoryListRes`
`ProtoOAAssetClassListRes`

Entities in response

`ProtoOALightSymbol`

Field in entity

`ProtoOALightSymbol.symbolCategoryId`

Finding appropriate entity in `ProtoOASymbolCategoryListRes`

`ProtoOASymbolCategory`

Field in entity

`ProtoOASymbolCategory.assetClassId`

Relating to appropriate entity in `ProtoOAAssetClassListRes'

`ProtoOAAssetClass`

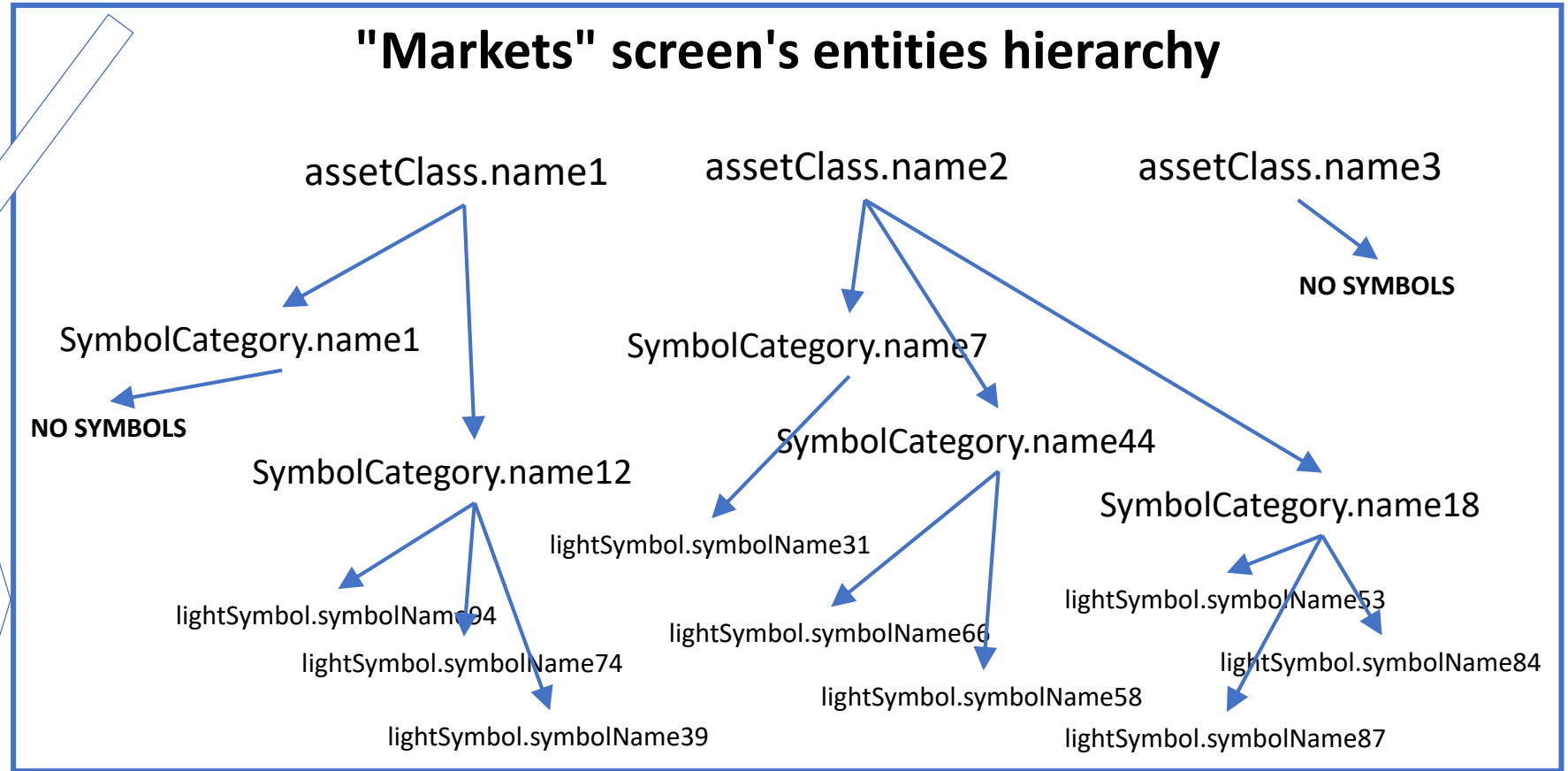
Getting

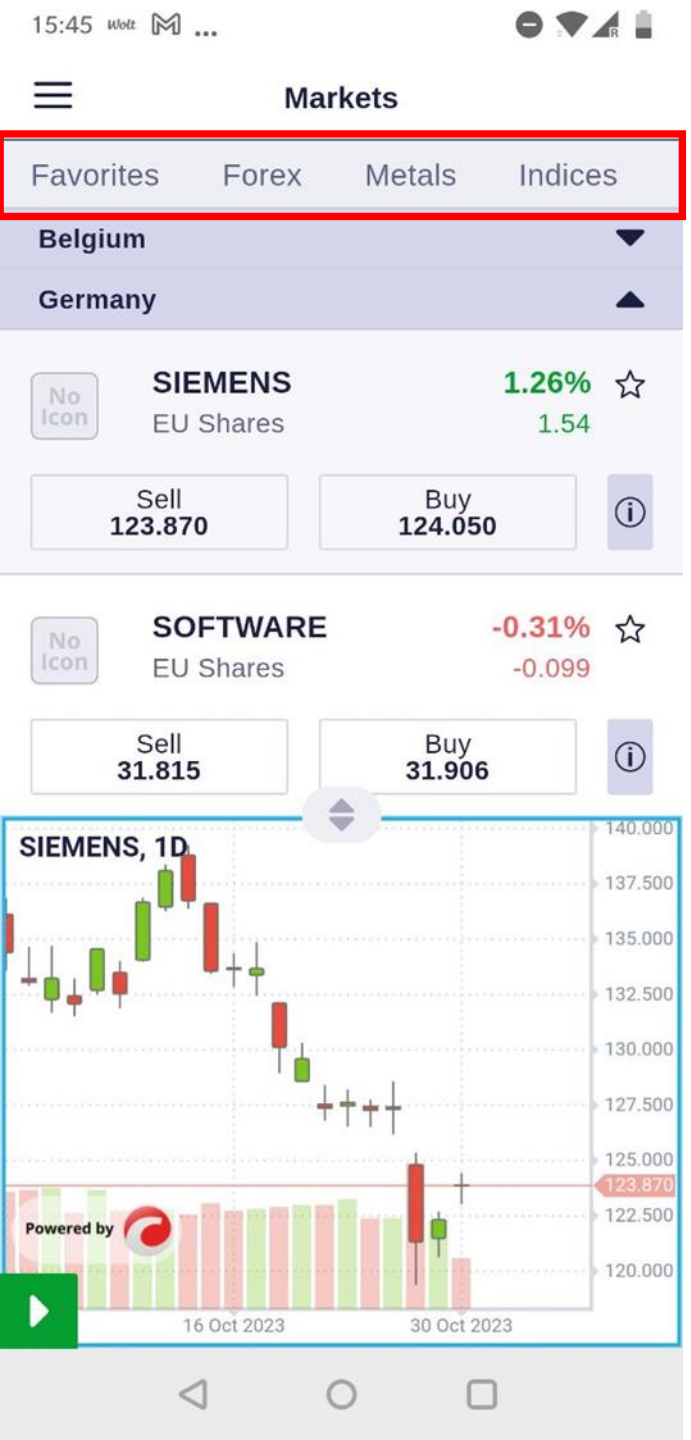
`ProtoOASymbolCategory.name`

Getting

`ProtoOAAssetClass.name`

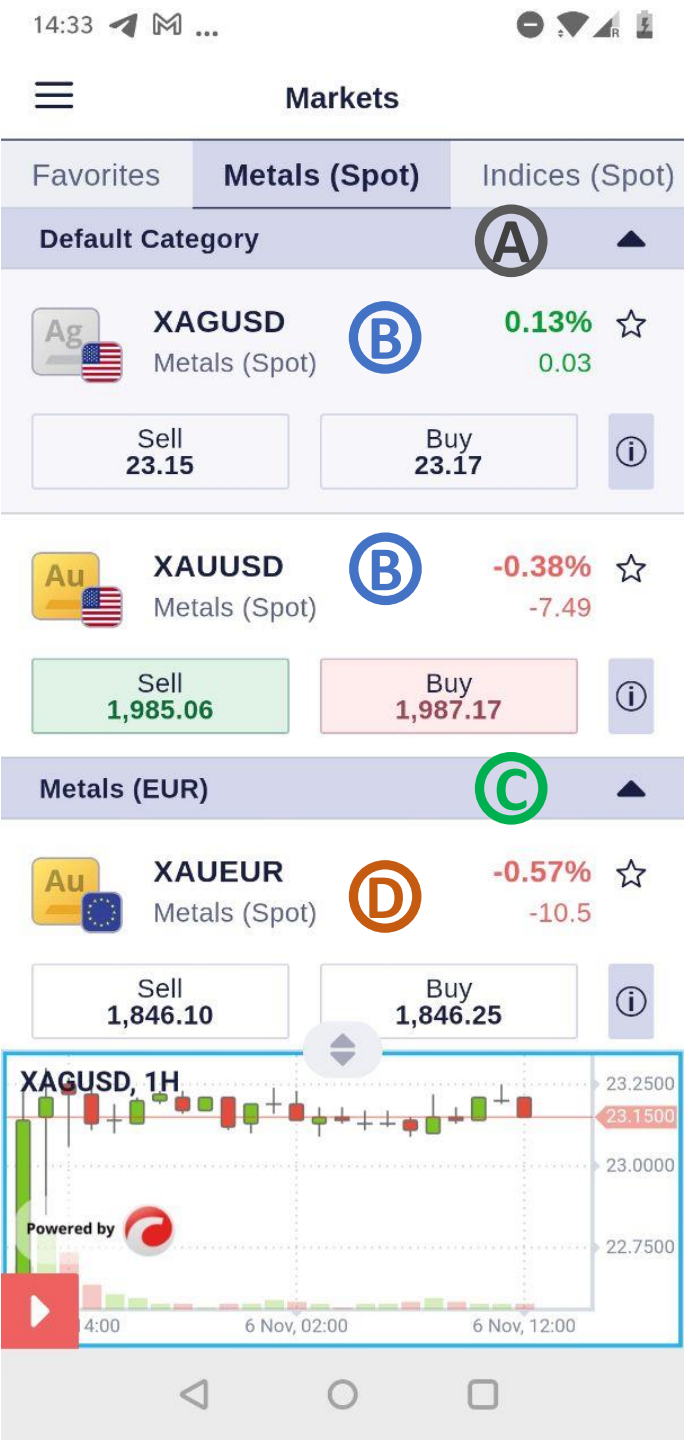
"Markets" screen's entities hierarchy



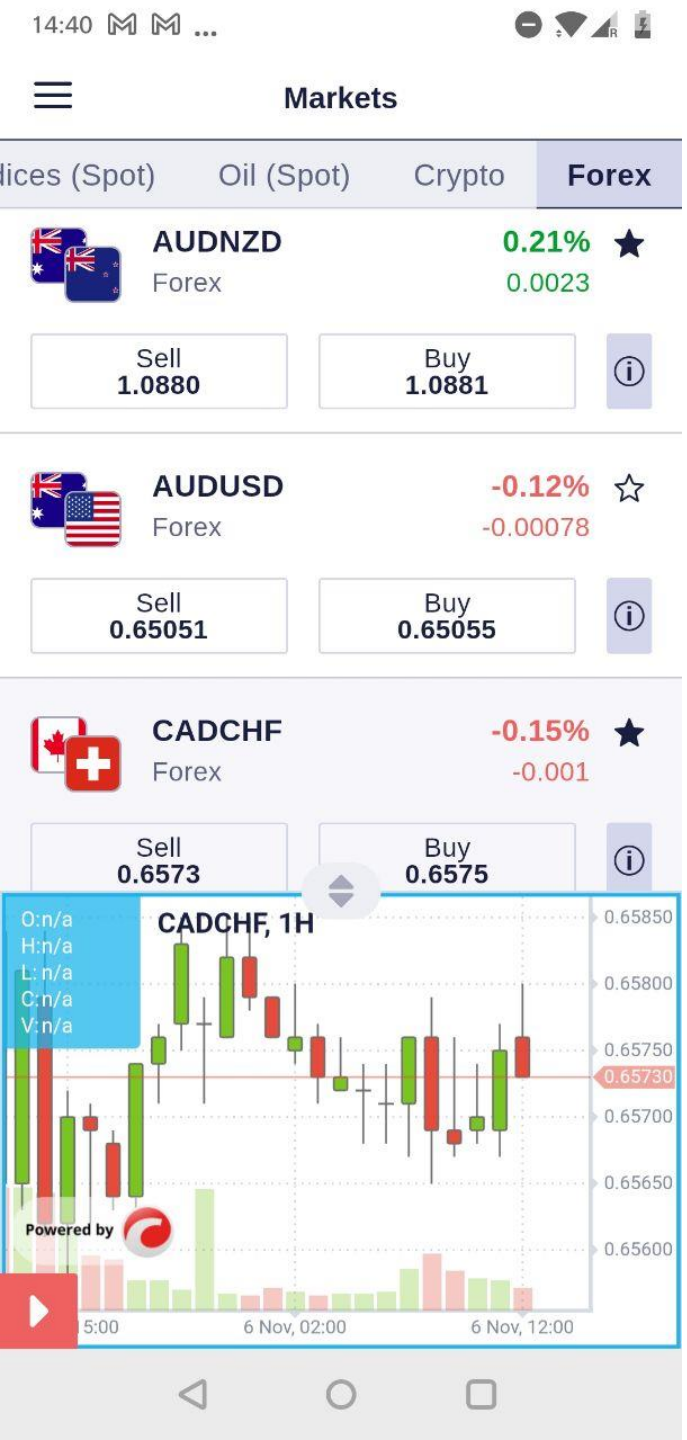


Getting symbols list and arranging it (Markets)

1. As every "logged in user" session will be initiated, send `ProtoOASymbolsListReq`, `SymbolCategoryListReq` and `AssetClassListReq` for given `ctidTraderAccountId`.
2. `ProtoOASymbolsListRes` will provide 2 types of repeated entities: `symbol` and `archivedSymbol`; **ignore** `archivedSymbol` entities (if any). Then filter symbols with `LightSymbol.enabled=FALSE; only =TRUE will reach `markets` screen.
3. There will be 3 levels in markets hierarchy: asset class >> symbol category >> symbol.
4. Asset classes will be shown on Markets screen as values of `AssetClass.name` (entity from `AssetClassListRes`), while each name is different choice option in `markets menu`.
5. For example "Forex" and "Metals" are 2 typical `names` of `ProtoOAAssetClasses`.
6. **Always** add another choice option to `markets menu` - `Favorites`, despite it won't be in response from #4. `Favorites` will always be far left option of `Markets menu` swipe.
7. If on given session start there are any symbols in `favorites` - make it tapped, if no - open next to right option of `markets menu`. As new `AssetClass.name` is tapped - upload new set of related `symbol categories` and `symbols`.
8. Each `AssetClass` entity will also have a field named `sortingNumber`. The values of these fields set the order of all asset classes in `markets menu`.
9. Considering #7, `AssetClass.name` with the lowest `sortingNumber` will be right to `favorites` and be opened if `favorites` is empty (like 1st session). Asset classes with ascending `sorting Numbers` will be placed right, till the highest: the far right option of `markets menu` swipe.
10. If any asset class <u>except favorites</u> will not have any related symbols – do not show it in `markets menu`. More details about favorites on p. 20.



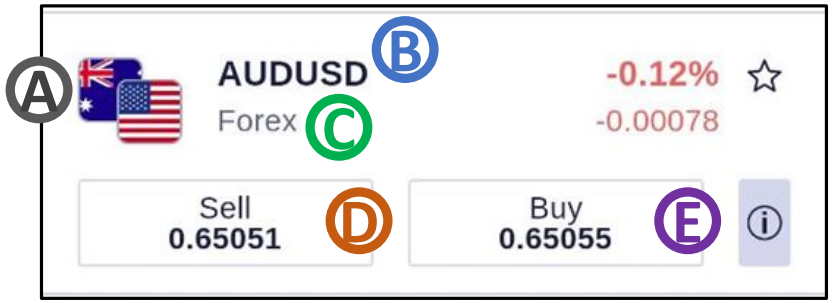
11. Each `ProtoOAAssetClass.name` will have 1 or more `SymbolCategory.name` (field in `SymbolCategory`, which is an entity in `SymbolCategoryListRes`) related to 'asset class'. All `symbol categories` will also have `SymbolCategory.sortingNumber`.
12. `ProtoOASymbolCategory.sortingNumber` of categories related to given `asset class` will help setting the order of `symbol categories` in the list. That will be helpful in case there are 2 or more categories with (at least) 1 related symbol in given `asset class`.
13. Any `symbol category` with no symbols must be **invisible and ignored** for any further action.
14. In case there is only 1 `symbol category` related to given `asset class` with 1 or more related symbols in it – ignore/don't show its `ProtoOASymbolCategory.name`, but place the related symbols in the `symbols list` in given `asset class`.
15. In case there is more than 1 `symbol category` with at least 1 related symbol – set the order of `symbols categories` by `SymbolCategory.sortingNumber`.
16. The `SymbolCategory.name` of related `symbol category` with the lowest `sortingNumber` will be placed as sub header on top of `symbols list`, just below `markets menu` >> see Ⓐ
17. Below `ProtoOASymbolCategory.name` – place all symbols Ⓑ related to that `symbol category` (more details on the next page).
18. Below the last symbol of `symbol category` with the lowest `sorting number`, place new sub header with `ProtoOASymbolCategory.name` of `symbol category` with 2nd lowest `sorting number` Ⓒ and related symbols Ⓓ.
19. Continue the same way with other `symbol categories` related to given `asset class` till the last related symbol is placed in the `symbols list`.
20. `Favorites` `asset class` will not have any `symbol categories` by definition.



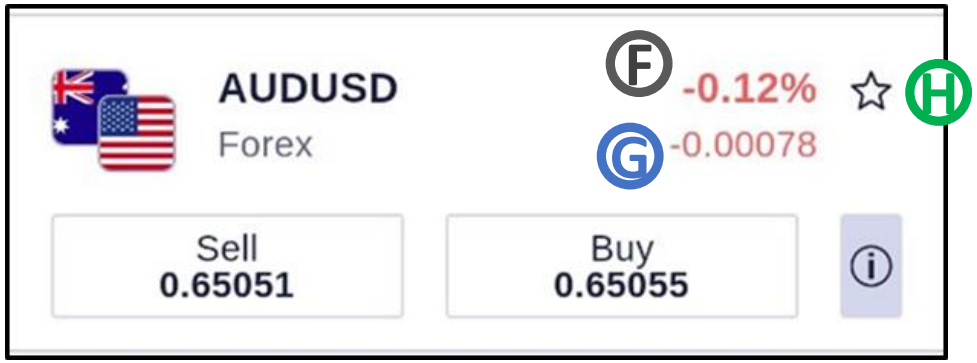
21. As it was mentioned on the previous page, `SymbolCategory.name` will have from 0 to theor. infinite symbols related.
22. Also, as it goes with other entities, symbols order will be set by each symbol's `ProtoOASymbol.sortingNumber`. The smaller the `sorting number` value – the higher given `ProtoOASymbol.name` will be placed in the list for given `symbol category`.
23. If there is more than 1 `SymbolCategory.name` in given `assetClass` - `symbolCategory` tabs will have expand/collapse icon. It will be placed on the right side of the tab.
24. By default all `symbolCategory` lists will be expanded.
25. On tap on expand/collapse icon all the symbols of given `symbolCategory` will collapse, so the next `symbolCategory.name` will appear in the next line.
26. When list of symbols will be collapsed – icon will change its direction respectively.

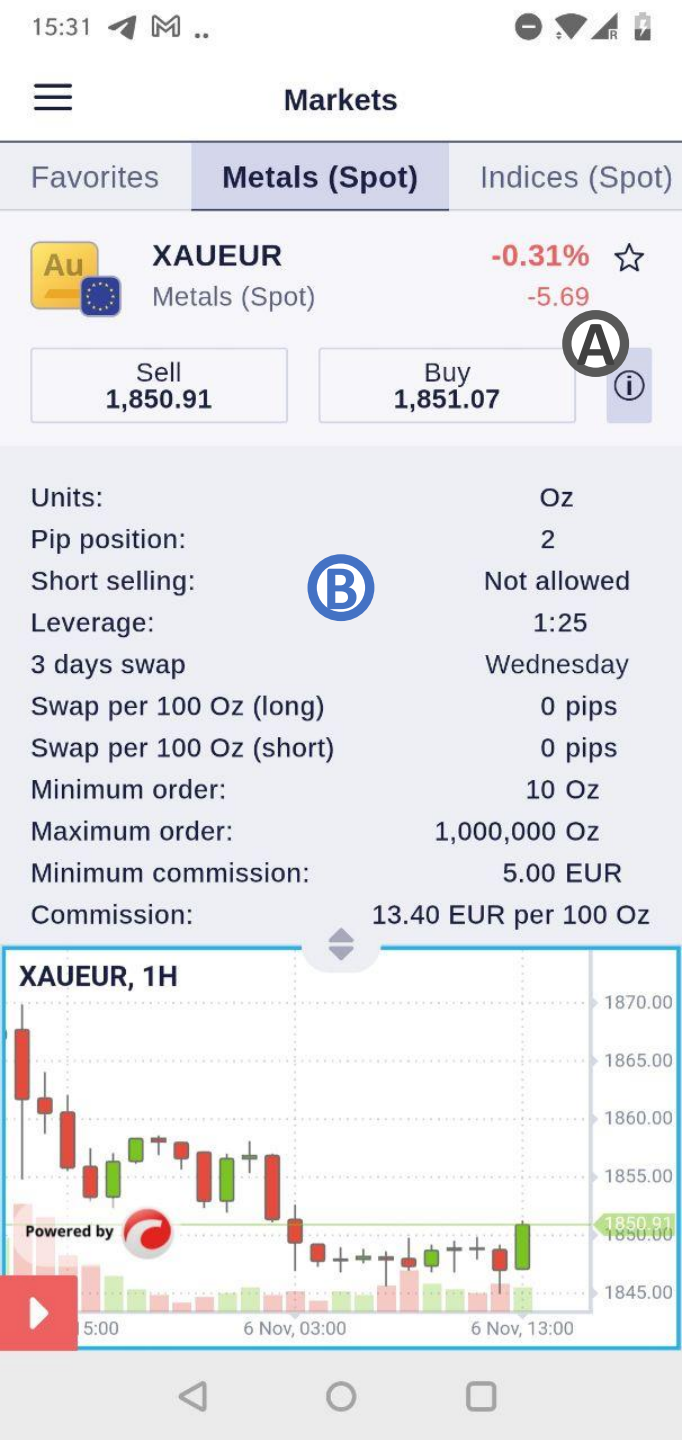
`Symbol area` entities and behavior

1. Every symbol and all related to it data will appear in `symbol area` interface. Every new session top `symbol area` background in default asset class will be darkened (in light theme; and vice versa in dark theme). This way active symbol will be shown. Tap on other `symbol area` will darken newly tapped `symbol area` and make the default `symbol area` color lighter again.
2. As **any symbol becomes active** (default on app's start/later on tap) – activate chart for that symbol (more details will be provided).
3. Add `symbol icon`Ⓐ in left corner of each `symbol area` (more details on icons matching on the last page of presentation).
4. Right to icon add `ProtoOALightSymbol.symbolName` Ⓑ and below it - the appropriate `ProtoOAAssetClass.name`Ⓒ.
5. At **any moment** that Markets screen is opened – determine the `LightSymbol.symbolId` of `symbol areas` that are visible to end user and send `ProtoOASubscribeSpotsReq` for these `symbol id`. Send all in one request , since there is a limitation of requests per second in Open API. Verify once in XXX miliseconds the list of "visible" symbols and update subscription to spots, if needed.
6. After subscription will be successful - `ProtoOASpotEvent` for each symbol will start coming with every new change of rate (relate to `symbol` by`ProtoOASpotEvent.symbolId`).
7. Below icon, symbol name and category, there will be 2 buttons, "Sell" Ⓓ and "Buy" Ⓔ
8. Every `ProtoOASpotEvent.bid` rate show in `sell button` under the word "Sell". Update its value with every new `ProtoOASpotEvent`. Every `ProtoOASpotEvent.ask` rate show respectively under "Buy" in `buy button`. Every new bid or ask should activate color effect: if rate is bigger than previous – green blink, if smaller – red blink (design will be provided).
9. Keep rates format according to `Symbol.digits`, so if for given symbol digits=3, and the last `bid`/`ask` is 12.4 >> show 12.400 (completing with zeros till `digits` number).



10. Right to `symbol name` a daily change in -% will appear (F). It will be shown in format of *.XX% and will be calculated by the next formula: (`ProtoOASpotEvent.bid` - `ProtoOASpotEvent.sessionClose`)/`ProtoOASpotEvent.sessionClose` *100
    Example: session close=100.00, bid=100.57 >> daily change will be (100.57 – 100.00)/100.00 * 100 >> 0.57%
11. If `daily change` > 0 – color it with green, if <0 – with red and if=0 – with grey.
12. Below daily change in-%, add an absolute daily change; it will be equal to `bid` - `sessionClose`; color: according to #2 (G).
13. Right to `daily change` - `favorites icon` will appear (H). It will have 2 states: tapped and untapped, when tapped will be filled with color (see screenshot on p. 18). By default (1st session) all `favorites icons` will be in untapped state.
14. Tapped/untapped symbols data will be kept **in app's cache**, while all symbols with tapped `favorites icon` will appear **also** in `favorite symbols list`. Tapping the icon again will change its state to untapped and if symbol will be at this moment in `favorite symbols list` - it will disappear from that list after tap.
15. As it was mentioned earlier (Getting symbols list and arranging it (Markets #5, #20**)** "favorites" is the only `asset class` that will be shown in `symbols list` even w\o symbols. When "favorites" will not have any symbols (at least during 1st session of every new user) – show below `markets menu` the next text: "You still don't have any symbols in your "favorites" list. Tap a star icon on any symbol to add it to "favorites"."



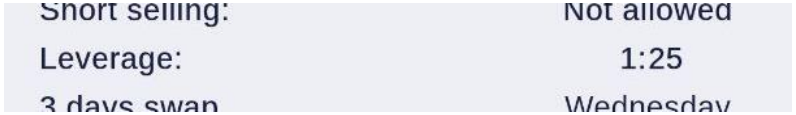


16. When user will tap on `symbol: more details` icon Ⓐ more details for given `LightSymbol.symbolName` will be shown. The `symbol area` will expand down and all new details will appear there.
17. On tap on `symbol: more details` icon – send `ProtoOASymbolByIdReq` with appropriate `ProtoOALightSymbol.symbolId`.
18. Show the next fields in expanded area below Ⓑ, while reversing camel case format of field names and adding it on the left side of each line. So, `ProtoOASymbol.pipPosition` - add on the left side as expanded area as "Pip position".
19. If a different text should be shown (not field name), it will appear in brackets and quotes.
20. Arrange the next `ProtoOASymbol` fields (from top to bottom) in "more details" area:
    `measurement units` ("Units"),
    `pipPosition`,
    `enableShortSelling` (**only** if =false; "Short selling"),
    `leverageId` ("Leverage"),
    `swapRollover3Days` ("3 days swap"),
    `swapLong` ("Swap per `Symbol.lotSize`/100 `Symbol.measurementUnits` (long)"),
    `swapShort`, ("Swap per `Symbol.lotSize`/100 `Symbol.measurementUnits` (short)"),
    `minVolume` ("Minimum order"),
    `maxVolume` ("Maximum order"),
    `minCommission` (**only if=value**, ignore if empty, `null`, etc.; "Minimum commission"),
    `commission`,
    `schedule`
    and `holiday` ("Next Holiday").

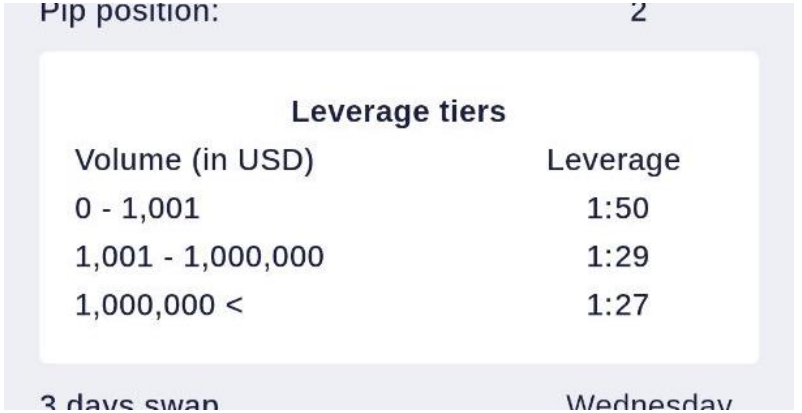
21. **Note that** if `Trader.swapFree`=TRUE, instead `swapRollover3Days`, `swapLong`&`swapShort` show "3 Days Rollover Commission" (`Symbol.rolloverCommission3Days`) and "Rollover Commission" (`Symbol.rolloverCommission`). "Rollover Commission" value must be divided by 100 (e.g. if value=1055 >> show 10.55) and its always will be accompanied with "USD per million USD volume" & "3 Days Rollover Commission" shows day of the week by field's ENUM.
22. Add appropriate values to `symbol: more details`. [value] will appear if its just need to be copy-pasted from response.
23. If a different text should be shown (not the value for given field name), it will appear in brackets and quotes.
24. Line (names appear on previous page)……data:
    a) `**measurementUnits**`.....[value],
    b) `**pipPosition**`……[value],
    c) `**enableShortSelling**` (**only** if =false)……"Not allowed", If=true – **don't show** that line. *For symbols with=false, on tap on SELL button, show popup – H:"Limited access", B: "This symbol is disabled for short trading".
    d) `**leverage**` (more details on the next p.),
    e) `**swapRollover3Days**` (and `rolloverCommission3Days`)… converted to day of week, by `DayOfWeek` ENUM from `Symbol.swapRollover3Days` field, if=N/A or empty, don't show.
    f) `**swapLong**`(and `rolloverCommission`)...[value]; add "pips", if `Symbol.swapCalculationType` ENUM=0/"%" – if ENUM=1
    g) `**swapShort**`(and `rolloverCommission`)...[value]; add "pips", if `Symbol.swapCalculationType` ENUM=0/"%" – if ENUM=1
    h) `**minVolume**`……[value]/100 "`measurementUnits`",
    i) `**maxVolume**`……[value]/100 "`measurementUnits`",
    j) `**minCommission**` (more details on p. 24),
    k) `**commission**` (more details on p. 24),
    l) `**schedule**` (more details on p. 25), and
    m) `**holiday**` (more details on p. 26, "Next holiday").
25. Please note that #e, #f, #g, #k, #l and #m are optional. If getting `null` or any of fields is missing – just don't show appropriate line in `symbol details`. **Same applies** for `rolloverCommission` and `rolloverCommission3Days`.

26. **`Leverage`** can have more than single number value, but some multilevel structure, that depends on user's trading volume. Use `ProtoOASymbol.leverageId` value and send `ProtoOAGetDynamicLeverageByIDReq`.
27. If in response's `ProtoOADynamicLeverage` will be a single `leverage tier` - show it as following: "Leverage: 1:(`ProtoOADynamicLeverageTier.leverage`/100) >> Ⓐ
28. If in response's `ProtoOADynamicLeverage` might be more than 1 `leverage tier`, make a mini list (with same design as for schedule) that will be named "Leverage Tiers". Sub headers will be named "Volume (in USD)" and "Leverage", while beneath tiers' data will be shown >> Ⓑ
29. For 1st tier show volume >> "0 – tier 1:`ProtoOADynamicLeverageTier.volume`/100" and appropriate 'leverage' as in #27. For the next tier show volume >>
"tier 1:`ProtoOADynamicLeverageTier.volume`/100 - tier2:`ProtoOADynamicLeverageTier.volume`/100"
and appropriate 'leverage' as in #27. Continue with this format till last tier is shown.
30. **Make another verification** when uploading leverage tiers. If `Trader.leverageInCents`< any `DynamicLeverageTier.leverage` - show the smallest (cause in that case server will execute by `Trader` leverage). If needed – present `Trader.leverageInCents` in more than one tier.

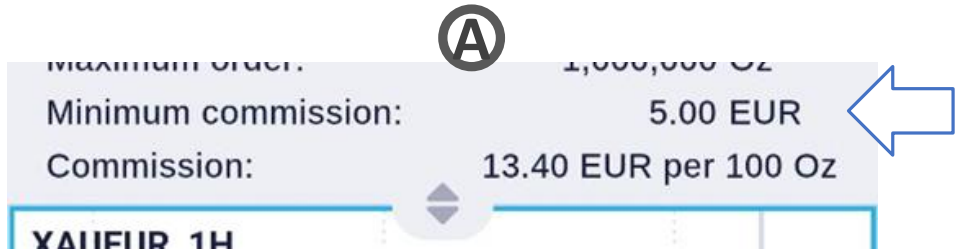
Ⓐ



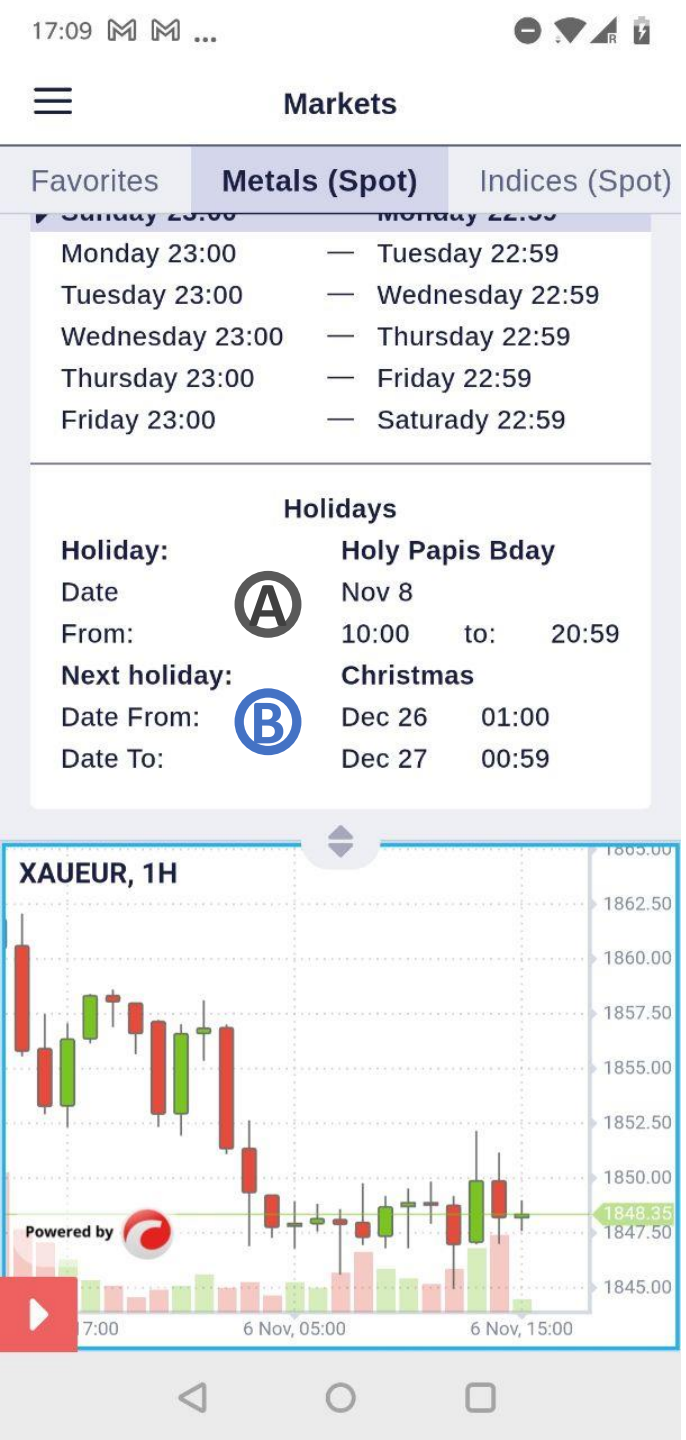
Ⓑ



31. Show **`minCommission`** only if there is any value for `Symbol.presiceMinCommission` >> Ⓐ
32. If there is a value – divide the value in response by 100000000; e.g. if the value is 1370000000 >> show 13.7.
33. Afterwards, check the `MinCommissionType` field. If its=1, add right to value from #2 (in capital letters) `Symbol.minCommissionAsset'.
34. If `minCommissionType`=2, add right to value from #2 `LightSymbol.symbolName` for appropriate symbol's `LightSymbol.quoteAssetId`.
35. `Commission` amount will be returned in `ProtoOASymbol.presiceTradingCommissionRate` >> Ⓑ
36. For converting the value of this field, convert the `Symbol.commissionType` value with `commissionType` ENUM; if value=[1,2,4] – divide `precise trading commission rate` by 100000000; if value=3 >> divide by 100000.
37. Right to amount add the following for `commission type` values: 1 >> "USD per million USD volume"; 2 >> "USD per `Symbol.lotSize`/100 `Symbol.measurementUnits`"; 3 >> "% of trading volume".
38. If `commission type`=4, use `LightSymbol.quoteAssetId` for getting matching asset in response to `AssetListReq`. Then add the following right to amount: "`Asset.name` per `Symbol.lotSize`/100 `Symbol.measurementUnits`".



39. **`Schedule`** (under sub header of "Trading hours") will be a mini list of all trading sessions of given symbol (can be more than 1 session during 1 day), while trading time for each day will be provided. Time must be adjusted to local time.
40. For each symbol schedule's time zone is specified in `Symbol.scheduleTimeZone` field.
41. Use https://www.javatpoint.com/java-util-timezone resource to convert time zones from .java format (that's the values sent in `Symbol.scheduleTimeZone`).
42. Make a list of all sessions sent in repeated `Interval` entities in `ProtoOASymbol.schedule`. All intervals are specified in seconds starting from Sunday 00:00 in specified time zone. Convert seconds to days and hours (e.g. 6300 sec will be translated to "Sunday 01:45" in the time zone of given symbol's schedule).
43. Each session (`interval`) will be shown as (day of week)(time HH:MM) – (day of the week)(time HH:MM), after conversion from symbol's schedule time zone to local time zone. Converted `Interval.startSecond` of each `interval` will be shown from the left side and converted `Interval.endSecond` will be shown from the right side and lines with smaller `startSecond` values will be above bigger `startSecond` values.
44. So, according to #41-42, place in first line below sub header "Trading Hours" interval with the smallest 'startSecond`.
45. If currently there is an active trading session - mark it with bold in Trading hours list.
46. `Schedule` field can **be empty**! Treat such case as 24/7 trading; show in one line: "Trading Schedule 24/7".



47. In the similar to `Symbol.schedule` way `Symbol.holiday` returns list of `Holiday` entities.
48. Holidays might be of 2 types: `Holiday.isRecurring`=true and `Holiday.isRecurring`=false.
49. `Is recurring`=false can be filtered by their absolute date (with year), while `Is recurring`=true might be filtered by date only, as `Is recurring`=true is param. of repeating every year holiday, so it happens every year in the same date (***check if expired are sent***).
50. If there is a holiday during current week – show 2 closest holidays ("Holiday" and "Next holiday"), if no holidays during the week – show 1 "Holiday".
51. In "Holiday" line the value will consist of date (format: Month, date) and `Holiday.name`.
52. There could be 2 formats of presenting a holiday: holiday happens during the same day on user's timezone and a period of 24 hours that cover 2 calendar dates in user's time zone.
53. If holiday is defined as "partial": fields `Holiday.startSession` and `Holiday.endSession` will return values; if fields empty: given holiday is defined in system as "whole day".
54. The next thing is to convert given holiday to user's time zone from `Holiday.scheduleTimeZone`, if needed. Use the same rules of #39-41.
55. If holiday happens during the same date of end user's timezone – use Ⓐ.
56. If holiday starts on one day by user's time zone and ends on the next – show holiday in format Ⓑ.
57. In case holiday is active: disable trading for that symbol, like its outside of trading hours.

Trading mode parameter

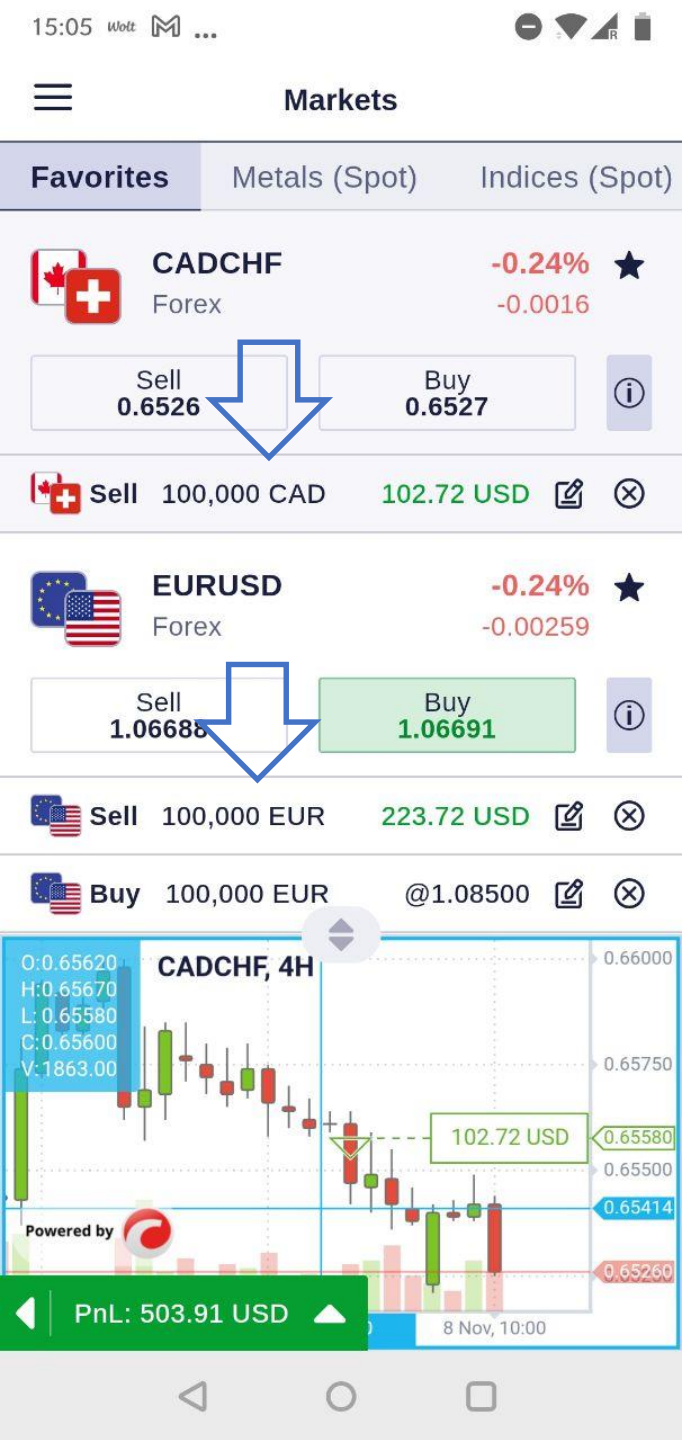
1. Additional parameter in `OASymbol` that should be considered, despite it won't be shown in app's symbol details is `TradingMode`.
2. Its settings are similar to `accessRights` parameter, but apply per symbol.
3. If `tradingMode`=0 (will be in 99% cases) – symbol can be traded without any limitations.
4. If `tradingMode`=1 or 2, treat **that symbol (only)** in a same way, as it would have `accessRights`=2 – trading disabled.
5. If user taps on buy/sell/edit/close, show the next popup: "Limited Access"; body - "Trading is disabled for this symbol, it is in a "view only" mode. Please contact your broker for more details."
6. If `tradingMode`=3, treat **that symbol (only)** in a same way, as it would have `accessRights`=1 – close only.
7. If user taps on buy/sell, show the next popup: "Limited Access"; body - "This symbol has a limited access. You can't create new orders, but only close or edit existing orders. Please contact your broker for more details."

Scrolling 'symbols list'

1. When user scrolls `symbols list` - the app should manage subscription to rates of visible symbols through `SubscribeSpotsReq` and symbols that scrolled out (became invisible to user) should be unsubscribed from `SpotEvents` (rules from **`Symbol area` entities and behavior**, #5 - #8*)* .
2. Exclude the symbol presented on chart from unsubscribing, till any new `symbol area` will be tapped, so chart's data will be updated **live** till "active" symbol was changed.
3. In order to avoid missing any request/response from the server, make an algorithm that will verify every symbol's request-response to subscribing-unsubscribing, so no repeated connection/disconnection attempts would be made (will get **an error response** from server).

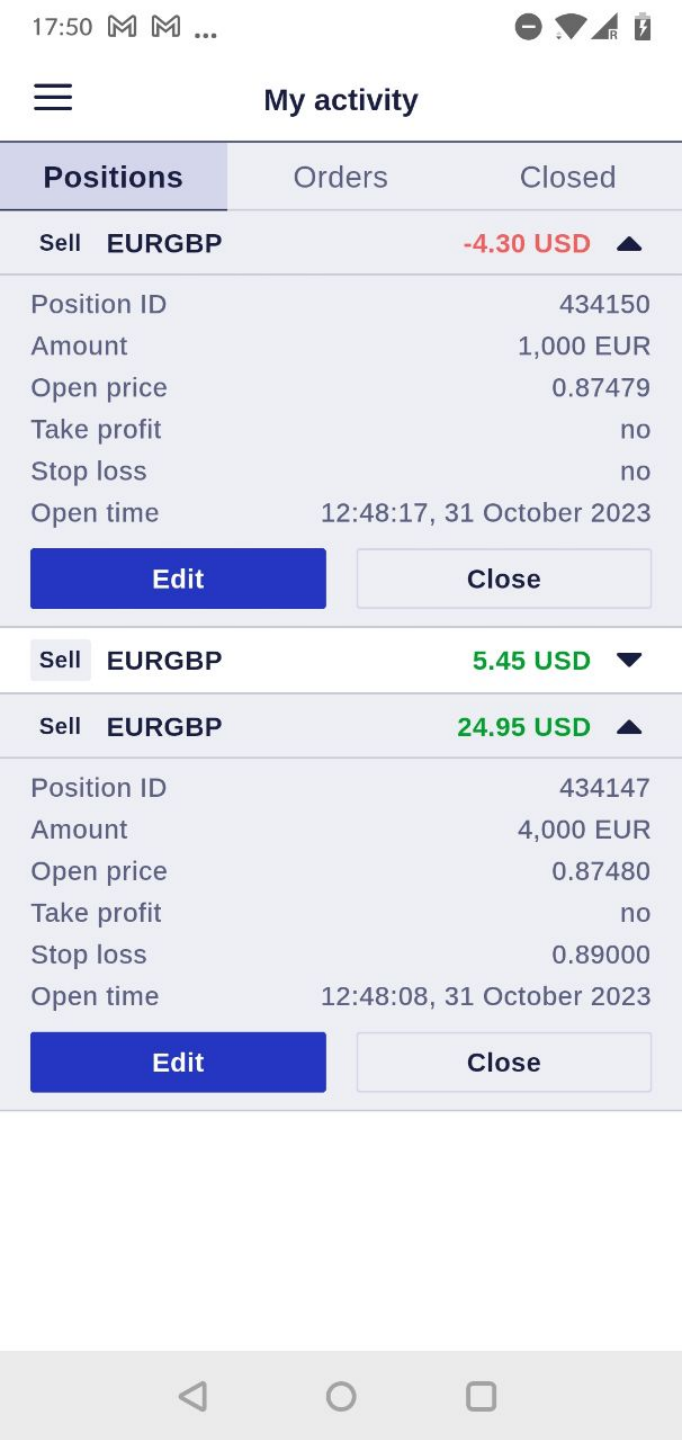
`Quick account stats`

1. In lower left corner of chart (unrelated to chart's code, will be in upper layer) there will be `quick account stats` object.
2. `Quick account stats` will have 3 positions: closed, half-open and open (see pictures).
3. By default (first session) `quick account stats` will be half-open (showing only PnL).
4. Tapping on arrow icons will expand/collapse it, while the order of states will always be closed >< half-open >< open, e.g. there won't be a way to switch form closed to open in a single tap.
5. When `quick account stats` will be fully collapsed – no data will be shown, but the color of the object will be an indication whether given account is profitable or not. If PnL=>0 – the color will be green, otherwise – red.
6. When `quick account stats` will be half-open – it will show live account's `Unr. net PnL`, updated live (same as in **My account** #18). Color object's background according to #5.
7. When `quick account stats` will be open – show more account stats:
   a) Balance
   b) Equity
   c) Margin
   d) Unr. net PnL
8. All these stats should be calculated in the same way as in **My account**.
9. Show `quick account stats` in half-open state as long as user won't tap any arrow. After tapping – remember the last state user left it.
10. Every time `quick account stats` changes its state – change the direction of appropriate arrow to reversed.



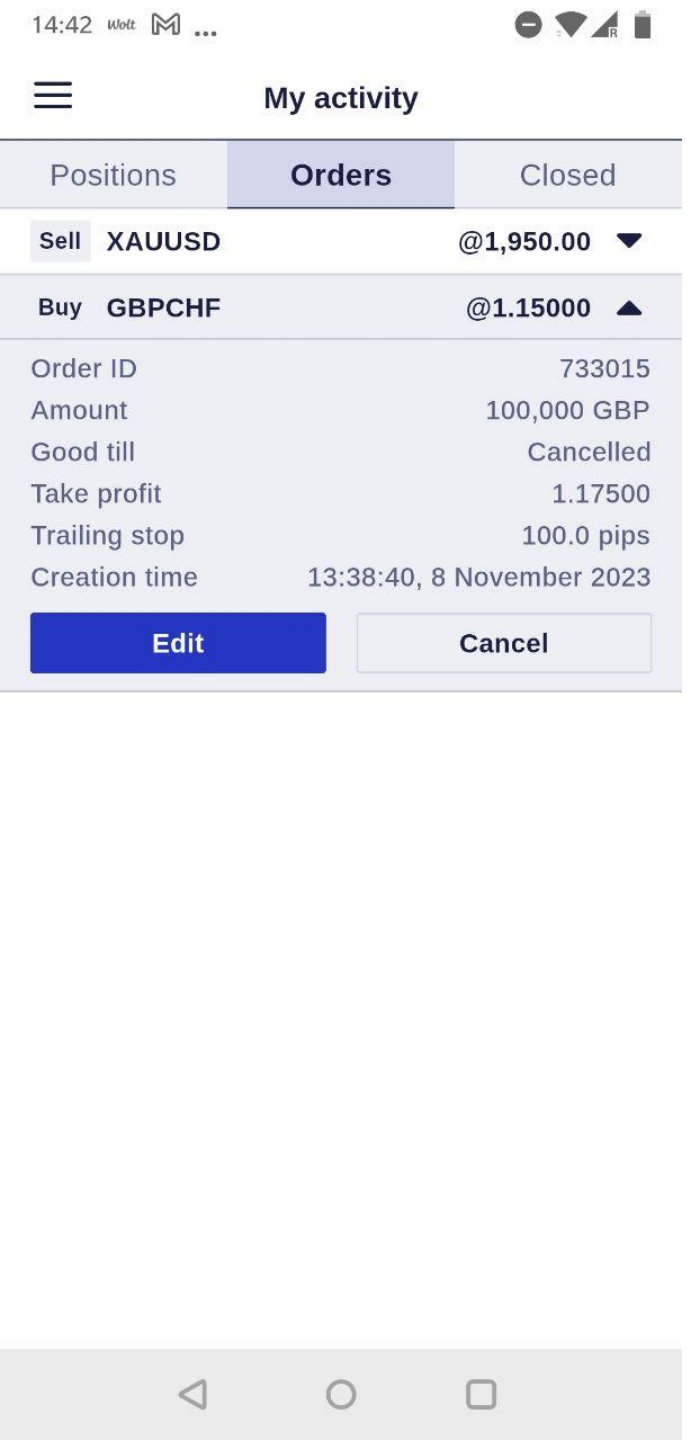
Showing positions and orders in "Markets" screen

1. If "Show open positions/limit orders in Markets screen" is on "ON" (default), user's `positions` and `orders` will be placed below appropriate `symbol area` in additional lines.
2. On new session's launch: all positions &orders accepted in `ReconsileRes` will be shown; later newly opened positions or created pending orders will be also added. 2nd case will be discussed in "**Execution events**".
3. Show the following details for both `orders` and `positions`: symbol icon (mini), direction (`tradeSide`, show "Buy" for ENUM=1 and "Sell" for ENUM=2), `amount` (`order/position.tradeData.volume`; in units; same as in `symbol: more details`, e.g. "50 **oz**."/"10,000 **AUD**").
4. In case of `orders`, if field `order.executedVolume`>0 – deduct its value from `order.tradeData.volume`. ***Note** that in case of editing given order – the amount in request should be unchanged.
5. If the line is too long (can happen for positions/orders >100,000,000) – reduce the font to 12. If still doesn't fit - to 10: smallest font possible. If still doesn't fit – make in 2 lines.
6. Last info piece for `order`: "@"'stop price`/'limit price'" (depends on which filed returned in appropriate 'ProtoOAOrder' entity in `ReconsileRes`; more details about `limit order` and `stop order` in `pending orders`, p. 41-43), 'edit` and `delete` icons.
7. Last info piece for `position`: value of `netUnrealizedPnl` for given `positionId` (as in My Account #18; add value of `depositAsset`). Will be followed by `edit` and `delete` icons.
8. Update PnL values of open positions by new responses of `GetPositionUnrealizedPnLReq`.
9. `tradeSide` and `netUnrealizedPnL` will be colored red if PnL<0, otherwise – green.
10. Tapping on `edit` or/and `delete` of both `orders` and `positions` will be explained later in Editing positions, Closing positions, Editing orders and Cancelling orders.



My activity: positions

1. My activity shows info of positions/orders/history of given `accountId`. As on **Showing positions and orders in "Markets" screen**, it uses data accepted in `ReconsileRes` (`DealListRes` for closed), later added by data accepted in `ExecutionEvents` (p. 54-74).
2. First tab shows `positions` for given account (bigger `openTimestamp` > higher in the list), i.e. details of all `position id` that were returned in `ReconsileReq` in `position` entities.
3. Each position line has a header with: 'tradeSide' ("Buy"/"Sell") from position's 'TradeData', 'symbolName' (find by appropriate `position.symbolId`), 'netUnrealizedPnL' (update once a sec; add value of `depositAsset`) and (↓) icon.
4. `tradeSide` and `netUnrealizedPnL` will be colored red if PnL<0, otherwise – green.
5. If (↓) is tapped, some details will open below and icon is changed to (↑):
6. "Position ID".....'ProtoOAPosition.positionId'
7. "Amount"..... `Position` > `tradeData.volume'/100 (apply thousands/decimals separators, if needed) "`Symbol.measurementUnits`". If `measurementUnits`=empty, use `LightSymbol.baseAssetId', find the asset and insert its value, like "USD".
8. "Open price".....'ProtoOAPosition.price'.
9. "Take profit"……'Position.takeProfit' (if value=0/empty >> show "No").
10. "Stop loss"….'Position.stopLoss' (if value=0/empty >> show "No"). Show the value also for 'guaranteedStopLoss'=TRUE ("Guaranteed stop loss" will be instead of "Stop loss").
11. "Trailing stop loss"….If `Position.trailingStopLoss'=TRUE – will be instead of "Stop loss". Show absolute |`price`-`stopLoss`| X $10^{\text{Symbol.pipPosition}}$ "pips". If=FALSE or N/A – go to #10.
12. "Open time"…'Position' > 'tradeData.openTimestamp', convert to HH:MM:SS, DD MM YY.
13. Tap on `edit`/`delete` positions is explained later in **Editing and Closing positions**.
14. If no positions were found show: "No active positions in this account at this moment".

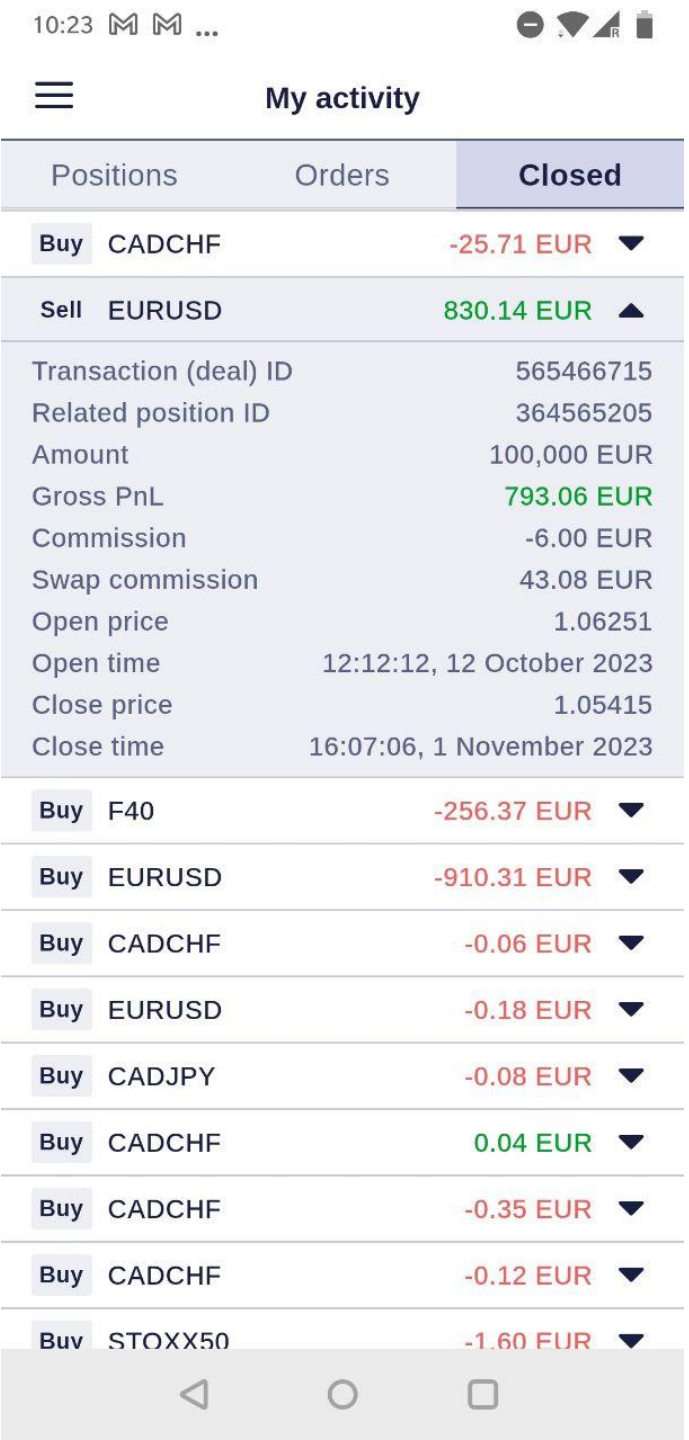


My activity: orders

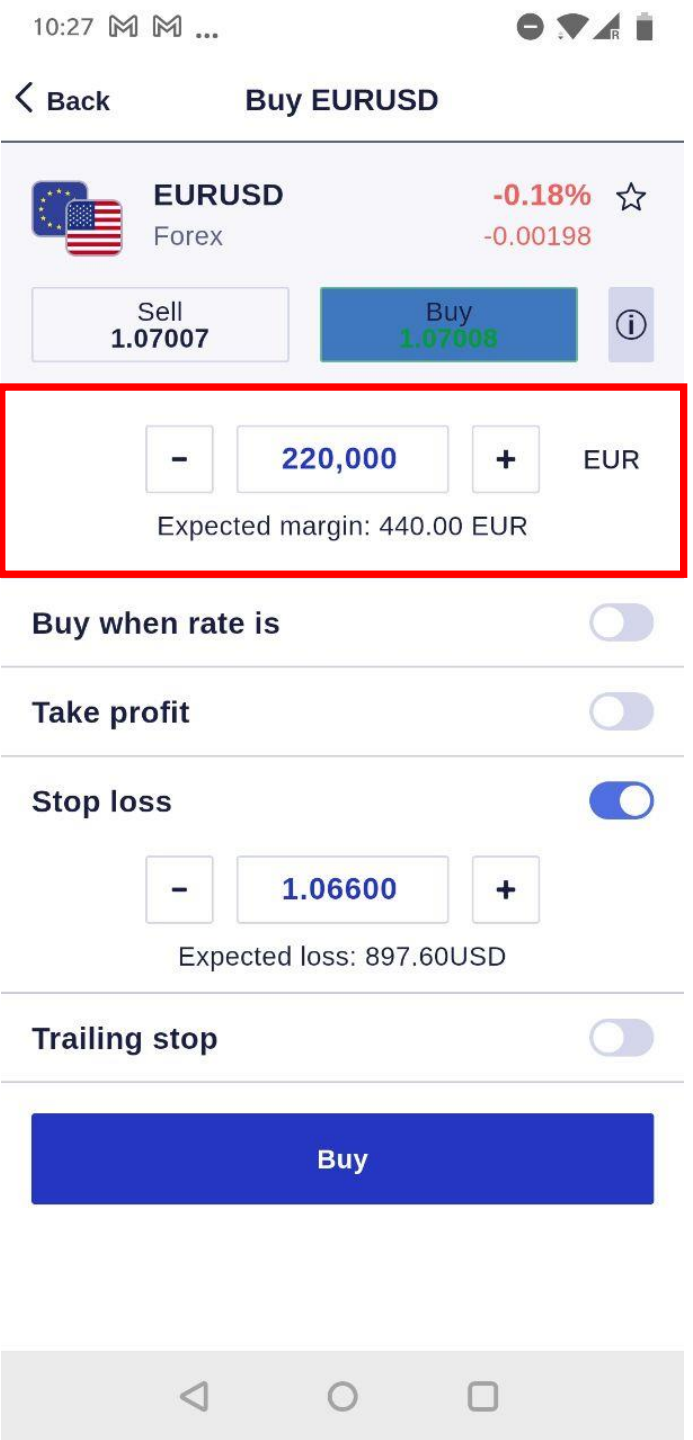
1. 2nd tab shows all pending `orders` for the account (bigger `openTimestamp` >> higher place), i.e. details of all `order id` that were returned in `ReconsileReq` in `order` entities.
2. Each order line has a header with: 'tradeSide' ("Buy" or "Sell") from this order 'TradeData', 'symbolName` (find by appropr. `symbolId`), "@"`limit/stop price` (depends on which field returned in appropriate 'ProtoOAOrder' entity in `ReconsileRes`;) and (↓) icon.
3. If (↓) is tapped, some details will open below and icon is changed to (↑):
4. "Order ID".....'ProtoOAOrder.orderId' in appropriate `order` entity in `ReconsileRes`.
5. "Filled amount"... ONLY if order.executedVolume>0, show its value in same format as #6.
6. "Amount"... `tradeData.volume'/100 (apply thousands/decimals separators if needed) "`Symbol.measurementUnits`". If `measurementUnits`=empty, use `LightSymbol.baseAssetId', find the asset and insert its value, like "USD".
7. "Good till"... If 'time in force' > ENUM=2 – "Canceled", if ENUM=1 (GTD) – convert the `expirationTimestamp` to HH:MM DD:MM:YY. If `expirationTimestamp`=null – "N/A".
8. "Take Profit" - calculate by the formula below (if empty – show "No");
   `buy` order >> `limitPrice`/`stopPrice` + `relativeTakeProfit`/100,000;
   `sell` order >> `limitPrice`/`stopPrice` - `relativeTakeProfit`/100,000.
9. "Stop Loss" - `buy` order >> `limitPrice`/`stopPrice` - `relativeStopLoss`/100,000;
   `sell` order >> `limitPrice`/`stopPrice` + `relativeStopLoss`/100,000.
10. "Trailing stop loss"....If `order.trailingStopLoss'=TRUE – will be instead of "Stop loss". Show `relativeStopLoss`/100,000 $*10^{Symbol.pipPosition}$. Add "pips". If=FALSE or N/A – go to #10.
11. "Creation time"...'Order'>'TradeData.openTimestamp' >> covert to HH:MM:SS, DD MM YY.
12. Tap on `edit`/`delete` `order` is explained later in **Editing orders and Closing orders.**
13. If there no orders were found show: "No pending orders in this account at this moment".

My activity: closed

1. 3rd tab shows account's closed positions; no historic data accepted in `ReconsileRes, so a special request to server should be made.
2. **Please note:** OA applications have a limitation of time period for deals request: 1 week. Also, only 5 requests per second can be sent for historical data (like `deals`).
3. Send `DealListReq`, with `toTimestamp`=current t.stamp and `fromTimestamp`= (`current` - 604800000), e.g. getting deals of 1 week to this moment (with `maxRows`=250)
4. **Plz note**: `fromTimestamp` should **never** be less than `Trader.registrationTimestamp`.
5. If `DealListRes` returned `deal` entities with `closePositionDetail`=value (some can have empty field) < 50 – send another request while its setting with timestamps 1 week exactly from `fromTimestamp` from previous request. If still `deals`<50, repeat up to 5 requests in order to get 50 or more deals.
6. If `deals.closePositionDetail`=value` =0 in 5 requests >> popup: header - "Trading History", body "No resent trading activity was found. Tap on "Load More" to get trading history for longer period". Tap "OK" closes the popup. If "Load more" is tapped, repeat #3, #5, #6.
7. If 1-50 (or more) deals with `closePositionDetail`=value are accepted after any `DealListReq` - upload the results w\o popup, but with active button "Load more" in the bottom.
8. If `fromTimestamp`<`registrationTimestamp` is reached – show on popup: "All history was loaded" with OK. After OK is tapped - "Load more" is inactive.
9. If during session user leaves `closed`, then `ExecutionEvent` is accepted and user is back on `closed` – repeat #3-8. If no `ExecutionEvent` comes – use accepted previously data.

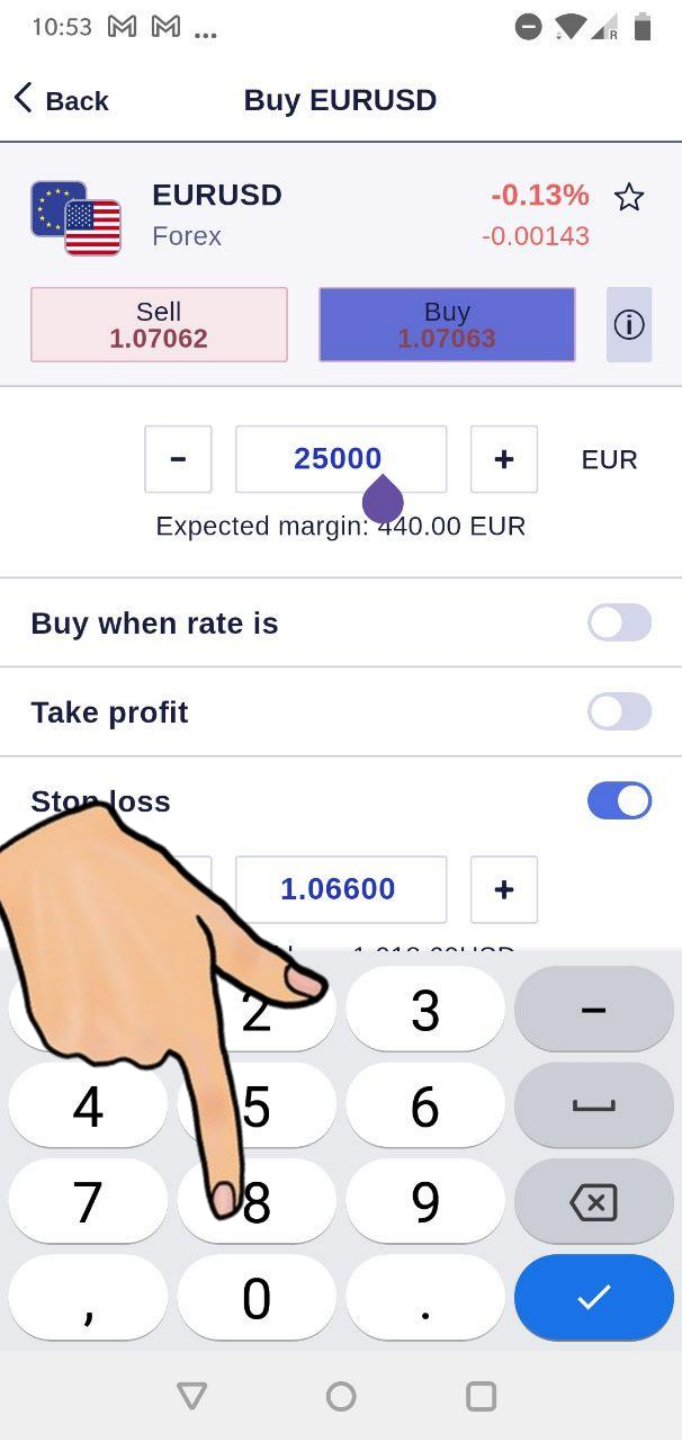


10. In case when no. of accepted deals in `DealListRes` has returned 250 **OR** `hasMore` field= TRUE, find deal with earliest `executionTimestamp`. Next REQ will be from that TS-1ms.
11. After filtering the list of deals with `closePositionDetail`=value from all deals in response – send for every such `deal.dealId` `ProtoOADealOffsetListReq` >> `offsetting` - that will return details of deals that opened given closing transaction. It will be used later.
12. Each closed position line has a header, that includes: 'tradeSide' ("Buy" or "Sell") from the deal with `closePositionDetail` – use the **opposite side**, 'symbolName` (find by appropriate `deal.symbolId`), position's Net PnL with value of `depositAsset` and (↓).
13. Net PnL= `closePositionDetail.grossProfit`+`swap`+`commission`). ***`swap` can be positive.**
14. "Transaction (deal) ID:"…`deal.dealId` (of appropr. deal with `closePositionDeal`=value).
15. "Related position ID:"…`deal.positionId`
16. "Amount:"…`deal.filledVolume`/100, add `measurementUnits` (if `measurementUnits` =empty, add `LightSymbol.baseAssetId').
17. "Gross PnL:"…`closePositionDetail.grossProfit`; with value of `depositAsset`.
18. "Commission:"…`closePositionDetail.commission` with value of `depositAsset`.
19. "Swap commission:"…`closePositionDetail.swap` with value of `depositAsset`.
20. "Open price:"… for 1 `dealOffset` - use its `executionPrice`; if returned 2+ `dealOffset`, get it by the formula: SUM [(dealOffset.volume * dealOffset.price) / SUM(dealOffset.volume)]. Round to closest `digit` if result has bigger precision (e.g. 1.234533333 round to 1.23453).
21. "Open time:"… If there is 1 `dealOffset` - use its `executionTimestamp`, convert it to local time in HH:MM DD:MM:YY format and add as line's value; if accepted more than 1 `dealOffset`, find the earliest `executionTimestamp` of all accepted, convert and add.
22. "Close price:"… `deal.executionPrice` (of deal with `closePositionDetails`=value);
23. "Close time:"… `deal.executionTimestamp` - convert and add.

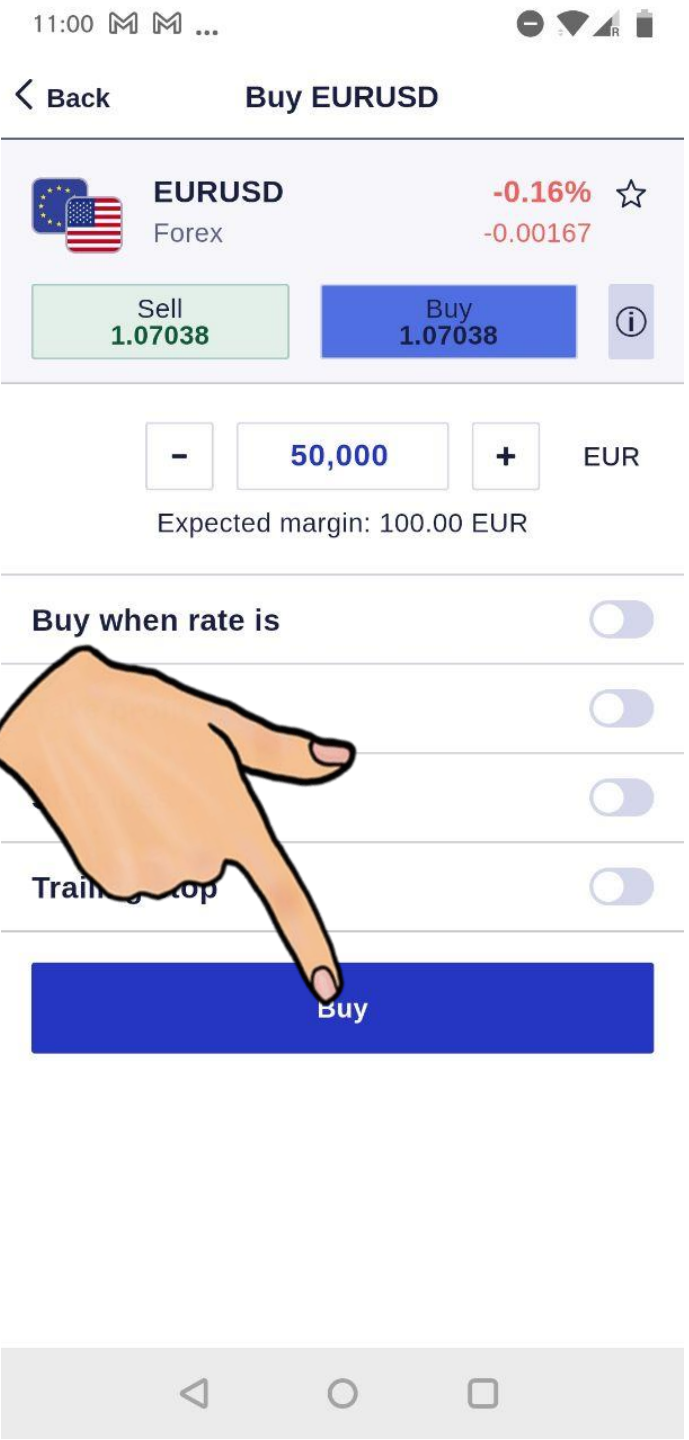


New Buy/Sell order

1. If user taps in `symbols list` on any `buy/sell button` of appropriate symbol - redirect user to new screen. Send `ProtoOASymbolByIdReq` with appropriate `LightSymbol.symbolId` (or use existing, if symbol was used in **`Symbol area` entities and behavior** #17).
2. Unsubscribe from other symbols (`UnsubscribeSpotsReq`), keep that symbol only.
3. Send `ExpectedMarginReq` for `Symbol.minVolume`. Use either `buy` or `sell` from response, depending on tapped earlier (`buy` or `sell`) button in `markets` screen. ***Note** that `expectedMargin` for limited risk accounts (p. 44-45) will be calculated manually.
4. Generate header for the screen: "Buy/Sell (by tapped button) `lightSymbol.symbolName`".
5. Below header, `symbol area` will remain with more options below. Tapped buy/sell from `Markets` screen is set (can be changed by tapping on sell/buy >> header changes, too).
6. First additional area below `symbol area` will be `trade amount`. It will have `trade amount` field and (-)/(+) buttons for changing the amount. Right to (+) – show `symbol.measurementUnits`.
7. Any value in `trade amount` will be equal to `Symbol.minVolume`+ `Symbol.stepVolume * X, but no more than `Symbol.maxVolume`. Values are sent in cents, so the amount shown to user always must be divided by 100. 1 tap on (+)/(-) adds/reduces 1 `stepVolume` to current `tradeAmount`. Default value=`minVolume`.
8. After any order is successfully executed for given symbol – let its `volume` be the next default volume (per account).
9. When minimal volume is shown: (-) button is deactivated.
10. Define max. volume (for disabling (+)) as such when `expected margin` < `free margin`.
11. If user has other open positions, `free margin` can change; verify the condition in #10=TRUE; if false, reduce `tradeAmount` by 1 `minVolume` till condition #10=TRUE again.

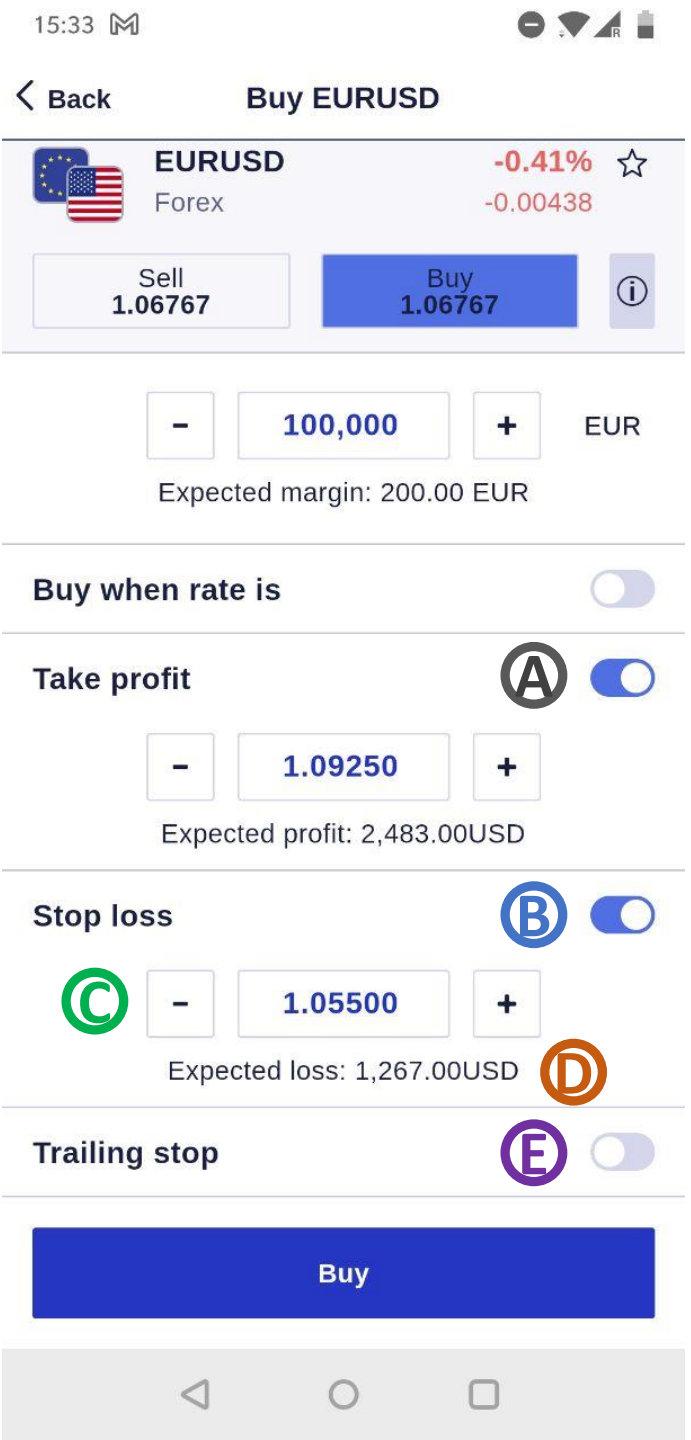


12. Below `trade amount` add "Expected margin:" value from #3. Proportionally calculate expected margin for `step volume` and proportionally add/reduce to margin, as `trade amount` added/reduced by user. Round the `expected margin` to closest min. value; anyway the exact margin will be available after execution of the order.
13. If "new order" screen is opened for more than 10 seconds – send another `ExpectedMarginReq` (cause rate was probably changed till now) to get new value – and calculate proportionally the correct new `expected margin` for current `trade amount`.
14. When user taps "+" or "-" – add/reduce 1 `stepVolume` from `trade amount`.
15. When user taps "+" or "-" and holds it, add or reduce 1 `stepVolume` each 0.3 seconds. If held for 2 seconds+: accelerate the addition/reduction to 1 `stepVolume` each 0.1 second.
16. After any change of `trade amount` - make the appropriate change of "expected margin" value, by multiplying the values you have from #3, #7, #10.
17. `trade amount` field will be tapable, in case user taps inside it – activate cursor in the place of tap (either left or right to amount, or between the digits) and activate **numeric** keyboard.
18. When numeric keyboard active – if user taps outside of keyboard – keyboard will collapse.
19. If user closes numeric keyboard – make validation of typed value and round it, if needed, on numeric keyboard's collapse.



Sending order to execution

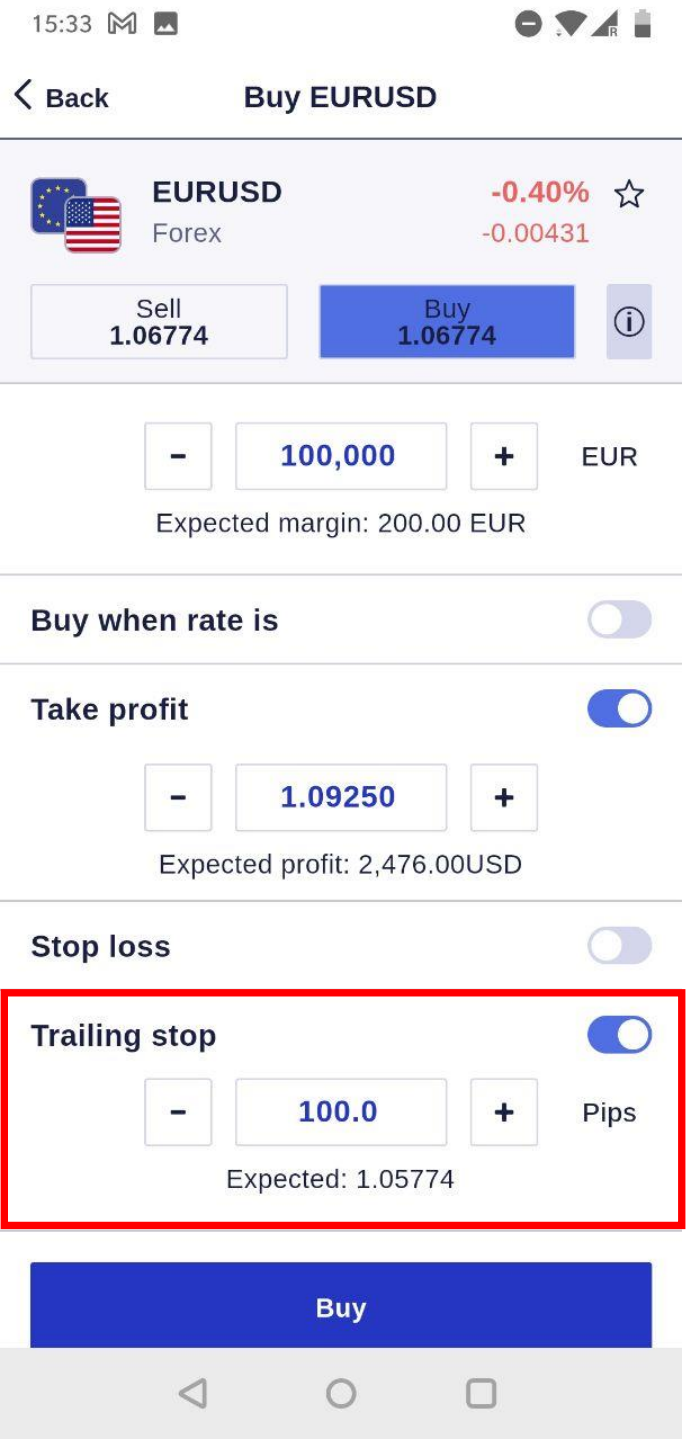
1. This page will go over basic parameters and options for sending orders for execution, while next screens will explain more advanced options.
2. When order screen has some amount (even default) and direction ("Buy"/"Sell"; was set previously by tap on "Markets" screen) – the order can be sent to execution.
3. If bottom "SELL" button is tapped on "Sell EURUSD" screen – send `NewOrderReq`.
4. The following fields must be filled in order that it will reach the market: `ctidTraderAccountId` (from `GetAccountListByAccessTokenReq` >> `CtidTraderAccount`), `symbolId` (from either `Symbol` or `LightSymbol`), `orderType` (with given settings – ENUM=1: MARKET), `tradeSide` (1 for BUY, 2 – SELL), `volume`: value in `trade amount` field, multiplied by 100 and `timeInForce` (with given settings) ENUM=3: IMMEDIATE_OR_CANCEL.
5. At first stage order will be validated in cServer, whether all details are filled appropriately. In case order was sent with any errors, order will be rejected and `OrderErrorEvent` will be sent in response. In `description` field it will include the reason ***for order rejection***.
6. If accepted – show popup with header: "An error occurred" and body=value of `ProtoOAOrderErrorEvent.description`. Button "OK" will close the popup.
7. ANY order that isn't sent/accepted properly and rejected on cServer's side – app will be notified by `OrderErrorEvent`; so expect it after sending any type of order or other trading action.
8. If `ErrorEvent` is accepted – show popup - H: "An error occurred", B: [description of `ErrorEvent`].
9. If order is validated, the first (but not the last) `ProtoOAExecutionEvent` will be sent to verify order's validity. All execution event types/possibilities will be discussed later.



Setting TP or/and SL

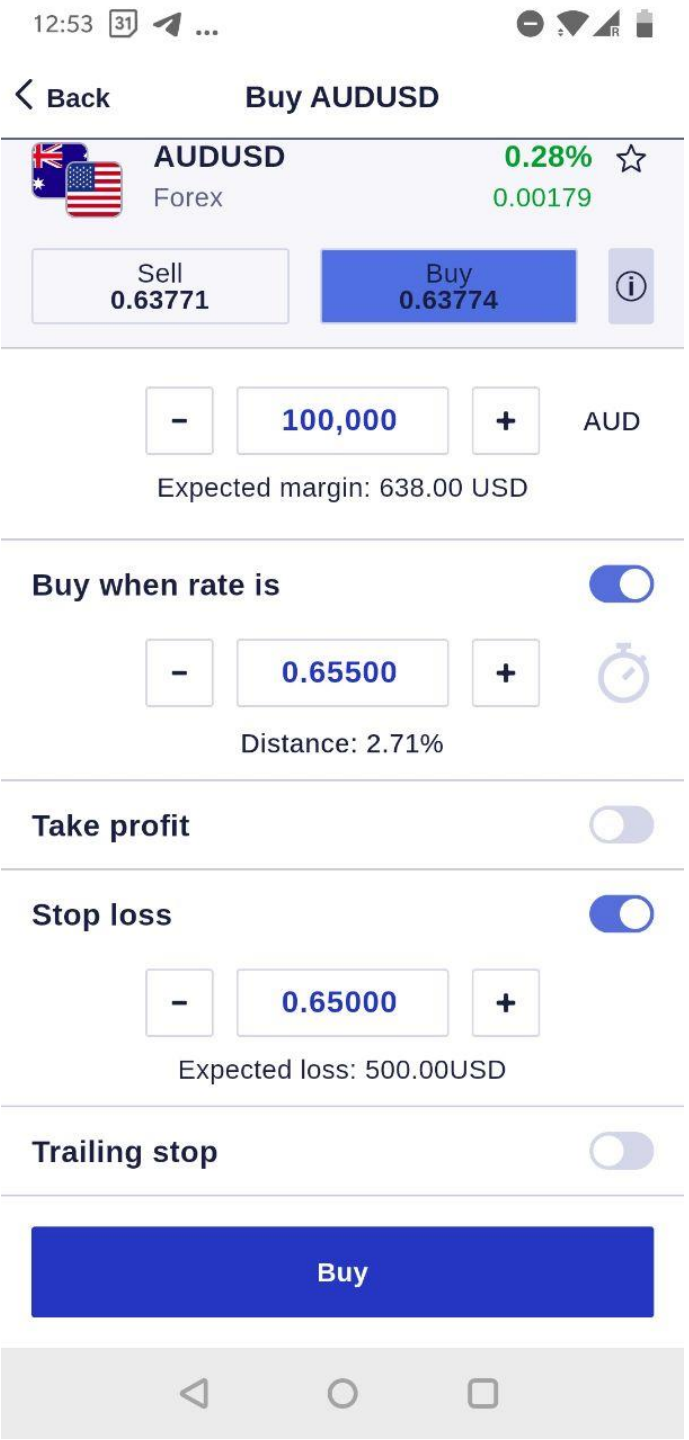
1. When user will activate `take profit` (A) or/and `stop loss` (B) options, next will happen:
2. Below the appropriate activated option (either `take profit` or `stop loss` or both) `take profit rate` or `stop loss rate` (C) field with (+)/(-) will appear.
3. Below `take profit/stop loss rate` field `expected profit/loss value` (D) will appear.
4. Below `expected loss` value a `trailing stop` (E) option will appear (or below 'stop loss' option, if untapped).
5. If `take profit` or `stop loss` option was activated, the following should be done:
6. Get values of `ProtoOASymbol.tpDistance` and `ProtoOASymbol.slDistance`. **Important:** for some users `slDistance` won't be provided, but `gslDistance`. It will have same usage as `sl`.
7. Get also `Symbol.distanceSetIn` in order to calculate `tp/sl distance rate`.
8. If `distance set in` ENUM=1 (points) >>`tp/sl distance rate`= `tp/sl distance`/$10^{Symbol.digits}$
9. If `distance set in` ENUM=2 (set in %), then `tp/sl distance rate`=`tp/sl distance`/10,000 *(last bid or ask; bid – for sell order, ask for buy order).
10. **Please note:** `sl/tp/gslDistance` might **not** be returned/be `null`. In that case theoretically both TP/SL can be placed in minimal distance (1 `digit`) from appropriate `bid/ask`. Please set it on 5 pips, so position won't be closed around the moment of its opening.
11. When user activates tp/sl option: if given order is `buy` - `tp distance rate`*2 will be added to `ask` rate and `sl distance rate`*2 deducted from `bid` rate. If given order is `sell` - `tp distance rate`*2 will be deducted from `bid` and `sl distance rate`*2 will be added to `ask`.
12. In no way tp/sl rate can't get closer to last bid/ask than 1 `tp/sl distance rate`. If such situation happens because of new bid/ask rates – automatically update tp/sl rate by minimal value, so the distance from last bid/ask will be bigger, than 1 `tp/sl distance rate`.

13. So, if order is SELL, last `bid` is 1.23458 and `tpDistance` is 138 points – the default `take profit rate` is 1.23458 – (138/10^5)*2 = 1.23182. In case at some point (before order execution) `bid` will slide below 1.23320 – update the `take profit rate`, so (`bid` - `take profit rate`) >= 'tp distance rate`.
14. Below either `take profit rate` or `stop loss rate`, calculate the `expected profit` (or loss). **Please note**, that values can come with minus: for example, given position is in big profit, so it's `stop loss` can be put also above "brake even" mark and position will be profitable in any case (e.g. `expected loss` will be with minus >> will be in profit).
15. For SELL order, calculate `expected profit` by next formula: [(`SpotEvent.bid` - `take profit rate`) * `trade amount`] "value of `LightSymbol.quoteAssetId`" and update it every new `bid`. For SELL and `expected loss` calculate [(`stop loss rate` - `SpotEvent.bid`) * `trade amount`] "quote asset" (and update it every new rate).
16. For BUY order, calculate `expected profit` by [(`take profit rate` - `SpotEvent.ask`) * `trade amount`] "quote asset" and `expected loss` by [(`SpotEvent.ask` - `stop loss rate`) * `trade amount`] "quote asset". Update it each new rate.
17. Allow user to change rate by tapping (+)/(-) buttons. Each tap changes rate for 1 `digit`. Tap and hold will allow quicker rate change, changing one digit each 0.3 seconds, after 2 seconds – digit each 0.1 seconds and after 2 **more** seconds – 10 digits each 0.1 second (change 2nd from right digit instead of changing 10 times 1st digit). After 3 more seconds change 3rd digit.
18. Also allow user to change the rate with numeric keyboard. In addition to #7-12, rate can't be negative – the lowest rate available should be 0. If user typed value which is closer to `ask`/`bid` than 1 `tp/sl distance` - update the rate **immediately,** as in previous page, #12.
19. In case user typed fraction, smaller than allowed by `digits` field value (e.g. 1.2345678 for symbol with `digits`=5) round the rate to the closest rate according to symbol's `digits` condition on keyboard collapse.
20. When "SELL" (from our example) button is tapped – fill `relativeTakeProfit` & `relativeStopLoss' fields to set `tp` or `sl`: *(`tp rate` - `current ask`)*100,000 or/and (`current ask` - `sl rate`)*100,000* for BUY `market` orders AND (`current bid` - `tp rate`)*100,000 or/and (`sl rate` - `current bid`)*100,000; e.g. if new order=SELL, `bid`=1.23458 and `tp rate` is 1.23001 = put in `relativeTakeProfit` = 457.



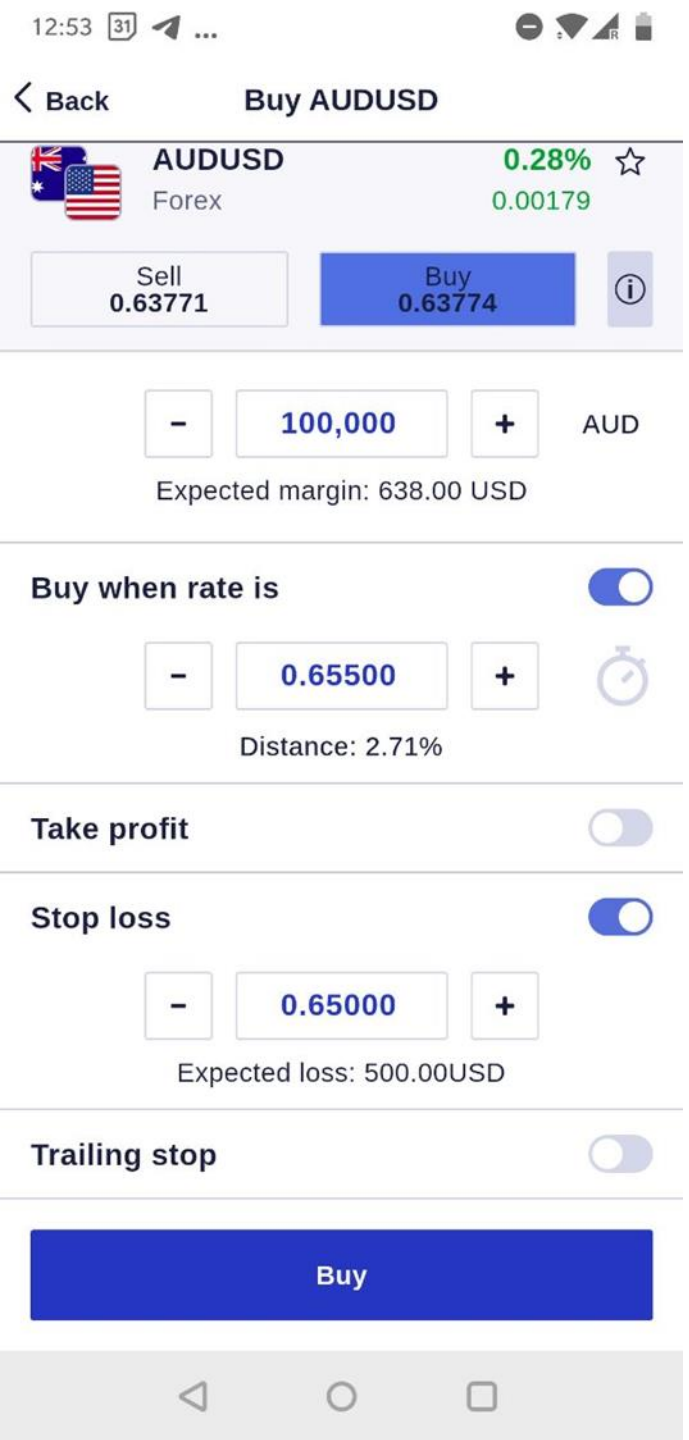
Setting Trailing stop

1. If user taps `trailing stop` – `trailing stop distance` field opens below, with (+)(-) buttons.
2. #1 collapses `stop loss` field/rate, but **keep in memory** its value, if user on this screen. If user deactivates `trailing stop` - activate previously set `stop loss` values as they were set.
3. Get `sl distance rate` for given symbol from **Setting Take profit or/and Stop loss** #6-10 and convert it to pips by multiplying by $10^{(\text{`pip position`})}$.
4. Show MAX: [#3 value X 2, 10] "pips" in `trailing stop` field at the moment of activation.
5. If `Symbol.digits` > `pipPosition`, allow setting tenths of pip (but no hundredths or smaller).
6. If `digits`=`pipPosition` - show in `trailing stop` only whole pips numbers.
7. As in **New Buy/Sell order** #14-15" – allow quick change by tap and hold action, while smallest unit of change will be either 1 pip or 0.1 pip, depending on #5, #6.
8. As in **New Buy/Sell order** #17-18" – allow typing in `trailing stop` value from numeric keyboard. If user types value of XXX.**XXX** format – round it to closest digit set by #5. On keyboard collapse make **New Buy/Sell order** #19 validation.
9. If user types value less than `sl/gsl distance` **in pips** - update the value **immediately**.
10. If user taps on SELL button when `trailing stop` option is activated and verified – fill in `NewOrderReq.trailingStopLoss`=true.
11. Fill in `relativeStopLoss`=`ask`- TS pips value/$10^{\text{pipPosition}}$ for Buy order and `bid`+ TS pips value/ $10^{\text{pipPosition}}$ for Sell order. Calculate this value according to changes in TS field and show live value in "Expected" field. **Verify that TS pips value > `sl/gslDistance`+spread.**
12. TS can't be bigger than current `bid` rate (in pips). Disable (-) button if TS=(`bid`– 1 digit).
13. If order sent for execution (both MARKET and pending) was verified by `ExecutionEvent` type=2 (order accepted) save in **app's cache** trailing stop value in pips linked to appropriate `orderId` and `positionId` - there will be a usage of this info when editing this order.

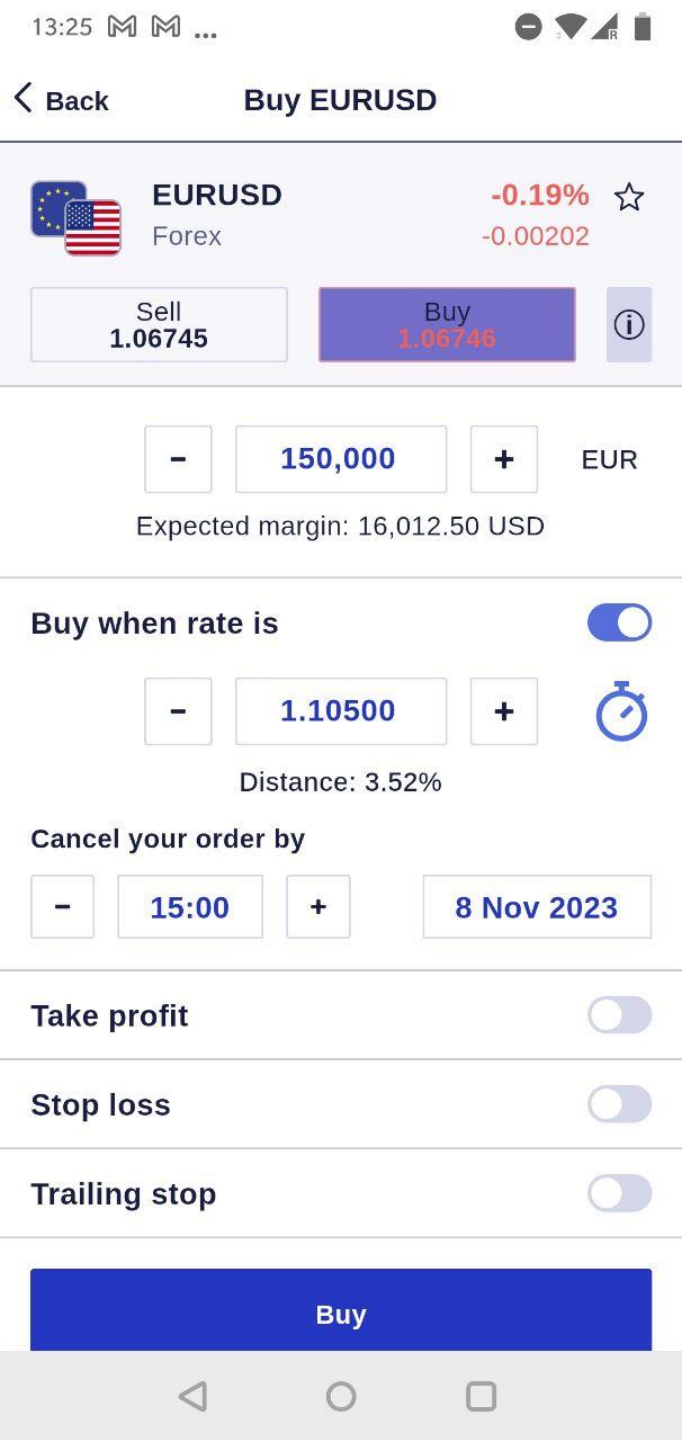


Sell/Buy when rate is

1. When user will activate `sell when rate is` (or buy, of course) `pending order rate` field will appear below the activation button with (+)/(-) buttons. Clock icon will appear from right side of "Sell when rate is" (more details on "clock" icon in **Sell/Buy when rate is #19**).
2. If "Buy when rate is" active, set a default rate that will be `SpotEvent.ask` *1.001.
3. If "Sell when rate is" active, set a default rate that will be `SpotEvent.bid` *0.999.
4. As an additional info for end user – calculate on tap and maintain the "Distance" (will be shown below `pending order rate`). Distance will show in-% the distance between current `pending order rate` and the last `ask` (for "Buy when…") or `bid` (for "Sell…").
5. When user is setting `pending order rate` - allow actions >> **Setting TP or/and SL** #17-19.
6. User can set any rate he wants in `pending order rate` window, in condition rate>0 and ≠ `current rate`. If `current rate` is reached (by market movement, for example) – decrease the rate in `pending order rate` window by 1 pip.
7. When setting `pending order` - tp/sl parameters can be set as well. For that purpose, use the `tp/sl distance rate` from **Setting TP or/and SL** #6-11.
8. When `pending order` is activated, calculate `spread` - the difference between `ask` and `bid` rates at that moment.
9. If user activates TP/SL when `pending order` is activated (+ some `pending order rate` is set) - if given `pending order` is `buy` - `tp distance rate`*2 will be added to `pending order rate` and (`sl distance rate`*2 **+`spread`**) will be deducted from `pending order rate` rate. If given `pending order` is `sell` - `tp distance rate`*2 will be deducted from `pending order rate` and (`sl distance rate`*2 +`spread`) will be added to `pending order rate`.
10. TP can NOT get closer to `pending order rate` than one 'tp distance rate` and SL can NOT get closer to `pending order rate` than one (`sl distance rate` + `spread').



11. In case end user activates `trailing stop` for `pending order` - set as default (`sl distance rate`*2 + `spread`)*$10^{(digits)}$ and show in `trailing stop` field. Can NOT be less than (`sl distance rate` + `spread`) (in pips); (deactivate (-) button when min. value is reached.
12. If activated `trailing stop` is set for `pending order`, add `trailing stop loss`=TRUE, divide `trailing stop` (in pips) by $10^{(pipPosition)}$ and either deduct that value from `pending order rate` (for BUY pending orders) OR add that value to `pending order rate` (for SELL pending orders). Verify that #10 and #11 are valid.
13. In case end user deactivates `Buy/sell when rate is` option – all settings/distances/tp/sl must get back to usual **Setting TP or/and SL** rules.
14. When user will tap BUY button, when `pending order` has rate **higher** than the last ask rate: add into `NewOrderReq` next fields: `orderType` ENUM=3 (STOP), `stop price`=`pending order rate`, `time in force` ENUM=2 (GOOD_TILL_CANCEL)(sometimes: GTC).
15. Same fields should be filled if end user taps SELL button, when `pending order` has rate **lower** than the last bid rate.
16. When end user will tap BUY button, when `pending order` has rate **lower**, than the last ask rate: add into `NewOrderReq` next fields: `orderType` ENUM=2 (LIMIT), `limitPrice`=`pending order price`, `time in force` ENUM=2 (GOOD_TILL_CANCEL).
17. Same fields should be filled if end user taps SELL button, when `pending order` has rate **higher**, than the last bid rate.
18. Fields: `ctidTraderAccountId`, symbol id`, `trade side`, `volume`, `stop loss` (if any), `take profit` (if any`, `guaranteed stop loss` (if any)) and `trailing stop` (if any) will be also should be sent for execution of 'pending order`.



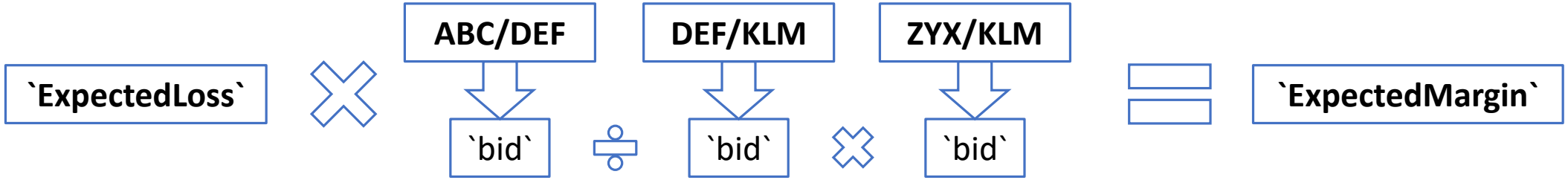
19. When user taps a "clock" icon – additional option of canceling pending order will be provided.
20. 'Buy/sell when rate is' will expand and open additional area with text: "Cancel your order by" with 2 fields: time and date choice.
21. By default open 'time' field with current time +1H and 'date' choice field with current date.
22. Pending orders ('order type` ENUM=2 &3: 'LIMIT' &'STOP') that have active time of cancelation must change 'timeInForce' param from ENUM=2 to ENUM=1 – GOOD_TILL_DATE.
23. Respectively, `Order.expirationTimestamp' should be added – that will be the time in `time` and `date` windows. In order to fill the appropriate field, please convert it to UNIX.
24. In case user sends `pending order` to execution, when "Cancel your order by" is activated – add change `timeInForce` and add `expirationTimestamp`; if the option was collapsed before sending the order for execution – proceed it as GTC orders, that were discussed previously.
25. When user taps HH:MM window – open device's clock setting (but allow changing HH:MM also with (+) and (-) buttons); when date is tapped – open device's date choice. Insert user's choice from both settings (or one, if for instance only time is changed) into "Sell/Buy when rate is" screen.
26. Make a validation that inserted time is no less than current +2 minutes (round the seconds till :00), if needed – adjust the values (in case user manually set other time or/and date). Run verification on every exit from device's time/date settings.

Limited Risk accounts

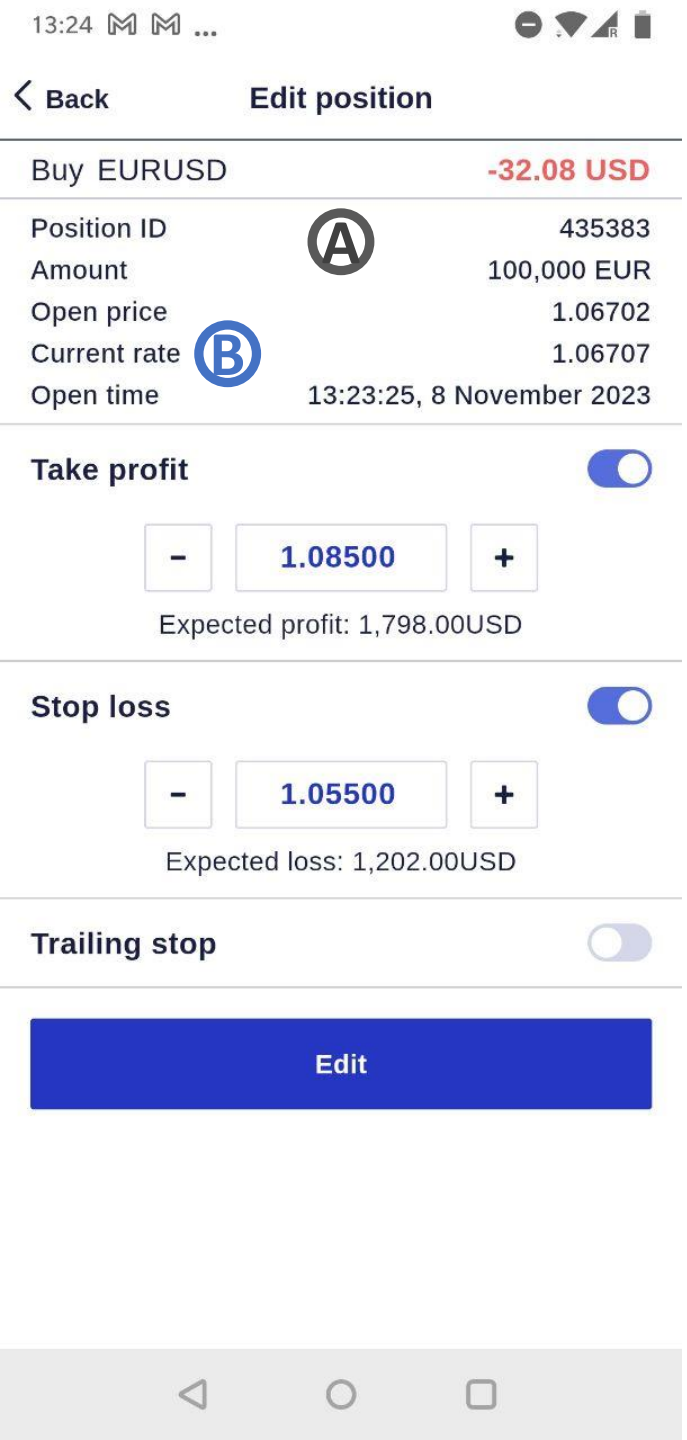
1. When `Trader.isLimitedRisk`=TRUE, the 2 immediate effects on sending orders for execution are:
2. Each order must have `stopLoss` or `trailingStop`, while its minimal value will be determined by `gslDistance` (instead of `slDistance`; also use value of `Symbol.distanceSetIn`).
3. Every `NewOrderReq`, `AmendOrderReq`, `AmendPositionSLTPReq` must mark `guaranteedStopLoss` field as =TRUE, otherwise the execution will be rejected.
4. When user with `LimitedRisk`=TRUE enters to Buy/Sell [symbol] screen, activate by default stop loss field and disable the radio button. Tap on `trailingStop` button will close `stopLoss` button+field, but later closing of `trailingStop` button+field will open `stopLoss` button+field (and color the button again in disabled color).
5. Insert in `sl field` rate that mirrors `gslDistance` X2 (or 10 pips, if `gslDistance`<5 pips); allow to reduce it till 1 `gslDistance` (or 5 pips), then disable +/-. As with `sl` – if rates changes against order's direction – update the minimal rate in `gsl` field.
6. If `Trader.limitedRiskMarginCalculationStrategy`=0 (by leverage) – expected margin will be returned in `ExpectedMarginReq`.
7. If `Trader.limitedRiskMarginCalculationStrategy`=1 (by GSL), `ExpectedMarginReq` will **still** return calculation by leverage. So, expected margin must be calculated on client. It's exactly the `expectedLoss` value, but it **must be converted** to `depositAsset`.
8. The most simple case of conversion will be no conversion: if traded symbol's `quoteAsset`=`depositAsset` - just copy `expectedLoss` value (and `assetName`) to `expectedMargin`.
9. Get `LightSymbol.quoteAssetId` of traded symbol (as `firstAssetId`) and `Trader.depositAssetId` (as 2nd) and send `SymbolsForConversionReq` for both. Response will return 1 or more symbols to complete the desired conversion rate. Subscribe to all symbols from response.
10. If one symbol was returned – check its `baseAsset` and `quoteAsset` order. If it’s equal to `firstAsset` and `secondAsset` in #9 – multiply `expectedLoss` by its `bid`, if the order is opposite – divide `expectedLoss` by its `bid`.

11. If there is more than one symbol returned in `SymbolsForConversionReq`, calculate like that: check whether traded symbol `quoteAsset` is base or quote asset in the first pair of returned `SymbolsForConversionReq`. If `base` – multiply `expectedLoss` by first `symbolForConversion` bid. If `quote` - divide `expectedLoss` by 1st `symbolForConversion` `bid`.
12. Now check the position (`base`/`quote`) of other symbol (non `quote` of traded symbol) in 2nd `SymbolForConversion`. If its the same (`quote` > `quote` / `base` > `base`) – multiply the result of #10 by 2nd `SymbolForConversion` `bid`. If not the same – divide.
13. Continue with the same way for 3rd + `SymbolForConversion`, checking every time non-used in previous calculation asset's position.

Expected margin calculation for
`quoteAsset` of traded symbol=**ABC** `depositAsset` of account=**ZYX** `SymbolsForConversionRes`: **ABC/DEF, DEF/KLM, ZYX/KLM**

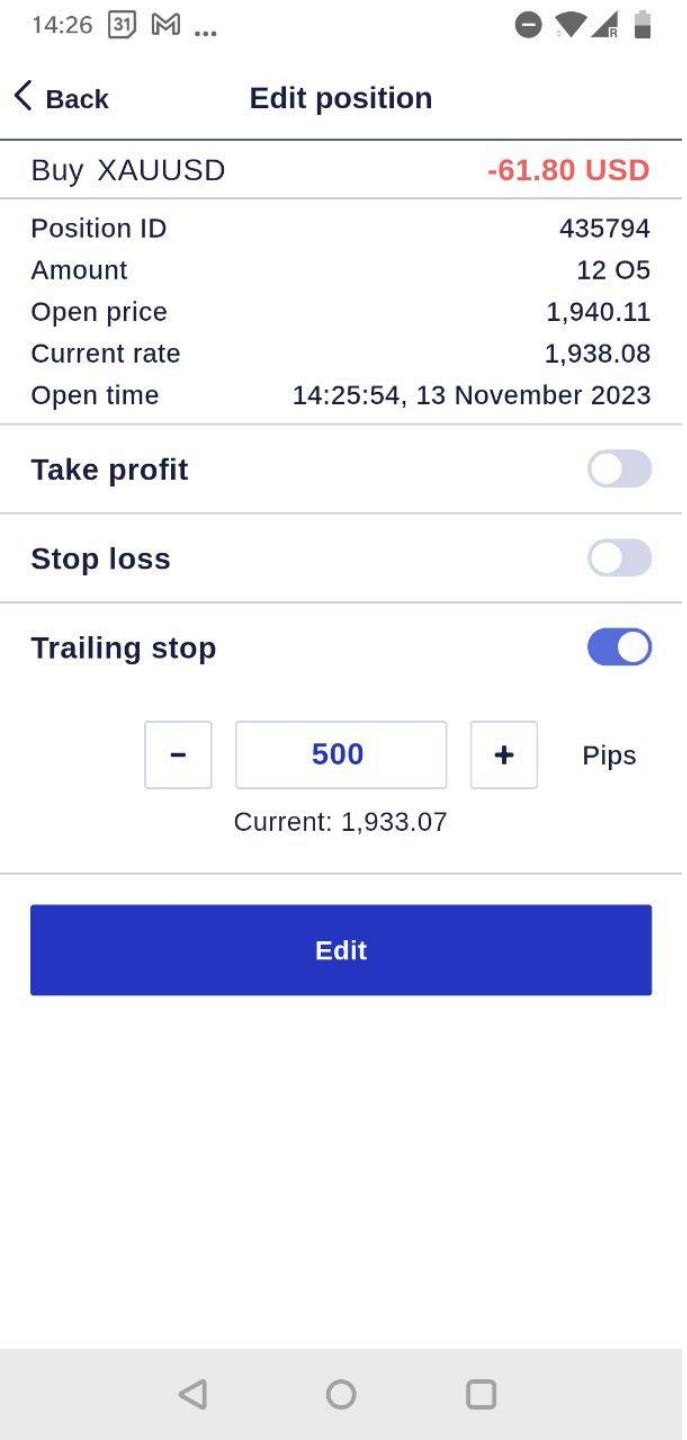


14. If `Trader.limitedRiskMarginCalculationStrategy`=2 (by leverage and by GSL), calculate `by gsl` (#8-13), then request `ExpectedMarginReq` and show **the biggest** of two. Remember that `ExpectedMarginRes` will be already converted to `depositAsset`.
15. When `Trader.isLimitedRisk` and `limitedRiskMarginCalculationStrategy`=1,2 – use `SUM` when calculating **account's** margin. Opened position's exact margin will be returned in `Position.usedMargin`.
16. Related to LimitedRisk accounts there is another verification that should be done: when pending order is created, verify that `price`>`gslDistance`, otherwise user won't be able to execute given order. Disable (-) button when this condition is reached.
17. Round to given value on keyboard's collapse, if user used keyboard to type a smaller value.



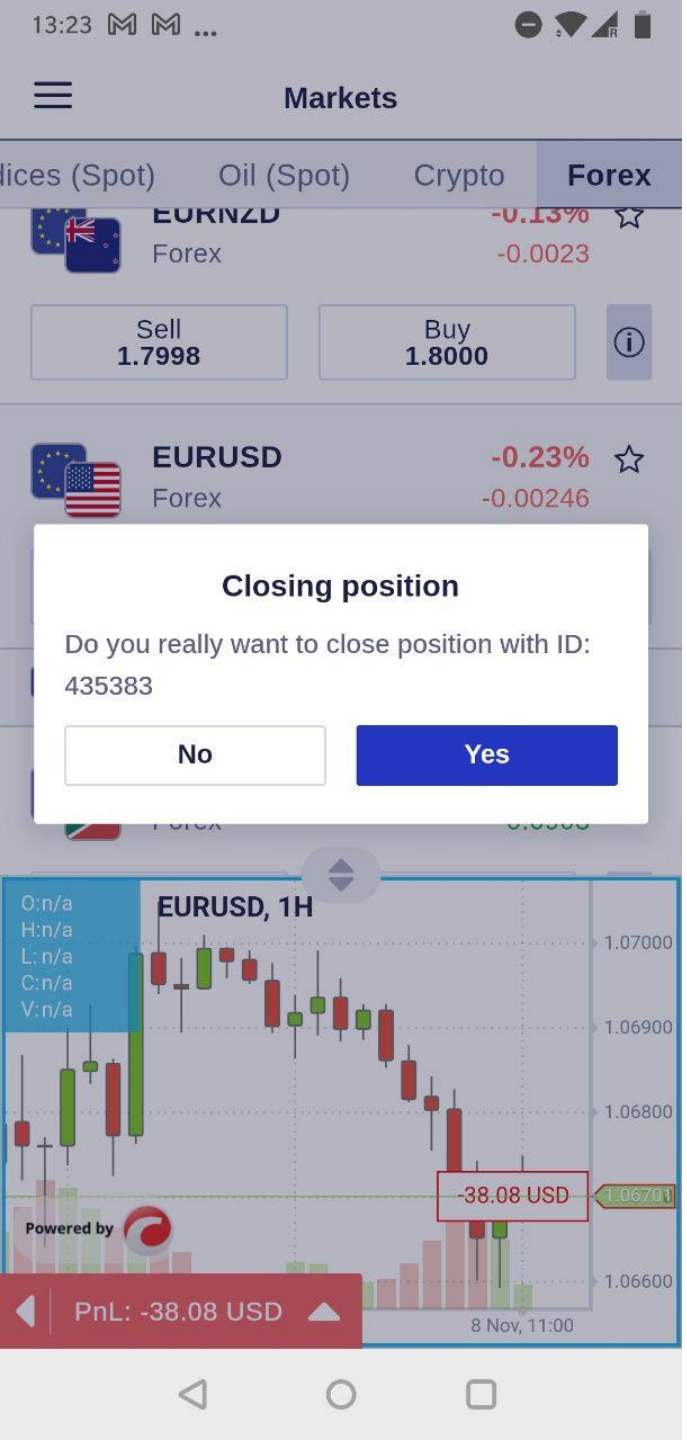
Editing positions

1. If user taps on 'edit position' icon, either in 'position line:markets' or in 'My activity: positions' – he must be redirected to 'Edit position' screen. This screen will include some basic (and unchanged) data of given position, when editable params (tp/sl/ts) will be available for change.
2. Besides fields from "My activity: positions" screen Ⓐ, there will be another line: current rate.Ⓑ
3. The order of fields: Header, Position ID, Amount, Open rate, Current rate, Open time.
4. "Current rate"….[ProtoOASpotEvent.ask/bid] (the **opposite** rate to position's 'side')
5. User will be able to edit take profit, stop loss and trailing stop (**Setting TP or/and SL** rules will still apply, like 'tp/sl distance'; accelerated hold on (+)(-) buttons of tp/sl/ts).
6. User also will be able to switch from sl to ts and vice versa, while the limit for it won't be `Position.price`, but current `bid` and `ask` (depends on position's `tradeSide`).
7. Lower button will be named "EDIT" and will send `ProtoOAAmendPositionSLTPReq` with all the new params. If params were not changed, "EDIT" will be inactive by default, till some value will be changed. Please note that in `AmendPositionSLTPReq` `takeProfit` and `stopLoss` values will be sent through appropriately named fields and **not** `relativeTakeProfit` or `relativeStopLoss` fields. E.g. insert the exact rates of sl/gsl/tp.
8. If user disabled either TP or SL after it was accepted (deleting TP or SL) – send `AmendPositionSLTPReq` with empty field for appropriate parameter.
9. Response will be returned in `ProtoOAExecutionEvent`, that will confirm that new sl/tp/ts were accepted on server. Confirmation popups will be discussed after `ExecutionEvents` section.



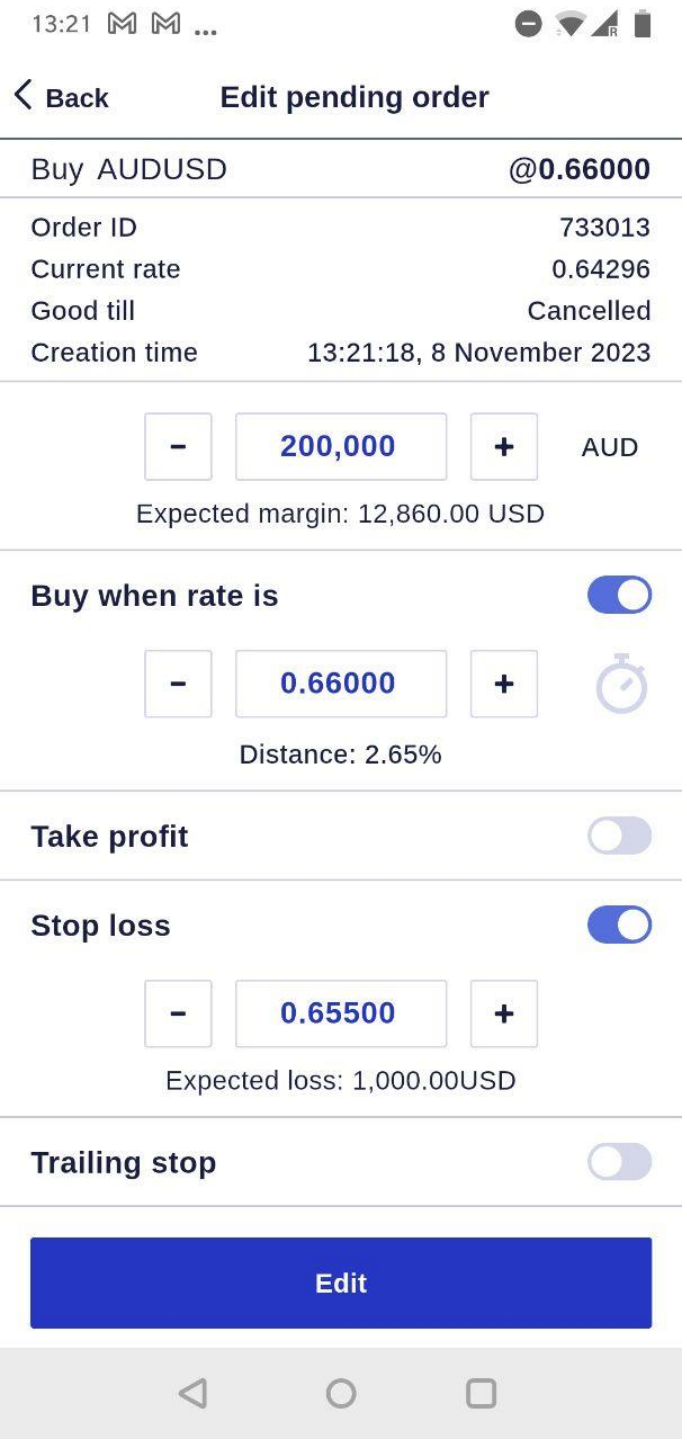
Editing trailing stop in positions

1. For trailing stop appropriate presentation/editing – on app's start handle more events: `TrailingSLChangedEvent`. It updates rates of active trailing stops, if any, for given account.
2. If any order from`ProtoOAReconcile` has 'trailingStopLoss`=TRUE, the current value of trailingStop will be in `Order.stopLoss` field. Later it will be updated through event #1.
3. If user taps in 'edit position' icon and given position has a trailing stop loss, go to app's cache and find the value in pips related to given `orderId`/`positionId`.
4. Show this value in `trailingStop` window and show below trailingStop window additional data (as `expectedLoss/profit` shown below `stopLoss`/`takeProfit` windows)–"Currently:".
5. Show "Currently:" (value of 'stopLoss` from `ReconcileRes` or `TrailingSLChangedEvent`).
6. If user didn't change trailing stop, but changed `takeProfit` and tapped "EDIT" - do not send any new value in `AmendPositionSLTPReq`, so existing trailing stop won't change.
7. If user changed TS – change "Currently:" to "Expected:" and deduct pips converted to rate from `position.price` for BUY positions / add pips converted to rate to `position.price` for SELL positions. Please note, that pips value can be negative, in case position in big profit.
8. Disable +\- buttons whether "expected" > 0.00000 or approaches current rate (`ask` for Buy positions and `bid` for Sell positions) up to (`sl/gslDistance`+spread).
9. If "EDIT" tapped add rate from #8 to `AmendPositionSLTP.stopLoss` and `trailingSL`=TRUE.
10. If trailing stop value is tapped back to orig. value – change text back to "Currently:" and rate – to rate from `ReconcileReq` or latest `TrailingSLChangedEvent'.
11. If in app's cache there is no pips value for some position (e.g. position was NOT opened in this app), show N/A in trailingStop window, but with appropriate value below. If user taps on +/- then – show value from **"Setting trailing stop", #3,4**. If done, user can't get back to orig., so he must leave the screen w/o tapping EDIT to keep the current state.



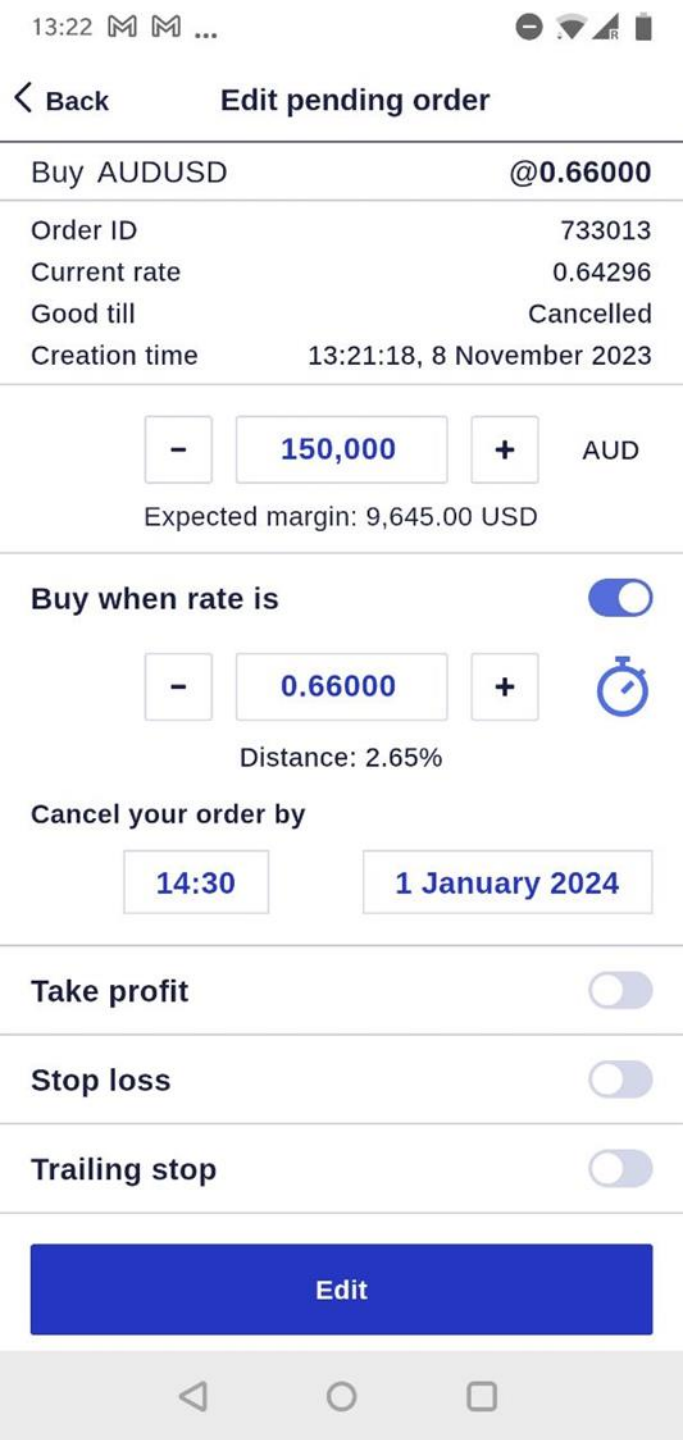
Closing positions manually

1. If user tapped 'delete' icon (either in 'Markets' or 'My activity' screen) – the position will be closed after user's final approval.
2. Show the next popup - header: "Closing position", body: "Do you really want to close position ID:['position id']?"
3. Below buttons "No" and "Yes" will appear. Tapping on "No" will close the popup w\o taking any action, while tapping "Yes" will send `ProtoOAClosePositionReq`, with appropriate `positionId` and its `volume`.
4. As soon as `ProtoOAExecutionEvent` for given position has returned, show user the appropriate popup from `Execution events` section.
5. If execution is rejected due to cServer failed validation – the `ProtoOAOrderErrorEvent` will be sent, show popup with header: "An error occurred", in body add the description that appears in event.
6. If user is trading in active "simultaneous trading" mode (check: **Simultaneous trading** #8,9), more popups might be required, more details will be provided.

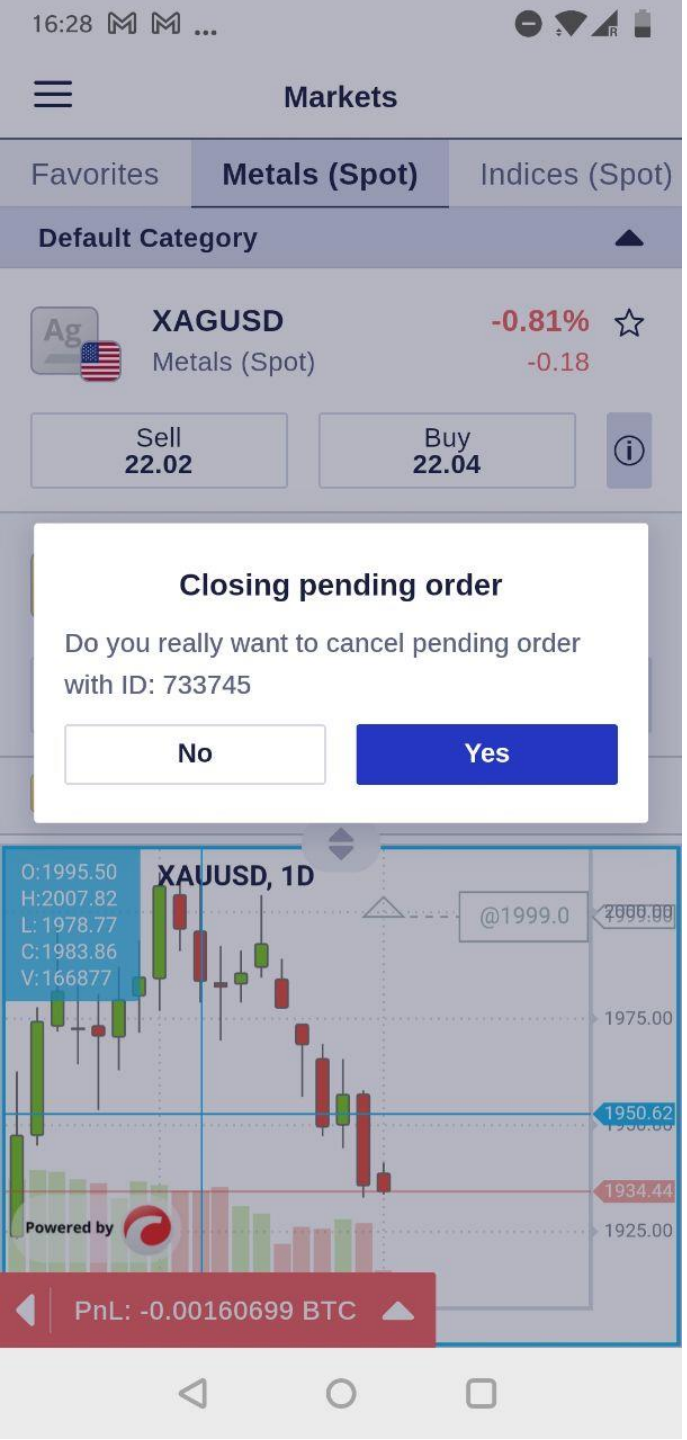


Editing pending orders

1. If user taps in 'edit pending order' icon, either in 'pending order line:markets' or in 'My activity: pending orders' – he must be redirected to 'Edit pending order' screen.
2. This screen will include some basic (and unchangeable) data of given pending order, when editable params (amount/rate/cancelation time/tp/sl/ts) will be available to change.
3. Some fields from "My activity: pending orders" screen will appear here, too.
4. User will be able to edit amount, take profit, stop loss and trailing stop (**Setting Take profit or/and Stop loss** rules will still apply, like 'tp/sl distance'; accelerated tap and hold on (+)(-) buttons of tp/sl/ts will also be available).
5. User also will be able to switch from tp/sl to ts and vice versa.
6. In case "amount" is unchanged – **don't** sent value in `AmendOrderReq.volume`.
7. User will be able to edit the rate of execution (Buy/Sell when rate is) with one condition:
8. If the rate stays from the same side (for example: execution bid should be above current bid) – the order will be edited through `ProtoOAAmendOrderReq`, while all appropriate fields get new values.
9. Execution can be either rejected by ProtoOAOrderErrorEvent or accepted by ExecutionEvent, which will be discussed later.



10. If the rate changes side (for example: execution ask was above current ask, but now user changes it to be below current ask, e.g. 'order type' is changing from 'stop' to 'limit') – current order should **be canceled** and new order should be opened.
11. If that's the case on user's tap on EDIT button, show next popup - header: Attention!, body: This action requires a new order. New order will be sent to the market only after cancellation of pending order ID: ['order id']."
12. If approved – send two consecutive commands: `ProtoOACancelOrderReq` (deleting current order) and after approval through `ProtoOAExecutionEvent`, send new `NewOrderReq` with appropriate details: change `orderType` from `STOP` to `LIMIT` or from `LIMIT` to `STOP`, depends on user's action. Leave the rest of appropriate fields as they are.
13. Response will be returned in `ProtoOAExecutionEvent`, too, that will confirm that new order was accepted. Exact details/flows will be explained in Execution events section.
14. If confirmed properly – show popup according to rules in Execution events section.
15. In case of simultaneous trading mode was activated – more popups might be required to be shown. More details are in (**Simultaneous trading**, #8,9).
16. If no param was changed, keep "EDIT" button inactive, while making it active only after at least one of params was changed.



Canceling pending orders

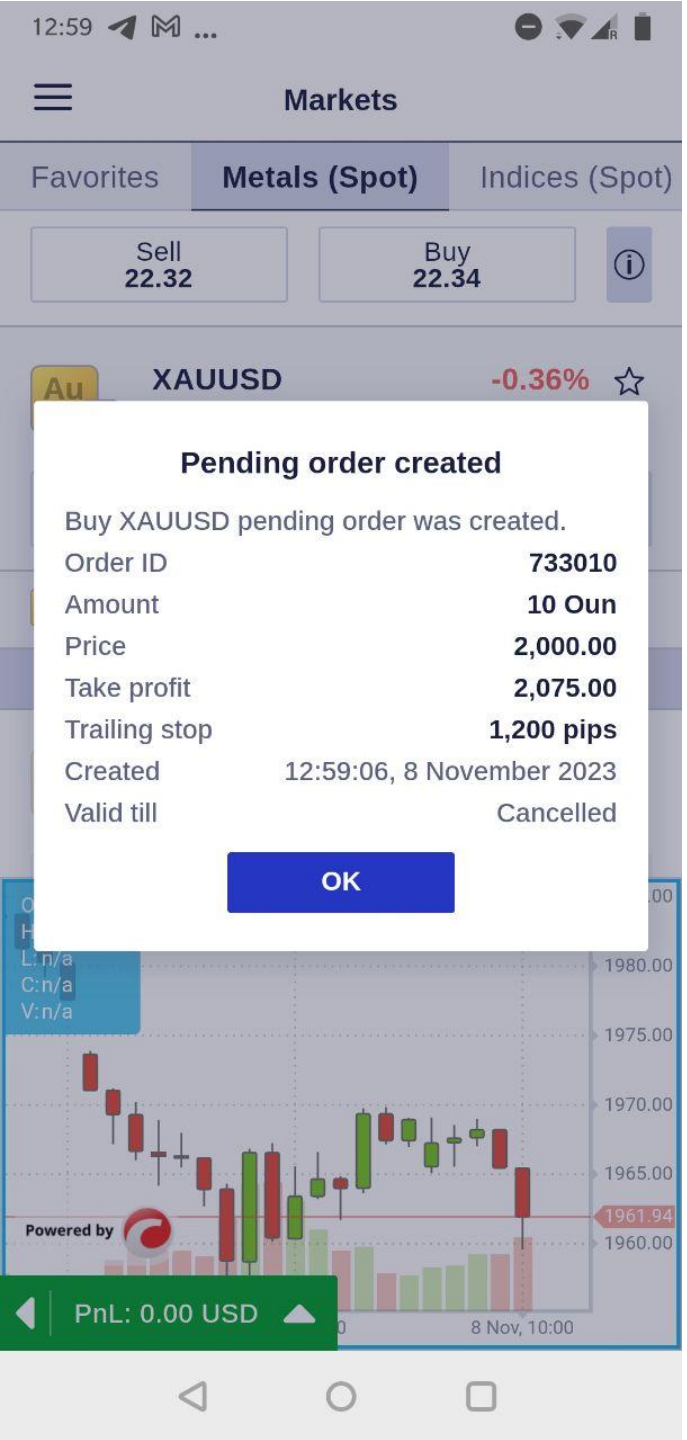
1. If user tapped 'cancel' icon (either in 'Markets' or 'My activity' screen) – the pending order will be canceled after user's final approval.
2. Show the next popup - header: "Cancel pending order", body: "Do you really want to cancel pending order ID:['order id']?"
3. Below buttons "No" and "Yes" will appear. Tapping on "No" will close the popup w\o taking any action, while tapping on "Yes" will send `ProtoOACancelOrderReq` with appropriate `orderId` to cancel it.
4. The response might be returned through `OrderErrorEvent` or `ExecutionEvent`. Appropriate details/flows will appear in further section about Execution Events.
5. If given user has activated "simultaneous trading" mode – there might be more popups to show; more details will be provided later.

Updates through events

1. Some data can be updated by given broker's staff during trading session. These changes can effect end user's trading process and therefore the appropriate events must be processed on client.
2. `ProtoOASymbolChangedEvent` - In case some details of symbol entity were changed, a `SymbolChangedEvent` will be sent. It will provide appropriate `symbolId`. In that case an immediate request of appropriate `LightSymbol` entity must be done and in case that during given session `SymbolByIdReq` was called (and is kept in cache) – send this request, too and update all the changes.
3. `ProtoOATraderUpdatedEvent` - If some changes were made to `Trader` entity – given event will be sent. In event's body there will be updated `ProtoOATrader` entity – update all the changed fields immediately.
4. `ProtoOAMarginChangedEvent` - in case there is a new margin requirement for some position – given event will be sent. It will include `positionId` of appropriate position and new `usedMargin` (and `moneyDigits`) for this position. If given event is accepted – account's margin must be recalculated immediately.

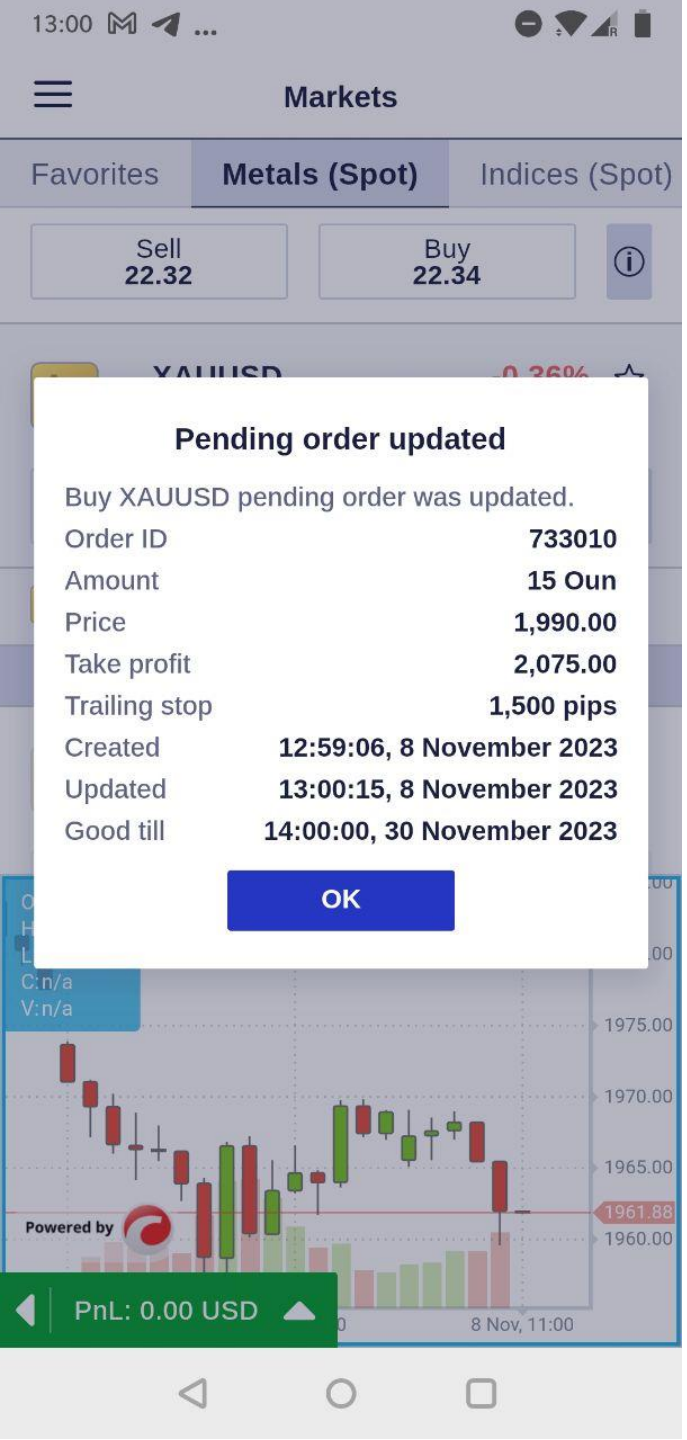
Orders execution: `OrderError` and `Execution` events

1. When any order is sent to system, it goes through 2 main stages:
2. a) validation on cTrader's side.
3. b) execution on the market/liquidity provider side.
4. In case order is failed on cTrader's side – cTrader's backend sends `ProtoOAOrderErrorEvent`. In that case show a popup with header: An error occurred" and body: Error description: `OrderErrorEvent.description`. OK button will close the popup.
5. If order passed cTrader's validations – it will be forwarded to execution or await to execution's time (in case of pending orders). From now and on any action regarding given order will be provided through `ProtoOAExecutionEvent`. In this case (passing cTrader's validation) each order will be updated through execution event with executionType=2 >> accepted.
6. After order's execution/partial execution/cancellation or rejection OA product will be updated through an appropriate execution event. All types of possible combinations of execution type and order type appear in attached .xls file.
7. Some combinations of execution type and order type have more than 1 possible outcome (like: position was opened, position was increased). The necessary conditions for each scenario verification are also added in .xls file.
8. After scenario is defined: the appropriate popup should be shown to end user.
9. On the next pages all possible popup types are presented, with appropriate fields and source of data for these fields.



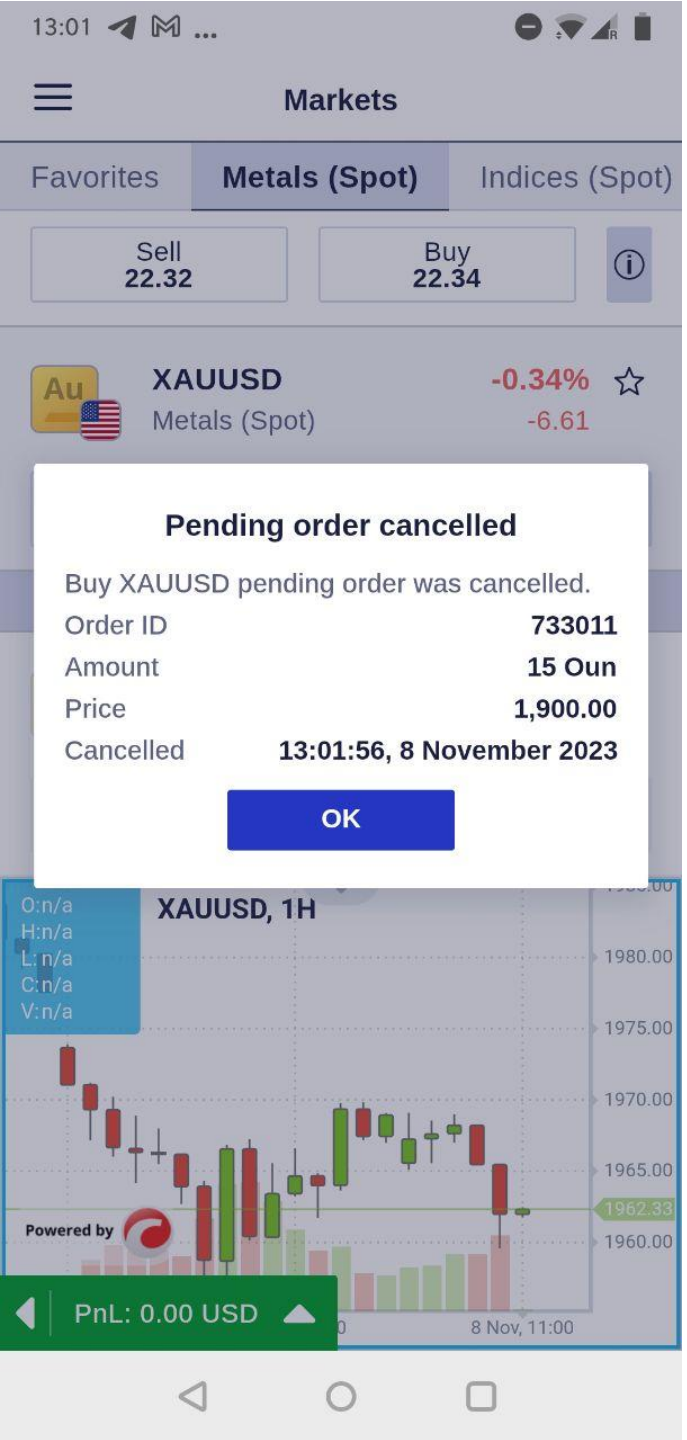
Execution popups: pending order created

1. H: "Pending order created"; B: "Buy/Sell [symbolName] pending order was created".
2. Direction (Buy/Sell): - `order.tradeData` >> `tradeSide`
3. "Order ID:" - `order.orderId`
4. "Amount:" - `order.tradeData` >> `volume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
5. "Price:" - either `order.limitPrice`/`order.stopPrice`, depends what is field is filled.
6. "Take profit:" - `stopPrice`/`limitPrice` (depends on order type) +/- (depends on order's `tradeSide`) `order.relativeTakeProfit`/100,000. If `relativeTP`=empty/NaN – show "No".
7. "Stop Loss:" - if `relativeStopLoss`=value and `order.tralinigStopLoss`=FALSE >> `stopPrice`/`limitPrice` (depends on order type) -/+ (depends on order's `tradeSide`) `order.relativeStopLoss`/100,000. If `relativeStopLoss`=empty/NaN – show "No".
8. "Trailing Stop:" - if `relativeStopLoss`=value and `order.tralinigStopLoss`=TRUE >> `relativeStopLoss`/100,000 * $10^{Symbol.pipPosition}$, add "pips". Place **instead** of "Stop Loss".
9. "Created:" - `order.tradeData` >> `openTimestamp` >> convert it to local time in HH:MM DD:MM:YY format.
10. "Valid till:" - If `order.expirationTimestamp`=value >> convert it to local time in HH:MM DD:MM:YY format; if =empty/ NaN >> "Cancelled".
11. Button OK will close the popup.
12. As soon as appropriate event/s accepted – add given pending order on `Markets` and `My activity:orders` screens.



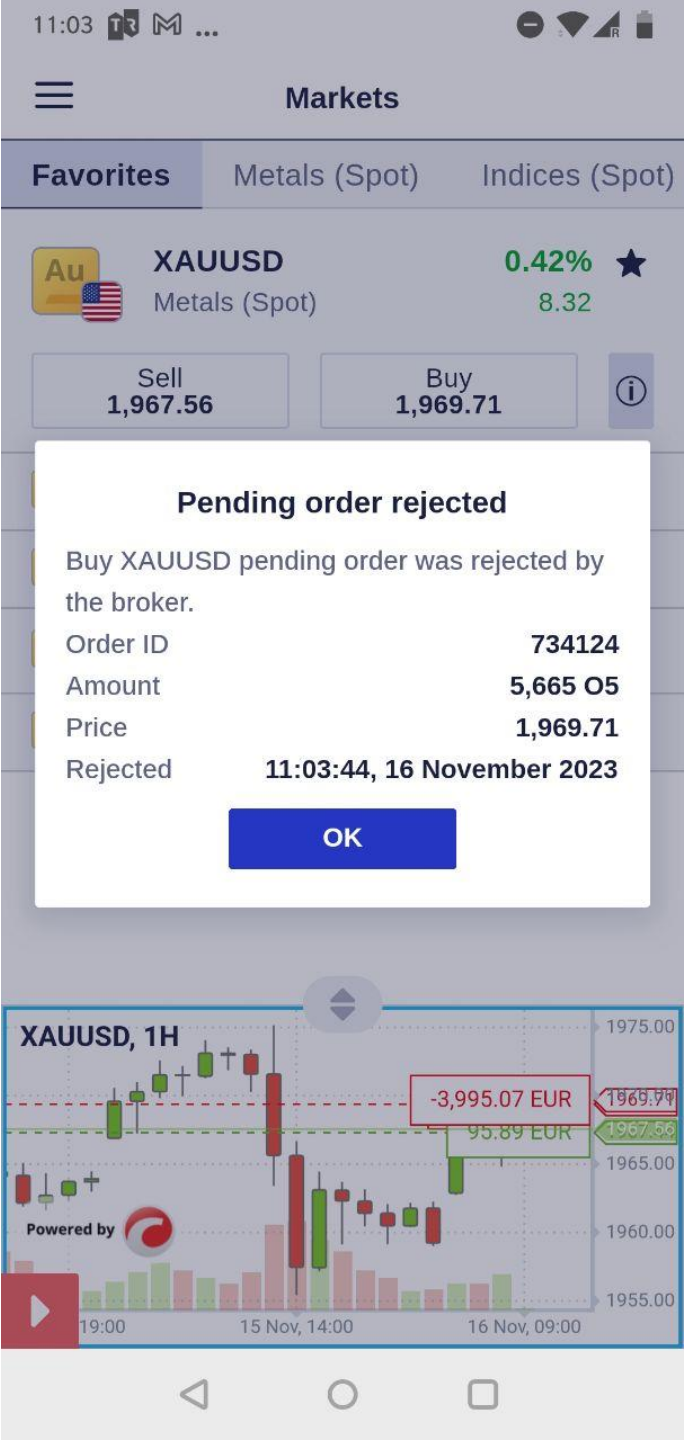
Execution popups: pending order updated

1. H: "Pending order updated"; B: "Buy/Sell [symbolName] pending order was updated".
2. Direction (Buy/Sell): - `order.tradeData` >> `tradeSide`
3. "Order ID:" - `order.orderId`
4. "Amount:" - `order.tradeData` >> `volume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
5. "Price:" - either `order.limitPrice`/`order.stopPrice`, depends what is field is filled.
6. "Take profit:" - `stopPrice`/`limitPrice` (depends on order type) +/- (depends on order's `tradeSide`) `order.relativeTakeProfit`/100,000. If `relativeTP`=empty/NaN – show "No".
7. "Stop Loss:" - if `relativeStopLoss`=value and `order.tralinigStopLoss`=FALSE >> `stopPrice`/`limitPrice` (depends on order type) -/+ (depends on order's `tradeSide`) `order.relativeStopLoss`/100,000. If `relativeStopLoss`=empty/NaN – show "No".
8. "Trailing Stop:" - if `relativeStopLoss`=value and `order.tralinigStopLoss`=TRUE >> `relativeStopLoss`/100,000*$10^{\text{Symbol.pipPosition}}$, add "pips". Place **instead** of "Stop Loss".
9. "Created:" - `order.tradeData` >> `openTimestamp` >> convert it to local time in HH:MM DD:MM:YY format.
10. "Updated:" - `order.utcLastUpdateTimestamp` >> convert it to local time in HH:MM DD:MM:YY format.
11. "Good till:" - If `order.expirationTimestamp`=value >> convert it to local time in HH:MM DD:MM:YY format; if =empty/ NaN >> "Cancelled".
12. Button OK will close the popup.
13. As soon as appropriate event/s accepted – update given pending order's details on `Markets` and `My activity:orders` screens.



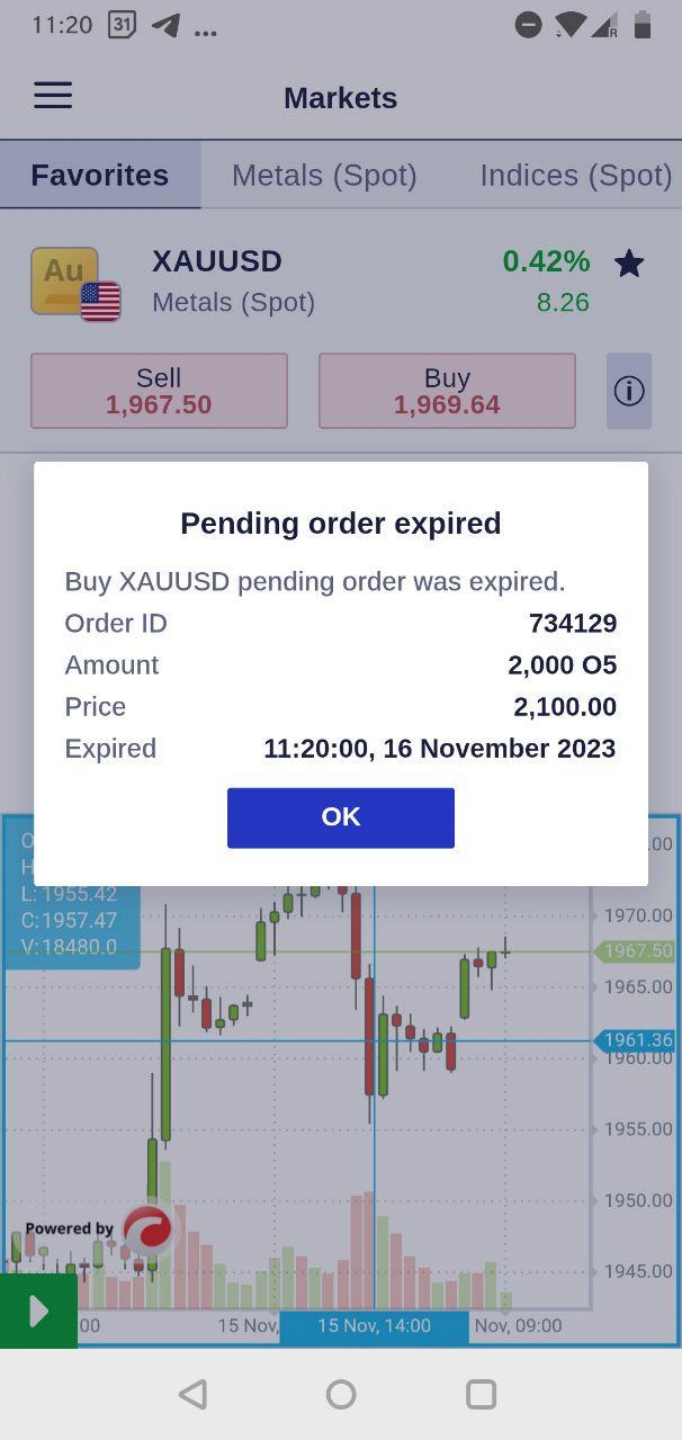
Execution popups: pending order cancelled

1. H: "Pending order cancelled"; B: "Buy/Sell [symbolName] pending order was cancelled".
2. Please note that `executionType`=5 can be initiated also by server; **still** show this popup with same fields.
3. Direction (Buy/Sell): - `order.tradeData` >> `tradeSide`
4. "Order ID:" - `order.orderId`
5. "Amount:" - `order.tradeData` >> `volume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
6. "Price:" - either `order.limitPrice`/`order.stopPrice`, depends what is field is filled. Or expect it by details of appropriate request.
7. "Cancelled:" - ` order.utcLastUpdateTimestamp ` >> convert it to local time in HH:MM DD:MM:YY format.
8. **If `Order.executedVolume` field is not empty**, change the body to "Buy EURUSD pending order was cancelled after partial execution" and in "Amount" deduct `order.executedVolume` from `order.tradeData.volume`.
9. Button OK will close the popup.
10. **As soon as appropriate event/s accepted – delete given pending order from `Markets` and `My activity:orders` screens.**



Execution popups: pending order rejected

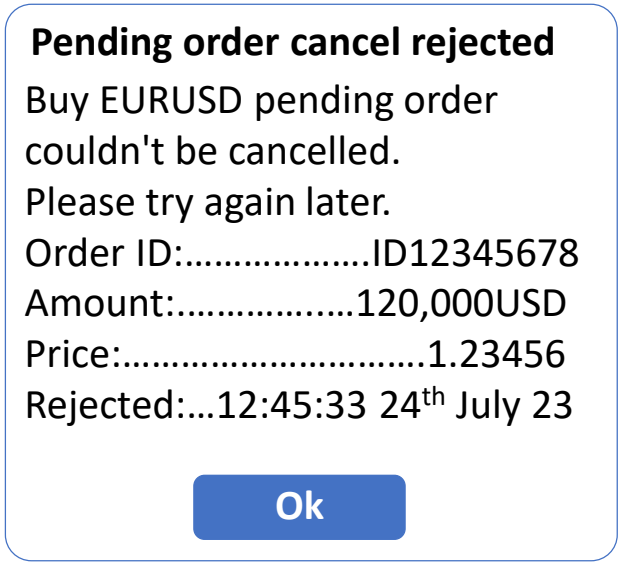
1. Direction (Buy/Sell): - `order.tradeData` >> `tradeSide`
2. "Order ID:" - `order.orderId`
3. "Amount:" - `order.tradeData` >> `volume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
4. "Price:" - either `order.limitPrice`/`order.stopPrice`, depends what is field is filled. Or expect it by details of appropriate request.
5. "Rejected:" - ` order.utcLastUpdateTimestamp ` >> convert it to local time in HH:MM DD:MM:YY format.
6. Button OK will close the popup.
7. As soon as appropriate event/s accepted – (if given pending order was on `Markets` and `My activity:orders` screens) delete it.



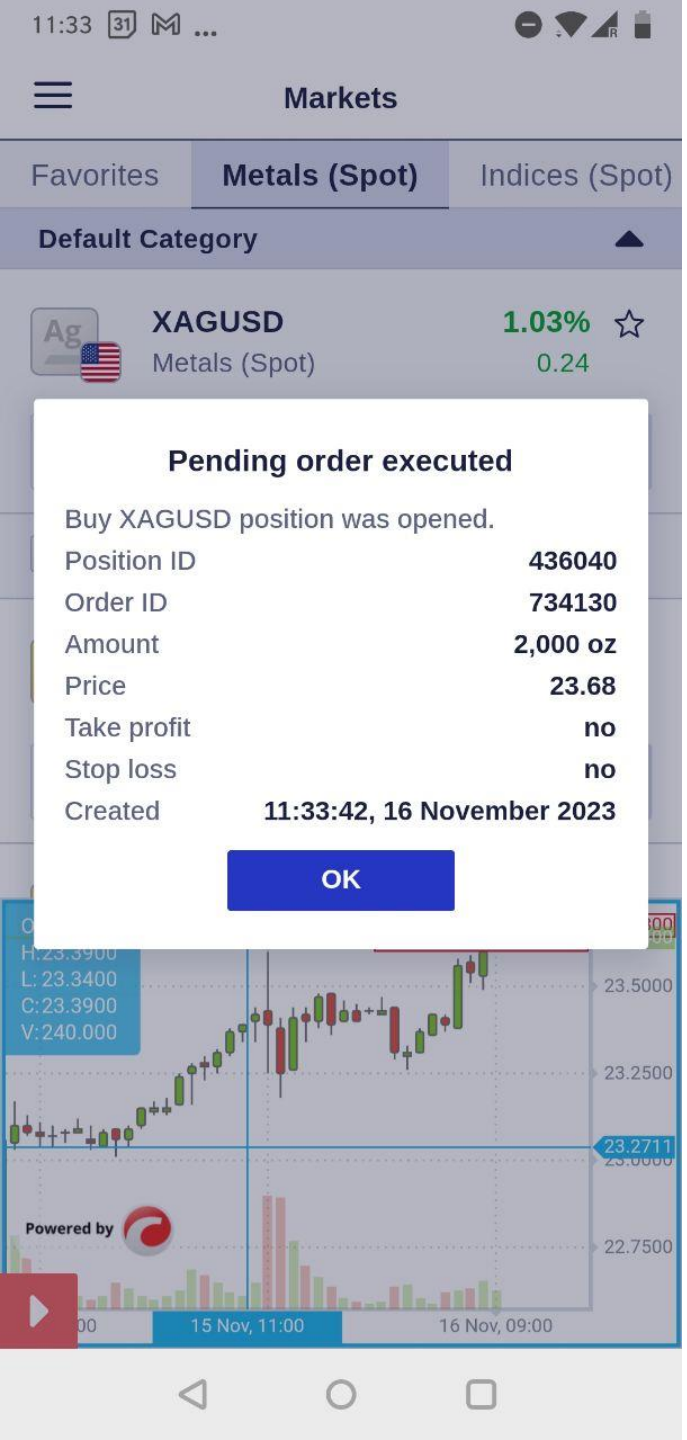
Execution popups: pending order expired

1. Direction (Buy/Sell): - `order.tradeData` >> `tradeSide`
2. "Order ID:" - `order.orderId`
3. "Amount:" - `order.tradeData` >> `volume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
4. "Price:" - either `order.limitPrice`/`order.stopPrice`, depends what is field is filled. Or expect it by details of appropriate request.
5. "Expired:" - ` order.utcLastUpdateTimestamp ` >> convert it to local time in HH:MM DD:MM:YY format.
6. Button OK will close the popup.
7. As soon as appropriate event/s accepted – delete given pending order from `Markets` and `My activity:orders` screens.

Execution popups: pending order cancel rejected

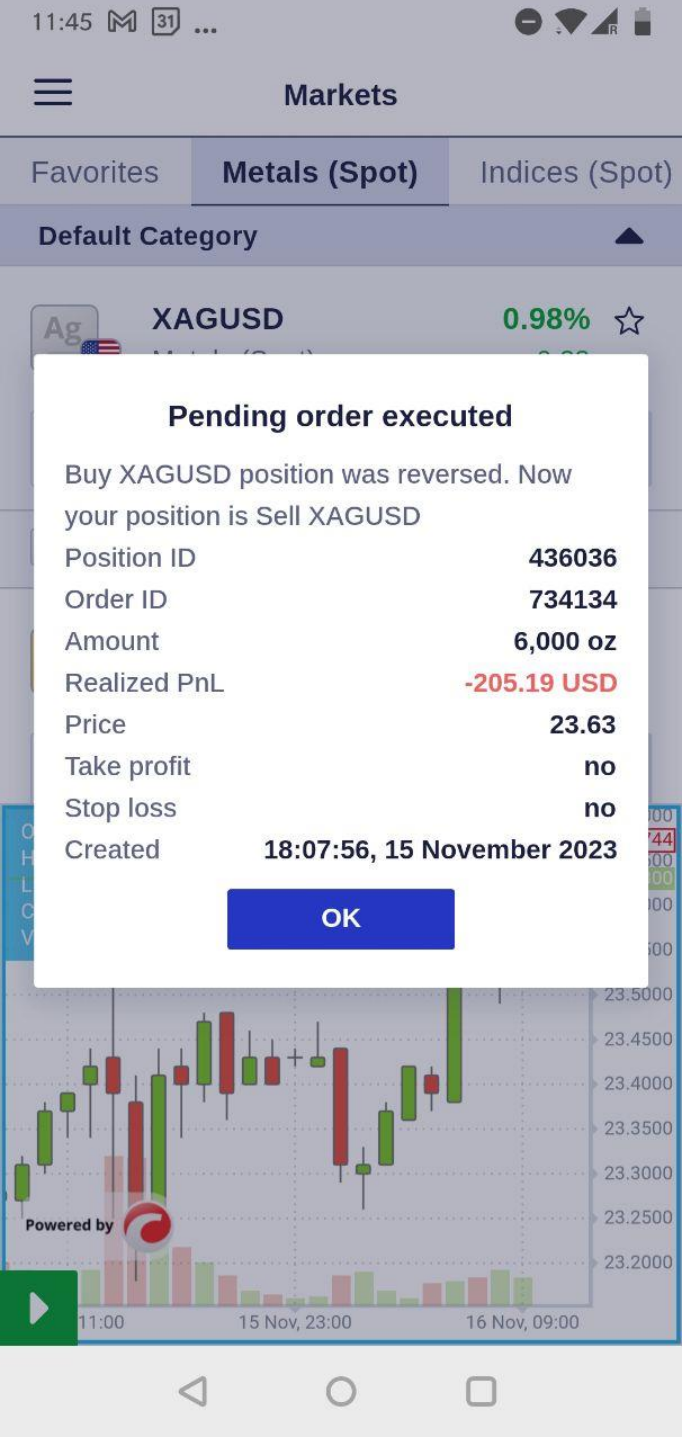


1. Direction (Buy/Sell): - `order.tradeData` >> `tradeSide`
2. "Order ID:" - `order.orderId`
3. "Amount:" - `order.tradeData` >> `volume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
4. "Price:" - either `order.limitPrice`/`order.stopPrice`, depends what is field is filled. Or expect it by details of appropriate request.
5. "Rejected:" - "Rejected:" - ` order.utcLastUpdateTimestamp ` >> convert it to local time in HH:MM DD:MM:YY format.
6. Button OK will close the popup.



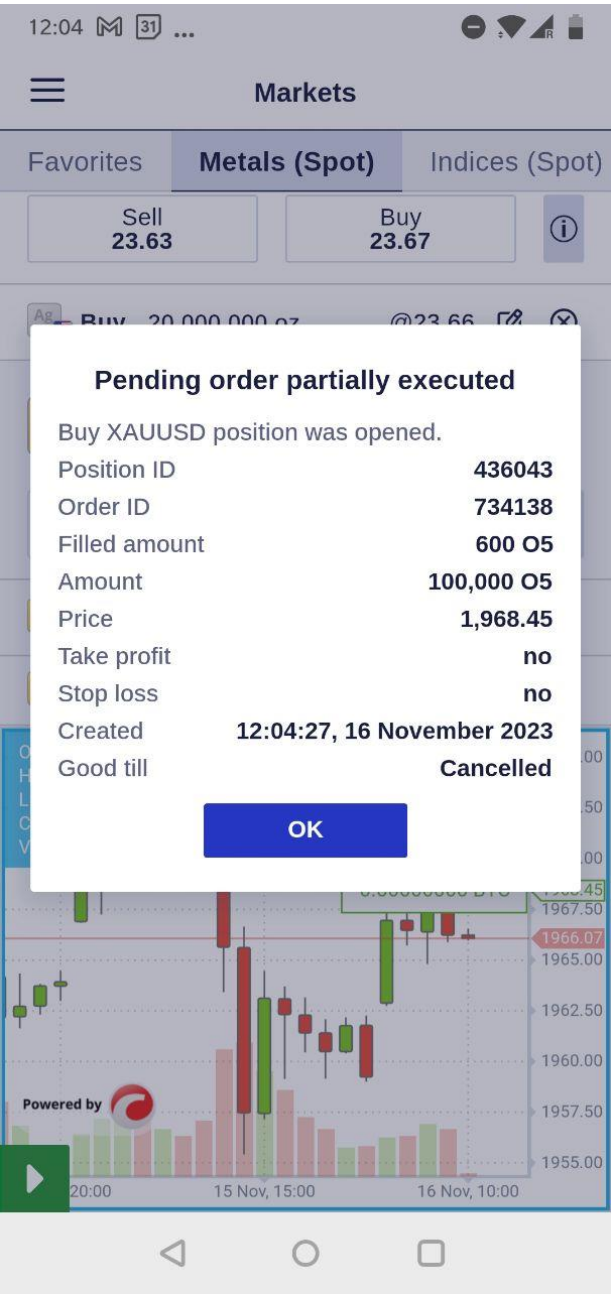
Execution popups: pending order executed/executed w/o SLTP/partially executed

1. This popup template has same header, but different bodies; it includes "Buy EURUSD position was **opened/increased/decreased/reversed/closed**", according to conditions in .xls file.
2. Direction (Buy/Sell): - `order.tradeData` >> `tradeSide`; if "reversed" – use **the opposite** side.
3. "Position ID:" - `position.positionId`
4. "Order ID:" - `order.orderId`
5. "Amount:" - `position.tradeData` >> `volume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
6. "Price:" - `position.price`.
7. **For `NETTED` accounts only**: if related order in `executionEvent.type`=3 has relativeStopLoss/ TakeProfit`=empty (e.g. executionEvent.type=3 should NOT be verified by `accepted` or `replaced` SLTP event) get SL/TP/TS by from `position.takeProfit`/`stopLoss`/`trailingStop` fields of position in same event. Process the results by same rules of next three paragraphs.
8. "Take profit:" - `position.takeProfit` of **the following** `executionEvent` with type=2. If `takeProfit`=empty/NaN – show "No".
9. "Stop Loss:" - `position.stopLoss` of **the following** `executionEvent` with type=2. If `stopLoss`=empty/NaN – show "No".
10. "Trailing Stop:" - if `position.stopLoss` of **the following** ExecutionEvent=value and `position.tralinigStopLoss`=TRUE >> |`position.price` -`position.stopLoss`|/$10^{\text{Symbol.pipPosition}}$, add "pips". Place **instead** of "Stop Loss".

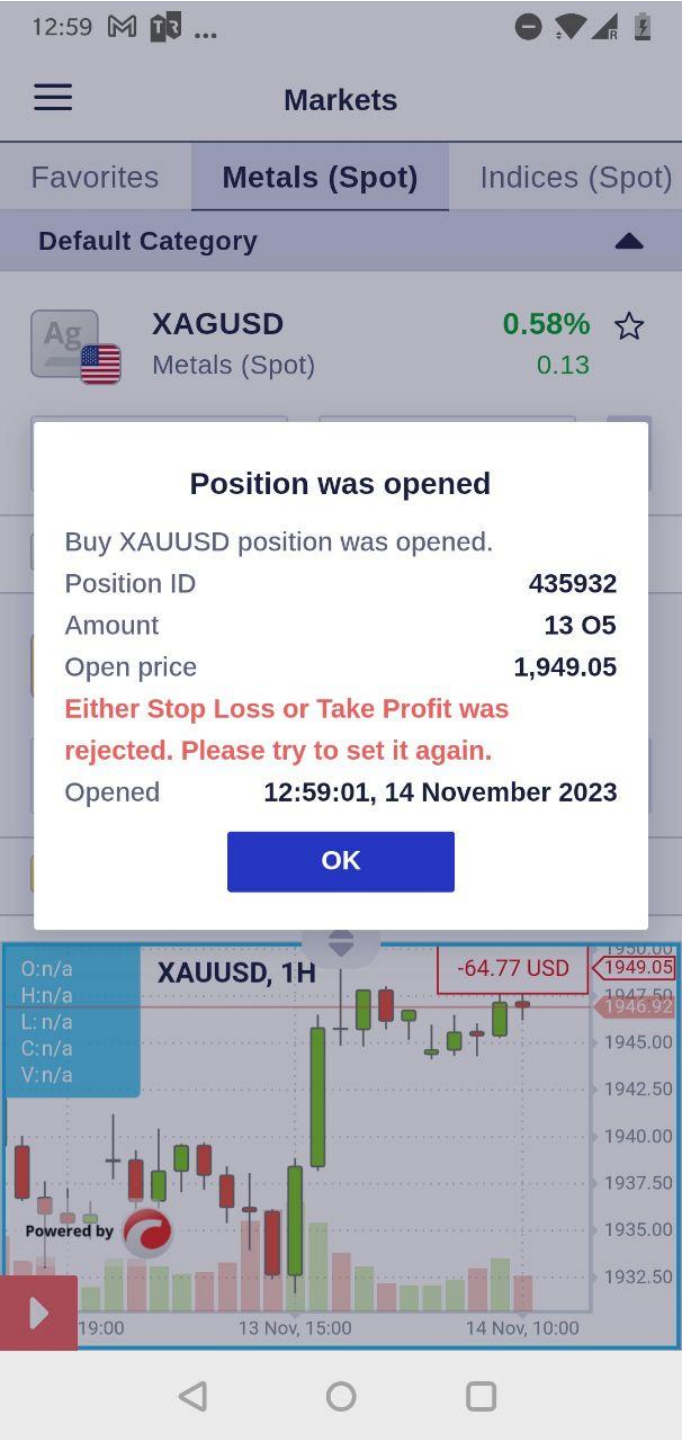


11. "Created:" - `position.tradeData.openTimestamp` (of executionEvent.type=3) >> convert it to local time in HH:MM DD:MM:YY format.
12. Button OK will close the popup.
13. **As soon as appropriate event/s accepted – add/update new position's details on `Markets` and `My activity:orders` screens.**
14. During execution/partial execution popups, when position is decreasing/closed/reversed, as a result of this action some PnL will be realized and should be resented to end user in popup's details.
15. To get knowledge whether there is a realized PnL: if in appropriate `ExecutionEvent.deal` >> `closePositionDetail`=value (should be expected in decreased/closed or reversed position), there will be realized PnL.
16. Add it below "Filled amount" with header: "Realized PnL".
17. "Realized PnL:" - (`deal.closePositionDetail.grossProfit`+`swap`+`commission`)/100; with value of `depositAsset`; if=>0 - paint the PnL value in green, if <0 – in red.

Execution popups: pending order partially executed



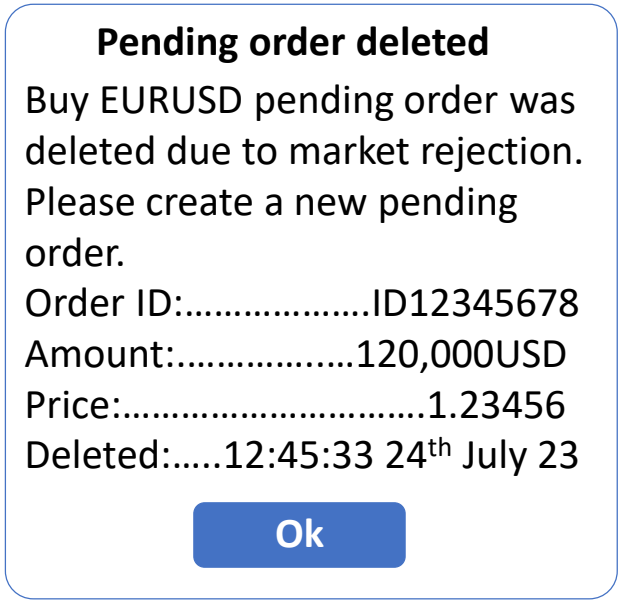
1. Popup template has same header, but different bodies; it includes "Buy EURUSD position was **opened/increased/decreased/reversed/closed**", according to conditions in .xls file.
2. Direction (Buy/Sell): - `order.tradeData` >> `tradeSide`; if "reversed" – use opposite side.
3. "Position ID:" - `position.positionId`
4. "Order ID:" - `order.orderId`
5. "Filled amount:" - `order.executedVolume`.
6. "Amount:" - `order.tradeData` >> `volume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
7. "Price:" - `position.price`.
8. **For `NETTED` accounts only**: if related order in `executionEvent.type`=3 has relativeStopLoss/TakeProfit`=empty (executionEvent.type=3 won't be verified by `accepted` or `replaced` SLTP event) get SL/TP/TS from `position.takeProfit`/`stopLoss`/`trailingStop` fields of position in same event. Process the results by same rules of next three paragraphs.
9. "Take profit:" - `position.takeProfit` of **the following** `executionEvent` with type=2. If `takeProfit`=empty/NaN – show "No".
10. "Stop Loss:" - `position.stopLoss` of **the following** `executionEvent` with type=2. If `stopLoss`=empty/NaN – show "No".
11. "Trailing Stop:" - if `position.stopLoss` of **the following** ExecutionEvent=value and `position.tralinigStopLoss`=TRUE >> |`position.price` -`position.stopLoss`|/$10^{\text{Symbol.pipPosition}}$, add "pips". Place **instead** of "Stop Loss".
12. "Created:" - `position.tradeData.openTimestamp` (of executionEvent.type=3) >> convert it to local time in HH:MM DD:MM:YY format. >>> Button OK will close the popup.
13. **As soon as appropriate event/s accepted – add/update new position's details on `Markets` and `My activity:orders` screens.**



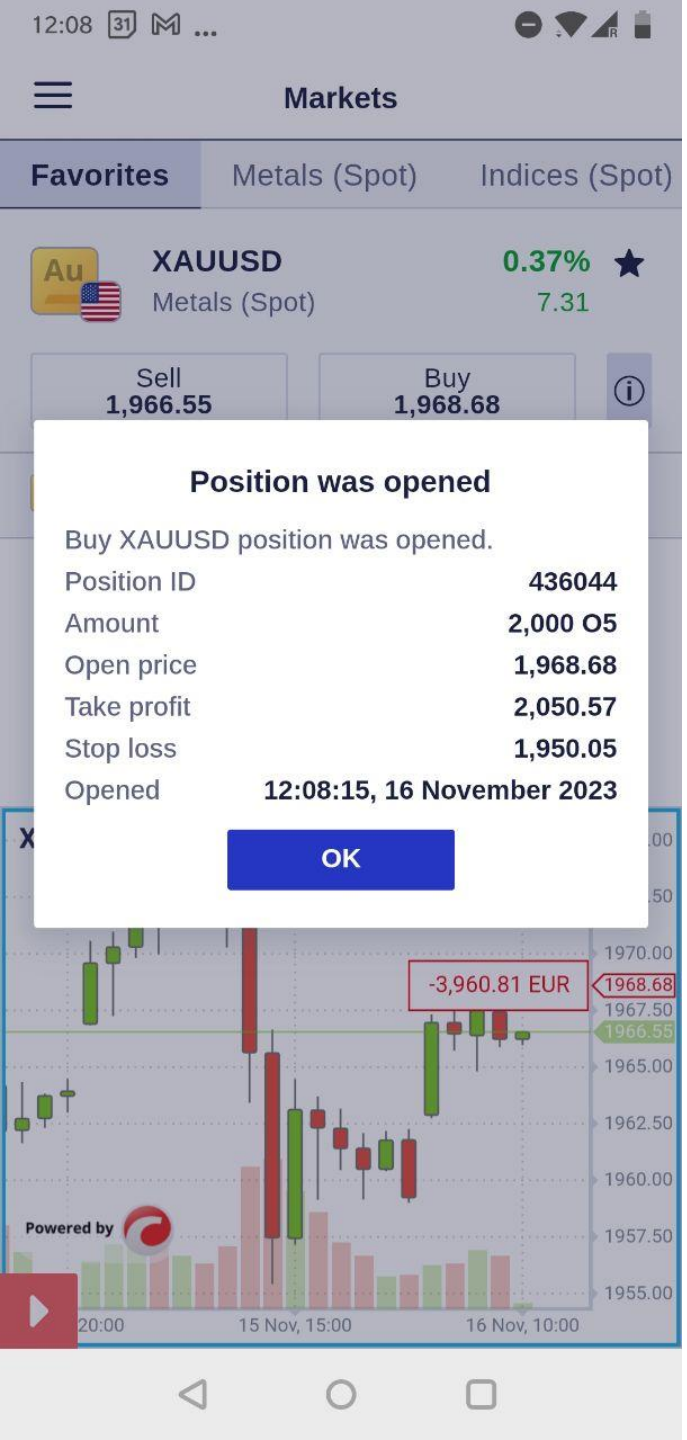
Execution popups: pending order executed W\O SLTP

1. ***The screen is a partial example of SLTP rejection.***
2. This popup template has same header, but different bodies; it includes "Buy EURUSD position was **opened/increased/decreased/reversed/closed**", according to conditions in .xls file.
3. Happens in case same case of "Pending order execution", but followed by `executionType` =7 (REJECTED) when `orderType`=4 (SLTP) ***instead of*** `executionType` =2 (ACCEPTED) when `orderType`=4 (SLTP) .
4. Direction (Buy/Sell): - `order.tradeData` >> `tradeSide`, `; if "reversed" – use **the opposite** side AND add the following (with orig. `tradeSide`): "Now your position is `tradeSide` EURUSD.
5. "Position ID:" - `position.positionId`
6. "Order ID:" - `order.orderId`
7. "Amount:" - `position.tradeData` >> `volume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
8. "Price:" - `position.price`
9. "Either Stop Loss or Take Profit was rejected. Please try to set it again."
10. "Created:" - `position.tradeData.openTimestamp` (of executionEvent.type=3) >> convert it to local time in HH:MM DD:MM:YY format.
11. Button OK will close the popup.
12. **As soon as appropriate event/s accepted – add/update new position's details on `Markets` and `My activity:orders` screens.**

Execution popups: pending order deleted

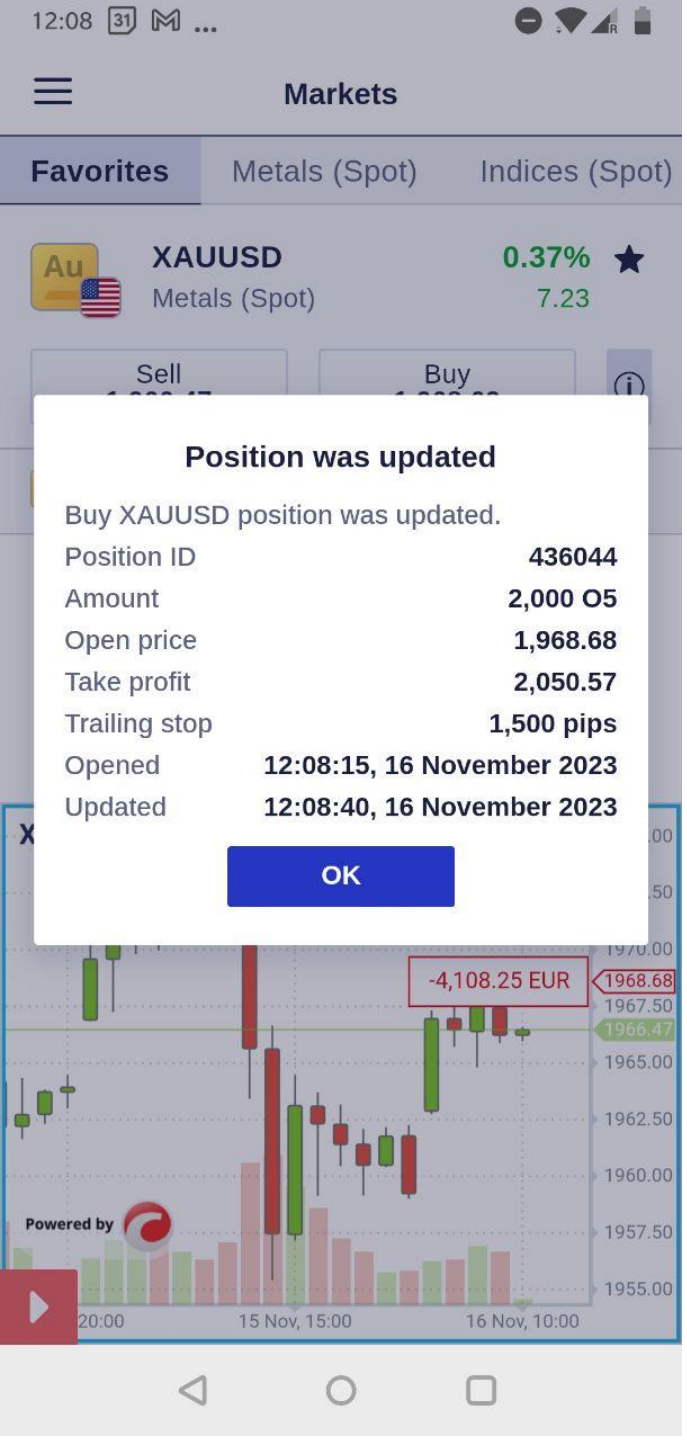


1. Direction (Buy/Sell): - `order.tradeData` >> `tradeSide`
2. "Order ID:" - `order.orderId`
3. "Amount:" - `order.tradeData` >> `volume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
4. "Price:" - either `order.limitPrice`/`order.stopPrice`, depends what is field is filled. Or expect it by details of appropriate request.
5. "Deleted:" - ` order.utcLastUpdateTimestamp ` >> convert it to local time in HH:MM DD:MM:YY format.
6. Button OK will close the popup.
7. **As soon as appropriate event/s accepted – delete given pending order from `Markets` and `My activity:orders` screens.**



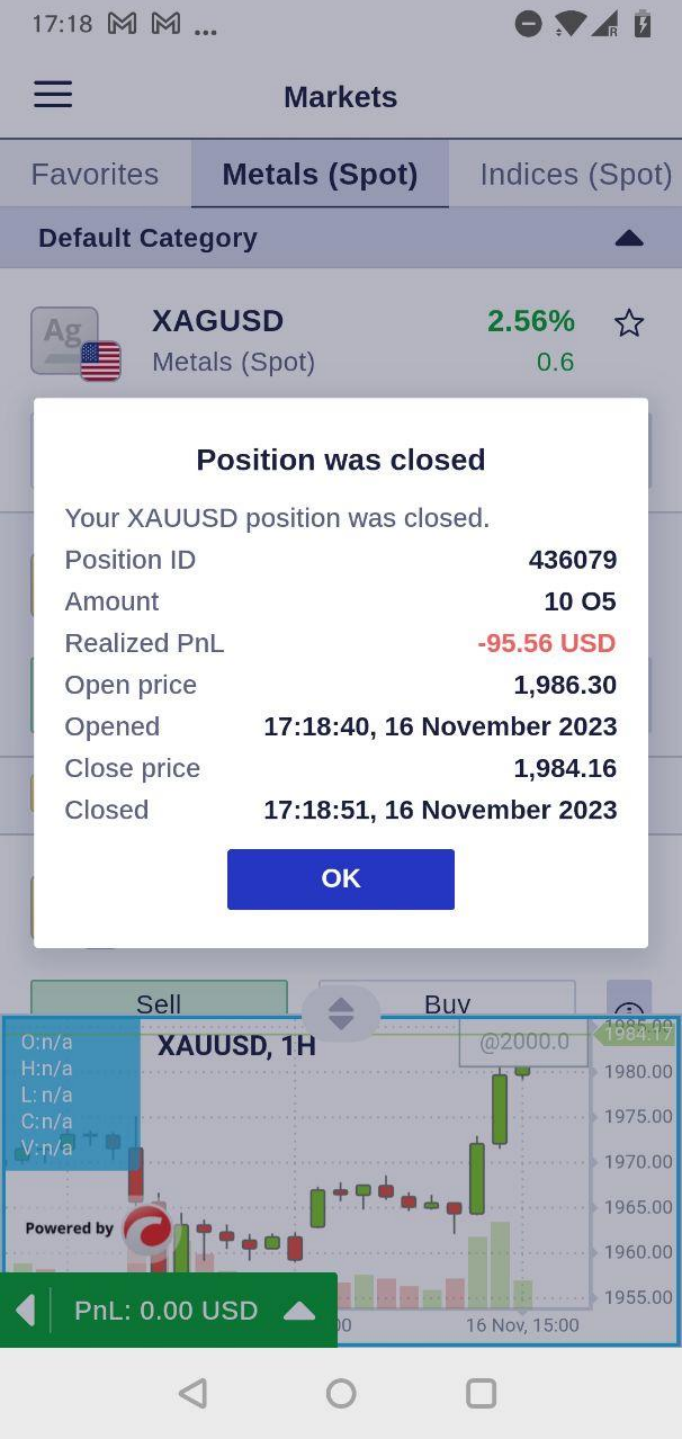
Execution popups: position opened

1. Direction (Buy/Sell): - `order.tradeData` >> `tradeSide`
2. "Position ID:" - `position.positionId`
3. "Amount:" - `position.tradeData` >> `volume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
4. "Opening price:" - `position.price`.
5. **For `NETTED` accounts only**: if related order in `executionEvent.type`=3 has relativeStopLoss/ TakeProfit`=empty (e.g. executionEvent.type=3 should NOT be verified by `accepted` or `replaced` SLTP event) get SL/TP/TS by from `position.takeProfit`/`stopLoss`/`trailingStop` fields of position in same event. Process the results by same rules of next three paragraphs.
6. "Take profit:" - `position.takeProfit` of **the following** `executionEvent` with type=2. If `takeProfit`=empty/NaN – show "No".
7. "Stop Loss:" - `position.stopLoss` of **the following** `executionEvent` with type=2. If `stopLoss`=empty/NaN – show "No".
8. "Trailing Stop:" - if `position.stopLoss` of **the following** ExecutionEvent=value and `position.tralinigStopLoss`=TRUE >> |`position.price` -`position.stopLoss`|/$10^{\text{Symbol.pipPosition}}$, add "pips". Place **instead** of "Stop Loss".
9. "Opened:" - `position.tradeData.openTimestamp` (of executionEvent.type=3) >> convert it to local time in HH:MM DD:MM:YY format.
10. Button OK will close the popup.
11. **As soon as appropriate event/s accepted – add new position's details on `Markets` and `My activity:orders` screens.**



Execution popups: position updated

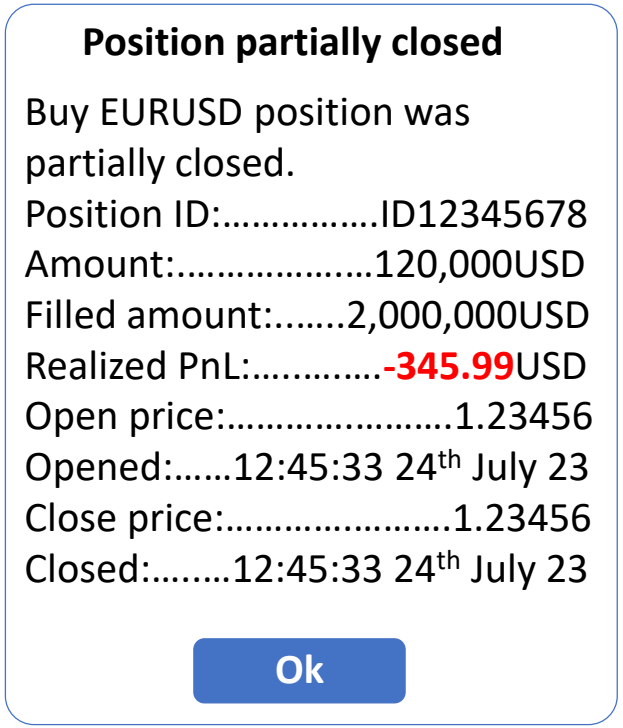
1. Direction (Buy/Sell): - `order.tradeData` >> `tradeSide`
2. "Position ID:" - `position.positionId`
3. "Amount:" - `position.tradeData` >> `volume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
4. "Opening price:" - `position.price`.
5. **For `NETTED` accounts only**: if related order in `executionEvent.type`=3 has relativeStopLoss/TakeProfit`=empty (e.g. executionEvent.type=3 should NOT be verified by `accepted` or `replaced` SLTP event) get SL/TP/TS by from `position.takeProfit`/`stopLoss` /`trailingStop` fields of position in same event. Process the results by rules of #6 - #8.
6. "Take profit:" - `position.takeProfit` of **the following** `executionEvent` with type=2. If `takeProfit`=empty/NaN – show "No".
7. "Stop Loss:" - `position.stopLoss` of **the following** `executionEvent` with type=2. If `stopLoss`=empty/NaN – show "No".
8. "Trailing Stop:" - if `position.stopLoss` of **the following** ExecutionEvent=value and `position.tralinigStopLoss`=TRUE >> |`position.price` -`position.stopLoss`|/$10^{\text{Symbol.pipPosition}}$, add "pips". Place **instead** of "Stop Loss".
9. "Opened:" - `position.tradeData.openTimestamp` (of executionEvent.type=3) >> convert it to local time in HH:MM DD:MM:YY format.
10. "Updated:" - `position.utcLastUpdateTimestamp` (of executionEvent.type=3) >> convert it to local time in HH:MM DD:MM:YY format.
11. Button OK will close the popup.
12. **As soon as appropriate event/s accepted – add new position's details on `Markets` and `My activity:orders` screens.**



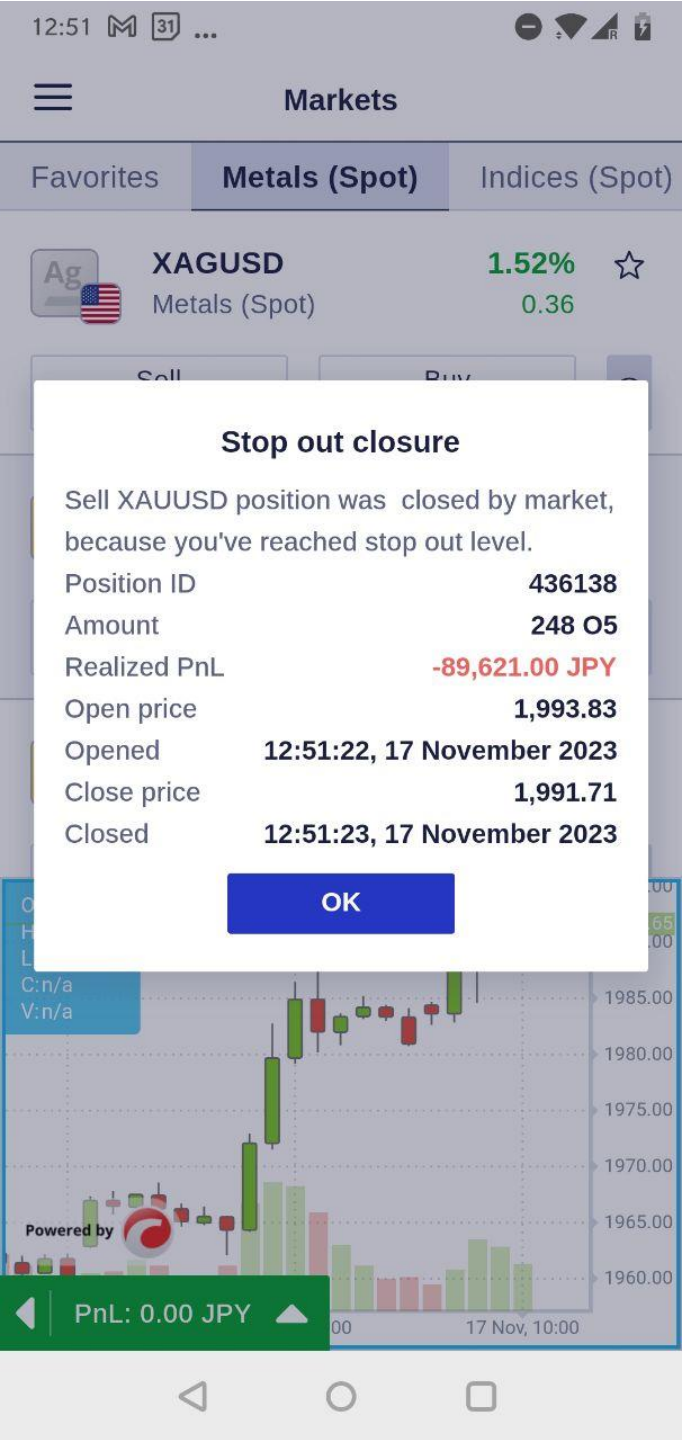
Execution popups: position closed

1. Direction (Buy/Sell): - `order.tradeData` >> `tradeSide`
2. "Position ID:" - `position.positionId`
3. "Amount:" - `deal.filledVolume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
4. "PnL:" - (`deal.closePositionDetail.grossProfit`+`swap`+`commission`)/100; with value of `depositAsset`; if=>0 - paint the PnL value in green, if <0 – in red.
5. "Open price:" - `deal.closePositionDetail.entryPrice`
6. "Opened:" - `position.tradeData.openTimestamp`
7. "Close price:" - `deal.executionPrice`.
8. "Closed:" - `deal.executionTimestamp ` >> convert it to local time in HH:MM DD:MM:YY format.
9. Button OK will close the popup.
10. **As soon as appropriate event/s accepted – delete given position from `Markets` and `My activity:orders` screens.**

Execution popups: position partially closed

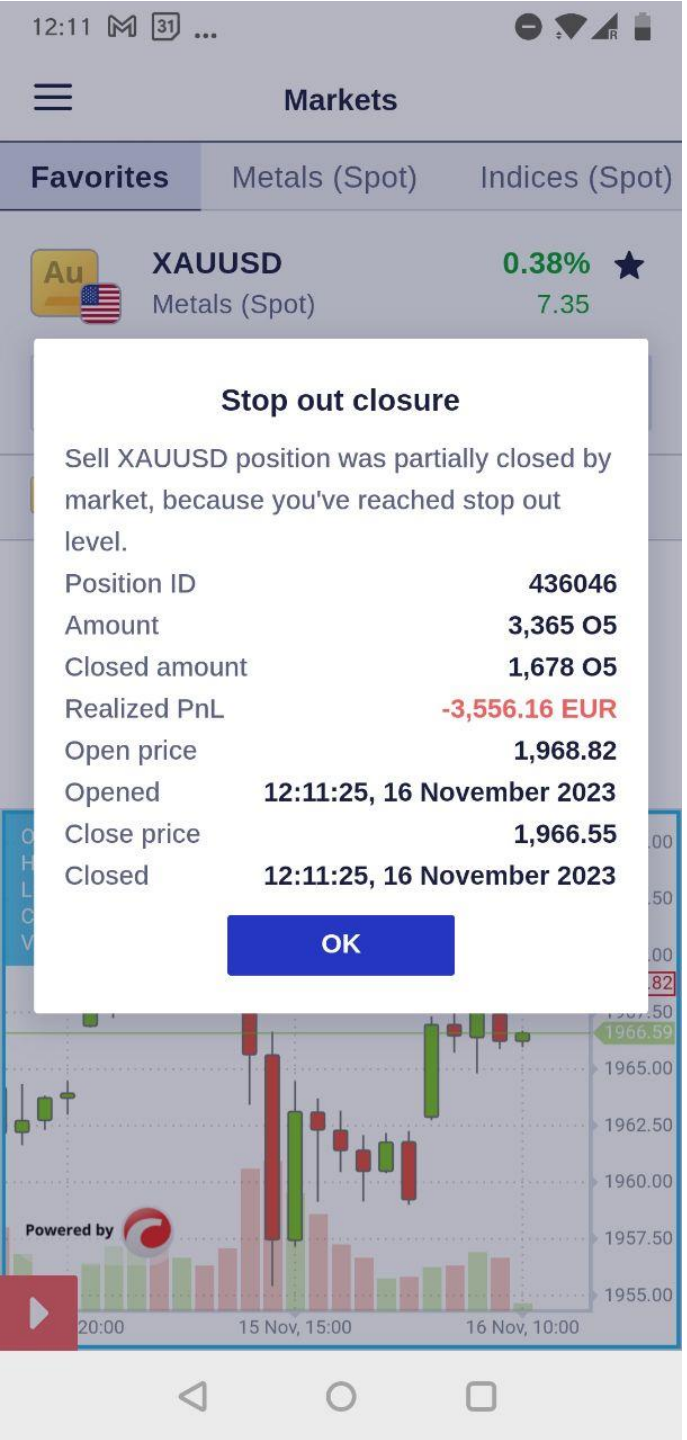


1. Direction (Buy/Sell): - `order.tradeData` >> `tradeSide`
2. "Position ID:" - `position.positionId`
3. "Amount:" - `position.tradeData` >> `volume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
4. "Filled amount:" - `deal.filledVolume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
5. "Realized PnL:" - (`deal.closePositionDetail.grossProfit`+`swap`+`commission`)/100; with value of `depositAsset`; if=>0 - paint the PnL value in green, if <0 – in red.
6. "Open price:" - `deal.closePositionDetail.entryPrice`
7. "Opened:" - `position.tradeData.openTimestamp`
8. "Close price:" - `deal.executionPrice`.
9. "Closed:" - `deal.executionTimestamp ` >> convert it to local time in HH:MM DD:MM:YY format.
10. Button OK will close the popup.
11. **As soon as appropriate event/s accepted – update given position's details on`Markets` and `My activity:orders` screens.**



Execution popups: position closed by market

1. Direction (Buy/Sell): - `order.tradeData` >> `tradeSide`
2. "Position ID:" - `position.positionId`
3. "Amount:" - `deal.filledVolume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
4. "Realized PnL:" - (`deal.closePositionDetail.grossProfit`+`swap`+`commission`)/100; with value of `depositAsset`; if=>0 - paint the PnL value in green, if <0 – in red.
5. "Open price:" - `deal.closePositionDetail.entryPrice`
6. "Opened:" - `position.tradeData.openTimestamp`
7. "Close price:" - `deal.executionPrice`.
8. "Closed:" - `deal.executionTimestamp ` >> convert it to local time in HH:MM DD:MM:YY format.
9. Button OK will close the popup.
10. **As soon as appropriate event/s accepted – update given position's details on`Markets` and `My activity:orders` screens.**

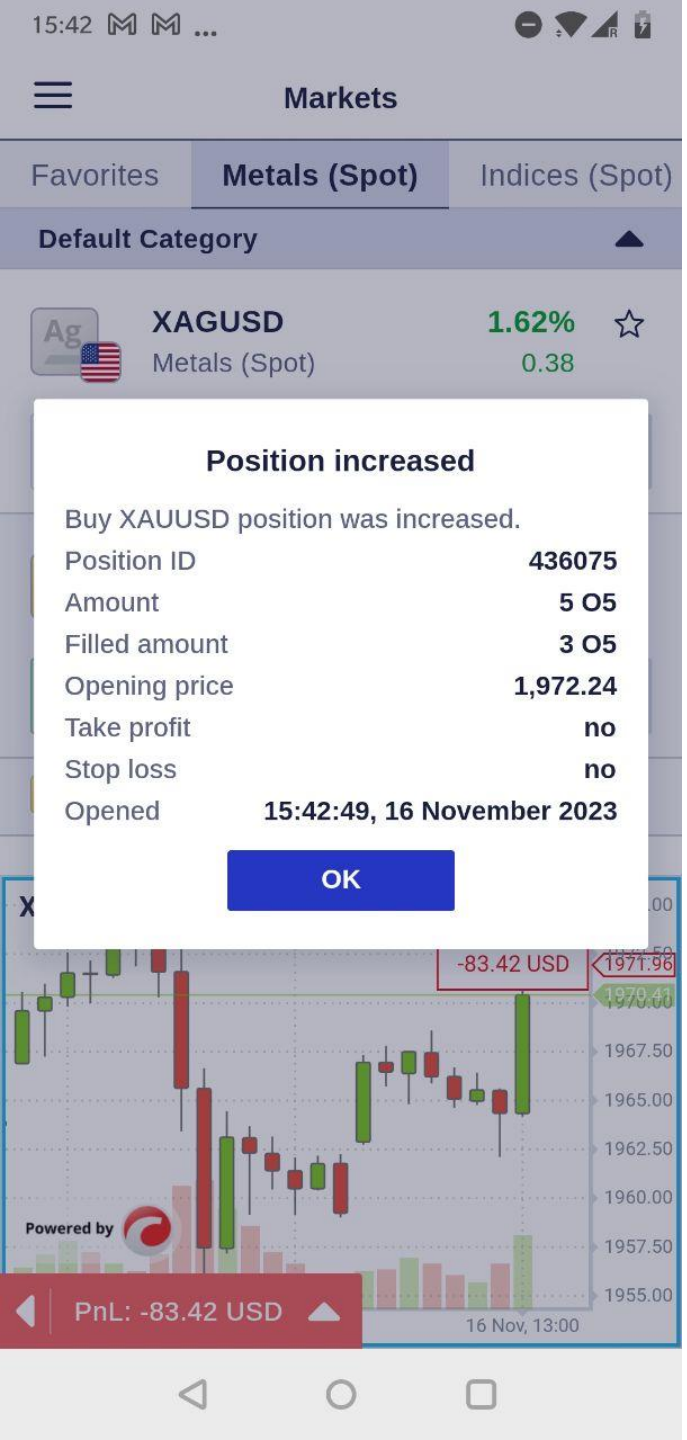


Execution popups: position partially closed by market

1. Direction (Buy/Sell): - `order.tradeData` >> `tradeSide`
2. "Position ID:" - `position.positionId`
3. "Amount:" - `position.tradeData.volume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
4. "Closed amount:" - `deal.filledVolume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
5. "Realized PnL:" - (`deal.closePositionDetail.grossProfit`+`swap`+`commission`)/100; with value of `depositAsset`; if=>0 - paint the PnL value in green, if <0 – in red.
6. "Open price:" - `deal.closePositionDetail.entryPrice`
7. "Opened:" - `position.tradeData.openTimestamp`
8. "Close price:" - `deal.executionPrice`.
9. "Closed:" - `deal.executionTimestamp ` >> convert it to local time in HH:MM DD:MM:YY format.
10. Button OK will close the popup.
11. **As soon as appropriate event/s accepted – update given position's details on`Markets` and `My activity:orders` screens.**

Execution popups: position temporary closed

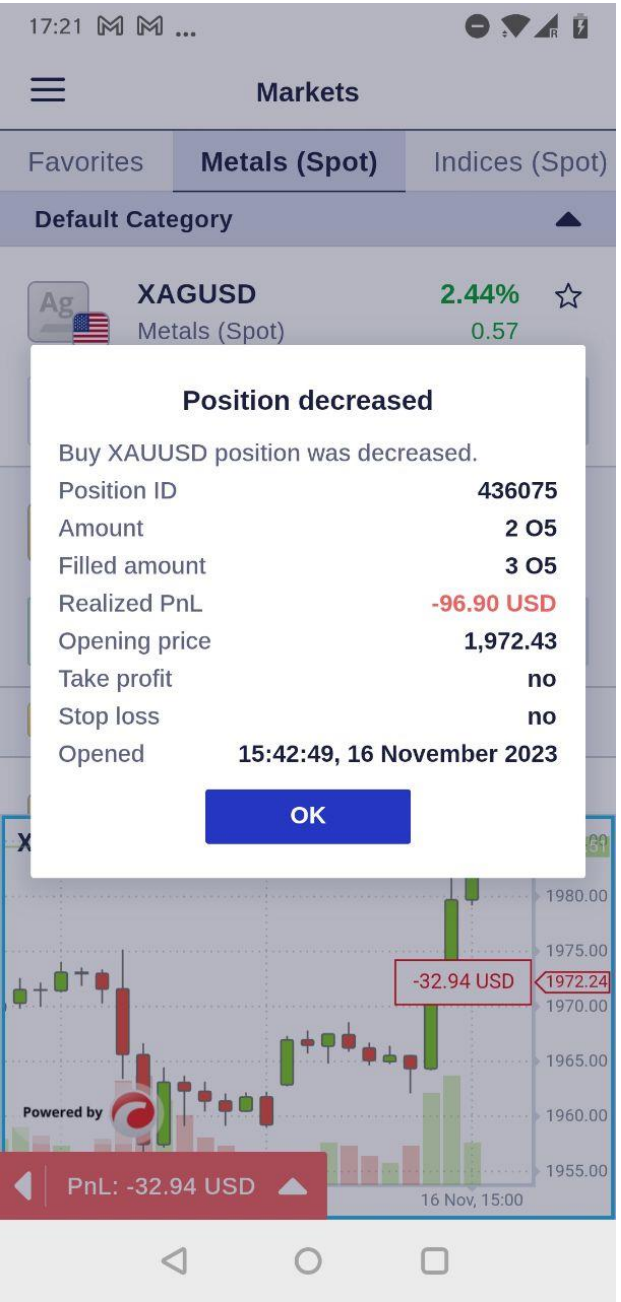
1. Direction (Buy/Sell): - `order.tradeData` >> `tradeSide`
2. "Position ID:" - `position.positionId`
3. "Closed amount:" - `deal.filledVolume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
4. "Pending amount:" - **(**`order.tradeData.volume` - `order.executedVolume`**)**/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
5. "Realized PnL:" - (`deal.closePositionDetail.grossProfit`+`swap`+`commission`)/100; with value of `depositAsset`; if=>0 - paint the PnL value in green, if <0 – in red.
6. "Open price:" - `deal.closePositionDetail.entryPrice`
7. "Opened:" - `position.tradeData.openTimestamp`
8. "Close price:" - `deal.executionPrice`.
9. "Closed:" - `deal.executionTimestamp ` >> convert it to local time in HH:MM DD:MM:YY format.
10. Button OK will close the popup.
11. **As soon as appropriate event/s accepted – delete given pending order from `Markets` and `My activity:orders` screens.**



Execution popups: position increased

1. Direction (Buy/Sell): - `order.tradeData` >> `tradeSide`
2. "Position ID:" - `position.positionId`
3. "Amount:" - `position.tradeData` >> `volume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
4. "Filled amount:"- `deal.tradeData.volume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
5. "Opening price:" - `position.price`.
6. **For `NETTED` accounts only**: if related order in `executionEvent.type`=3 has relativeStopLoss/ TakeProfit`=empty (e.g. executionEvent.type=3 should NOT be verified by `accepted`/`replaced` SLTP event) get SL/TP/TS by from `position.takeProfit`/`stopLoss` /`trailingStop` fields of position in same event. Process the results by rules of #7 - #9.
7. "Take profit:" - `position.takeProfit` of **the following** `executionEvent` with type=2. If `takeProfit`=empty/NaN – show "No".
8. "Stop Loss:" - `position.stopLoss` of **the following** `executionEvent` with type=2. If `stopLoss`=empty/NaN – show "No".
9. "Trailing Stop:" - if `position.stopLoss` of **the following** ExecutionEvent=value and `position.tralinigStopLoss`=TRUE >> |`position.price` -`position.stopLoss`|/$10^{\text{Symbol.pipPosition}}$, add "pips". Place **instead** of "Stop Loss".
10. "Opened:" - `position.tradeData.openTimestamp` (of executionEvent.type=3) >> convert it to local time in HH:MM DD:MM:YY format.
11. Button OK will close the popup.
12. **As soon as appropriate event/s accepted – update position's details on `Markets` and `My activity:orders` screens.**

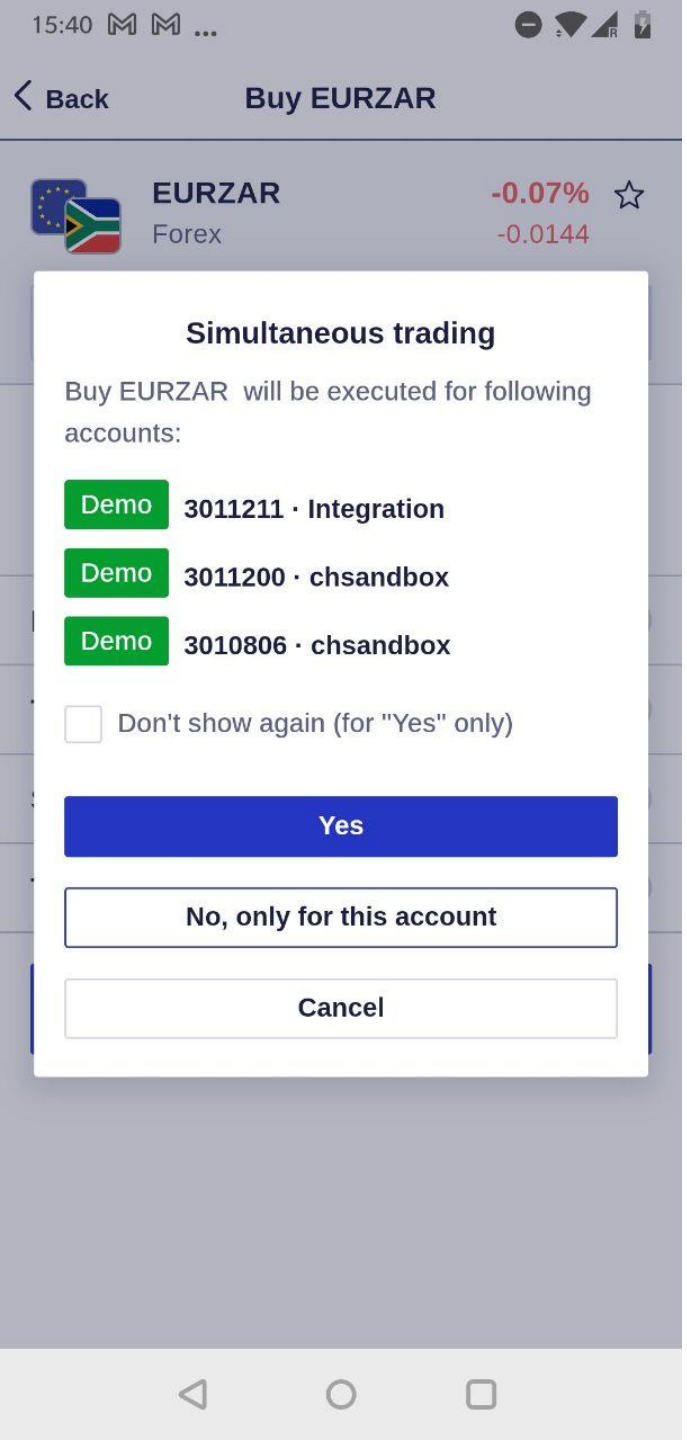
Execution popups: Position decreased



1. Direction (Buy/Sell): - `order.tradeData` >> `tradeSide`
2. "Position ID:" - `position.positionId`
3. "Amount:" - `position.tradeData` >> `volume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
4. "Filled amount:"- `deal.tradeData.volume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
5. "Realized PnL:" - (`deal.closePositionDetail.grossProfit`+`swap`+`commission`)/100; with value of `depositAsset`; if=>0 - paint the PnL value in green, if <0 – in red.
6. "Opening price:" - `position.price`.
7. **For `NETTED` accounts only**: if related order in `executionEvent.type`=3 has relativeStopLoss/ TakeProfit`=empty (e.g. executionEvent.type=3 should NOT be verified by `accepted` or `replaced` SLTP event) get SL/TP/TS by from `position.takeProfit`/`stopLoss`/`trailingStop` fields of position in same event. Process the results by same rules of next three paragraphs.
8. "Take profit:" - `position.takeProfit` of **the following** `executionEvent` with type=2. If `takeProfit`=empty/NaN – show "No".
9. "Stop Loss:" - `position.stopLoss` of **the following** `executionEvent` with type=2. If `stopLoss`=empty/NaN – show "No".
10. "Trailing Stop:" - if `position.stopLoss` of **the following** ExecutionEvent=value and `position.tralinigStopLoss`=TRUE >> |`position.price` -`position.stopLoss`|/$10^{\text{Symbol.pipPosition}}$, add "pips". Place **instead** of "Stop Loss".
11. "Opened:" - `position.tradeData.openTimestamp` (of executionEvent.type=3) >> convert it to local time in HH:MM DD:MM:YY format.
12. Button OK will close the popup.
13. **When appropriate event/s accepted – update details in `Markets`/`My activity:orders`.**

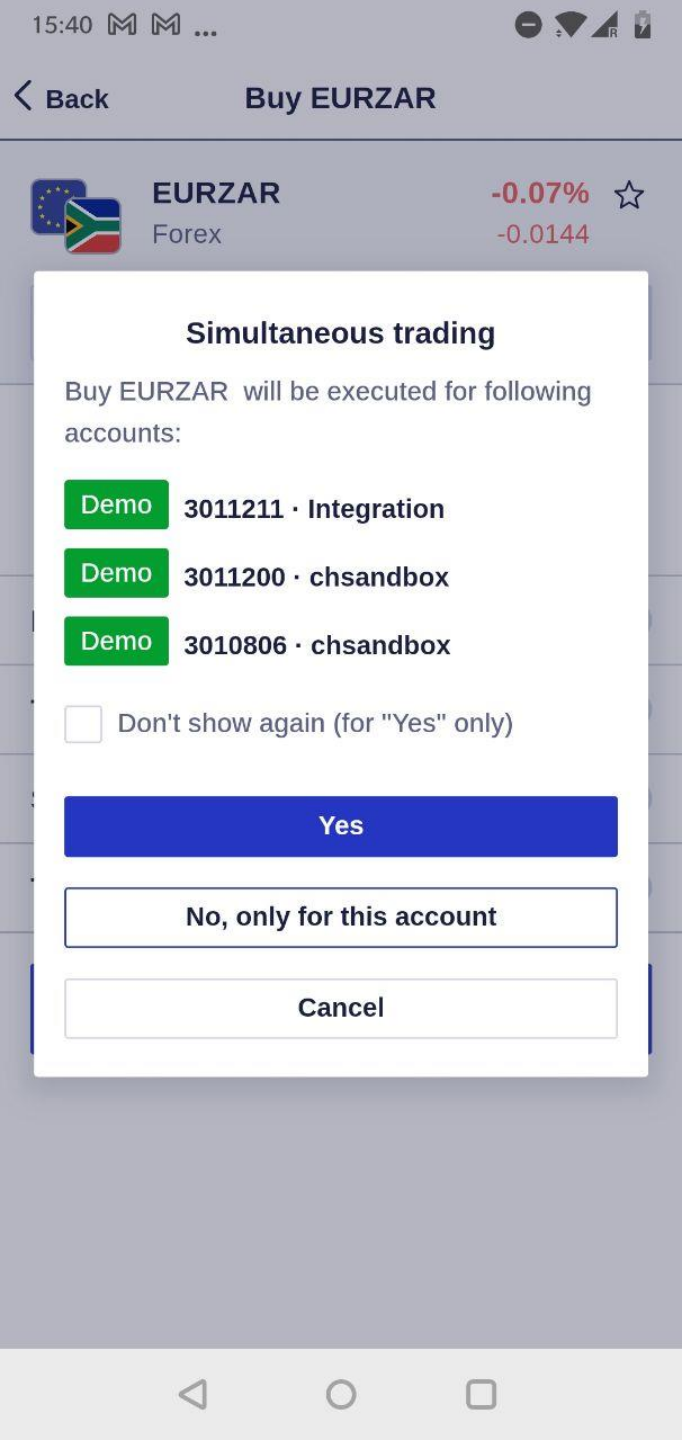
Execution popups: Position rejected (or cancelled)

1. Basically, this popup will be show if market order wasn't accepted by market. Most common scenario is that market order will be rejected. Still, expect `executionType`=5 for `orderType`=1.
   So when `executionType`=7 (REJECTED): popup's header is "Position rejected", while in case of `executionType`=5 header will be "Position cancelled". In same way, body will be "Buy EURUSD position was rejected by the market" for `executionType`=7 and "Buy EURUSD position was cancelled by the market" for `executionType`=5.
2. Direction (Buy/Sell): - `order.tradeData` >> `tradeSide`
3. "Order ID:" - `order.orderId`
4. "Amount:" - `order.tradeData` >> `volume`/100. Add `measurementUnits`. If =empty, request `baseAssetId` from `LightSymbol`, get its `assetName` (and add it).
5. "Rejected:" (or "Canceled") - either `createTimestamp` or `order.utcLastUpdateTimestamp` >> convert it to local time in HH:MM DD:MM:YY format.
6. Button OK will close the popup.

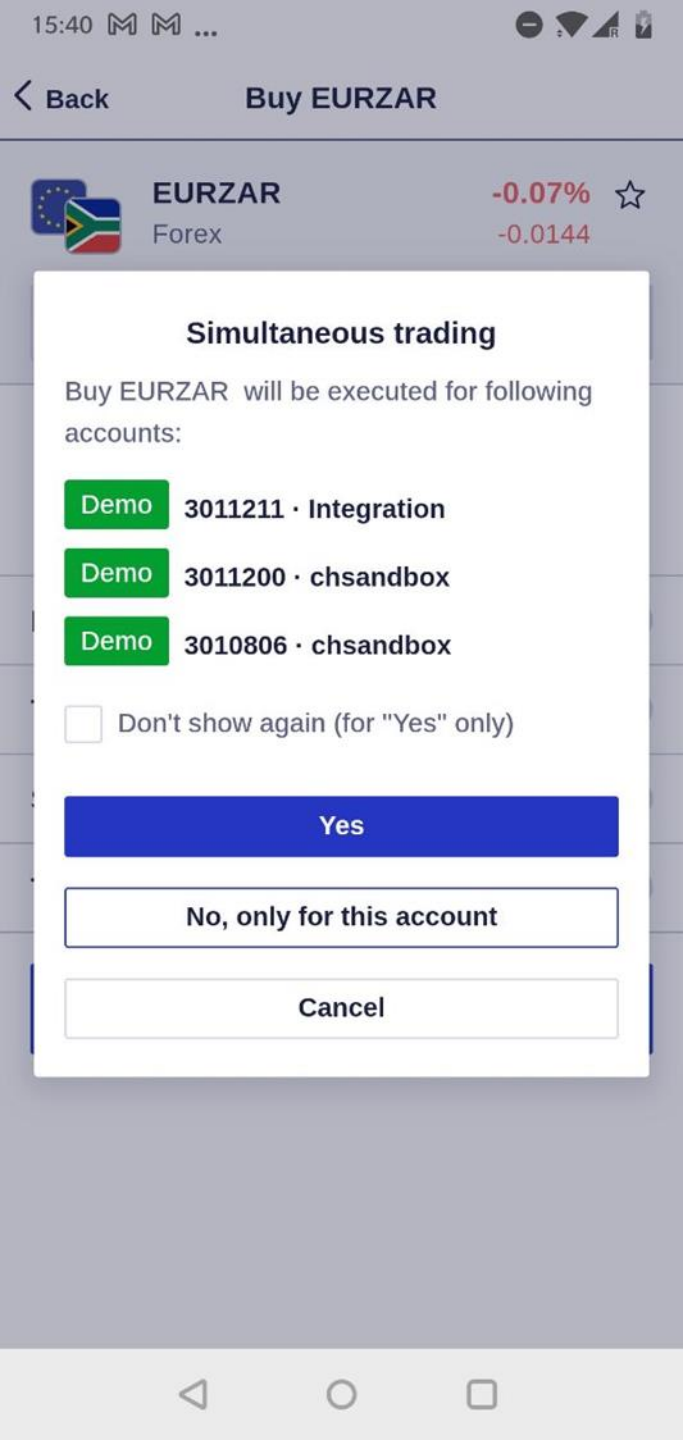


Simultaneous trading

1. If "simultaneous trading" option was successfully activated (see p. 10), send `SymbolsListReq` for other accounts (account in `accounts choice menu` already has available symbols list). Create a matching scheme of all symbols for accounts for simul. trading by following rules:
   a) Look for symbol with exact match, e.g. if an order is sent for "EURUSD.ABC" – look for that symbol in list related to account/s that are marked for simultaneous trading.
   b) If nothing was found – delete all non-letter and number (including spaces) characters from `symbolName` and try to match by first 6 letters of original symbol. Then, symbols like EURUSD, EUR/USD1 and EURUSD.AbCd will be matched.
   c) Also, special matching table can be created manually, with any possible customization, for example: **Germany 40 = DAX = DE30**.
   d) If nothing found, show a popup – header: "Incomplete matching", body: "Some symbols couldn't be matched for all accounts. In case of simultaneous trading some orders won't be executed in all your accounts". Below show "[symbolName] wasn't found on the following accounts:" BrokerNameShort..... Login, BrokerNameShort.....Login etc.
2. After approval of more than 1 account for simultaneous trading (radio button in "My account"), when "BUY/SELL" button with valid order details will be tapped – following popup will appear before sending the order to the market – header: "Simultaneous trading", body: "Buy EURUSD position will be executed for the following accounts:", while below there will be a list of approved accounts.



3. Below there will also a checkbox with "don’t show again (for "Yes" only)". If checked and then "Yes" will be tapped – execute current order after the tap and execute next orders w/o warning for all approved accounts. If not checked – show this warning popup again.
4. If #1 checkbox is tapped, but the choice is "No, for this account only" – ignore tapped checkbox and during the next order sent for simultaneous trading show again this popup.
5. (Probably related to p. 11 description of activation process of simultaneous trading) Another important verification must be performed: when choice of accounts for which simultaneous trading will be activated was successfully saved – check whether `Trader.accountType` is the same for all chosen accounts (either all are 0 or 1; as it was mentioned earlier - `accountType=2` won't be allowed to access this app).
6. If both `accountType`=0 and 1 can be found among accounts chosen for simultaneous trading, please show the following popup – H: "Warning", B: "You included both hedging and netting account types. Having more than one order for a symbol at any given moment might lead to different outcomes.
7. When execution of given order for more than one account starts (after user tapped "OK" on popup mentioned in #2 of previous page) – send consequently appropriate `NewOrderReq` for all approved accounts, when the first account **after currently chosen in `account choice menu`** will be the account with the highest 'cTidTraderAccountId'.



8. After order is sent to execution – show user the appropriate popups that appear after execution events or `OrderErrorEvent` (p. 54-74), but with addition of 2 fields:
   a) "Broker:"… `ProtoOACtidTraderAccount.brokerTitleShort` **(expect this filed; will be added soon; in a meanwhile add: "Broker Name").**
   b) "Account:"…`traderLogin`
9. Place popups one below another, so responding popup to 1st request will be on top until "OK" is tapped, then there will be a popup of 2nd request, etc.
10. When given order is successfully executed for more than one account – get from `execution events` appropriate `order/position id`, **link these ids together** and keep it in app's cache. Please note, that its necessary to keep `brokerTitleShort` as well, as `orderId`/`positionId` are unique only per broker (add "Broker Name" for all for now).
11. Next time when user will make any action with given order/position – before sending execution command – the next popup will appear: header: "Simultaneous trading", body: "A/an order/s / position/s with the same details was previously opened for other accounts: list: [traderLogin: brokerTitleShort] Would you like to apply the changes on other accounts?"
12. Below there will be a checkbox and "Don't show this message again" text and 3 buttons: Yes (executes for linked positions), No, only for this account (executes only for this account) and Cancel (closes popup w\o taking any action). Send appropriate request/s according to user's choice to activated for simultaneous trading accounts. Show approp. popups by rules in #2.

Symbol Icons

1. A pack of symbols' flags was created, including icons for most forex pairs and major commodities, metals, crypto and indices.
2. The pack is located on the following link and free for commercial usage >> https://www.figma.com/file/FtFCM92zSJbH8a2DKV9WVL/Open-API-cTrader?type=design&node-id=29-2&mode=design
3. Considering #3 – forex symbols' flags will be the easiest to adjust to appropriate symbols automatically, as name (for example) EURUSD can be easily added to any possible symbol variation name, like "EUR/USD", "EURUSD1" or "EUR/USD.".
4. In cTrader Open API example app's code there will be easy accessible option to add different names (like mentioned in #3, but `assetClass` like indices can have names that will be difficult for automatic adjustment) for same symbol flag icon, so any broker will get as much as possible flags on its traded symbols.
5. For worst case scenario there is a "N/A" icon, that will automatically match any symbol without appropriate flag.

Demo/Live account

1. Current version of Open API Trader application has only `demo` accounts allowed.
2. In case you'd like to allow trading for 'live` accounts as well, please follow next instructions:
3. Go to `lib/main.dart` file and update ONLY_DEMO constant to TRUE for disabling and to FALSE for enabling `live` accounts.

Translations

1. cTrader Open API Example app currently released in English only.
2. All English texts are placed in lib/dictionaries/dict_en.arb.
3. In order to add translation to another language, a new file should be created (for example, for Spanish >> lib/dictionaries/dict_es.arb) and activate `flutter gen-l10n`.