**Referrals**

Get fee discounts and earn rebates through the GMX referral program.

**How it works**

To create a referral code:

* Go to <https://app.gmx.io/#/referrals>​
* Click on the Affiliates tab
* Create a referral code using any combination of letters, numbers and underscores

Note that referral codes are case sensitive and that your code must be created on both Arbitrum as well as Avalanche to earn rebates on both networks.

Once you've created your code, click on the copy icon next to the code to get your referral link, it should look something like this: [https://gmx.io/#/?ref=<your code>](https://gmx.io/#/?ref=xdev_10).

You can share this link on any platform, e.g. Twitter, Telegram. When a user clicks on your link, your referral code would be stored with the user's account. When the user makes a trade they would receive a discount and you would earn rebates from their trading fees. You will continue to earn rebates even if the user uses another device later on as the referral code is stored on the contract the first time the user makes a trade.

The discounts and rebates will be distributed as ETH on Arbitrum and AVAX on Avalanche every Wednesday, your rebates history will be viewable on <https://app.gmx.io/#/referrals>.

Note that the referral program is subject to change as it is determined by the token holders of GMX. Full referral terms at <https://gmx.io/#/referral-terms>.

**Tiers**

The referral program has a tier system to prevent gaming through self-referrals, this helps to ensure that referrers receive the rebates for the users they brought onto the platform.

* Tier 1: 5% discount for traders, 5% rebates to referrer
* Tier 2: 10% discount for traders, 10% rebates to referrer
* Tier 3: 10% discount for traders, 15% rebates to referrer paid in ETH / AVAX, 5% rebates to referrer paid in esGMX

Anyone can create a Tier 1 code. To upgrade your code to Tier 2 or Tier 3:

* Tier 2: At least 15 active users using your referral codes per week and a combined weekly volume above $5 million
* Tier 3: At least 30 active users using your referral codes per week and a combined weekly volume above $25 million

If your account fulfils this criteria, please send a dm to [@GMXPartners](https://t.me/GMXPartners).

Rebates and discounts apply on the opening and closing fees for leverage trading.

The opening and closing fees are 0.1% on GMX, there is no price impact for trades and zero spread for tokens like BTC and ETH, rebates are calculated before user discounts so referrers earn on the full maker fee and from what would otherwise be spread on other exchanges. As a result, referrers would earn equivalent amounts of rebates per volume on GMX when compared to other referral programs.

Note that there is a cap of 5000 esGMX distributed per week. If the price of GMX is $30 the full 5% bonus can be paid for total Tier 3 referral volumes up to $3 billion per week. esGMX tokens distributed for this program will not require GMX or GLP to [vest](https://gmxio.gitbook.io/gmx/rewards), the vault to vest the tokens will be available towards the end of Q1 2023.

The price of esGMX will be based on the 7 day TWAP of GMX.

Wallet providers and other protocols will be eligible for Tier 2 and Tier 3 rewards as well.

**API**

For contract interaction: <https://gmxio.gitbook.io/gmx/contracts>.

The URLs below are for Arbitrum, for Avalanche the same URLs are available, with the base URL being [https://gmx-avax-server.uc.r.appspot.com](https://gmx-avax-server.uc.r.appspot.com/).

**Tokens**

* URL: <https://api.gmx.io/tokens>​
* Retrieve a list of tokens and info for each token
* USD amounts are multiplied by (10 \*\* 30)
* Token amounts are multiplied by (10 \*\* token.decimals)

**Prices**

The tokens endpoint will return the current prices in the contract, these prices are used for swaps, GLP minting / redeeming. For opening and closing positions, the [realtime median prices](https://gmx-io.notion.site/GMX-Technical-Overview-47fc5ed832e243afb9e97e8a4a036353) are used, to get these values:

* URL: <https://api.gmx.io/prices>​
* Retrieve a list of token prices
* Prices are multiplied by (10 \*\* 30)

**Actions**

* URL: <https://api.gmx.io/actions>​
* Retrieve a list of actions, these will include swaps, increasing a position, decreasing a position, orders and liquidations
* Params:
  + account (optional): pass in the checksum address of an account
  + after (optional): for pagination, the id to start after
* To get the checksum address: <https://web3-tools.netlify.app/>, you can also use a library like [ethers](https://docs.ethers.io/v5/)​

**Volume**

* Hourly: <https://api.gmx.io/hourly_volume>​
* Daily: <https://api.gmx.io/daily_volume>​
* Weekly: <https://api.gmx.io/weekly_volume>​
* Total: <https://api.gmx.io/total_volume>​
* Volumes are separated by action, token, day / week
* Params:
  + after (optional): for pagination, the id to start after

**Contracts**

List of GMX contracts.

**Overview**

For a technical overview of GMX: <https://gmx-io.notion.site/GMX-Technical-Overview-47fc5ed832e243afb9e97e8a4a036353>. These docs are meant as an overview, please check the actual contract code for the exact implementation and for any possible edge cases if building an application, integration or similar using the contracts.

**Arbitrum**

* GMX: [0xfc5A1A6EB076a2C7aD06eD22C90d7E710E35ad0a](https://arbiscan.io/address/0xfc5A1A6EB076a2C7aD06eD22C90d7E710E35ad0a)​
* Vault: [0x489ee077994B6658eAfA855C308275EAd8097C4A](https://arbiscan.io/address/0x489ee077994B6658eAfA855C308275EAd8097C4A)​
* Router: [0xaBBc5F99639c9B6bCb58544ddf04EFA6802F4064](https://arbiscan.io/address/0xaBBc5F99639c9B6bCb58544ddf04EFA6802F4064)​
* PositionRouter: [0xb87a436B93fFE9D75c5cFA7bAcFff96430b09868](https://arbiscan.io/address/0xb87a436B93fFE9D75c5cFA7bAcFff96430b09868)​
* OrderBook: [0x09f77e8a13de9a35a7231028187e9fd5db8a2acb](https://arbiscan.io/address/0x09f77e8a13de9a35a7231028187e9fd5db8a2acb)​
* Reader: [0x22199a49A999c351eF7927602CFB187ec3cae489](https://arbiscan.io/address/0x22199a49A999c351eF7927602CFB187ec3cae489)​
* RewardReader: [0x8BFb8e82Ee4569aee78D03235ff465Bd436D40E0](https://arbiscan.io/address/0x8BFb8e82Ee4569aee78D03235ff465Bd436D40E0)​
* OrderBookReader: [0xa27C20A7CF0e1C68C0460706bB674f98F362Bc21](https://arbiscan.io/address/0xa27C20A7CF0e1C68C0460706bB674f98F362Bc21)​
* StakedGmx: [0xd2D1162512F927a7e282Ef43a362659E4F2a728F](https://arbiscan.io/address/0xd2D1162512F927a7e282Ef43a362659E4F2a728F)​
* StakedGlp: [0x5402B5F40310bDED796c7D0F3FF6683f5C0cFfdf](https://arbiscan.io/address/0x5402B5F40310bDED796c7D0F3FF6683f5C0cFfdf)​
* GlpManager: [0x3963FfC9dff443c2A94f21b129D429891E32ec18](https://arbiscan.io/address/0x3963FfC9dff443c2A94f21b129D429891E32ec18)​
* RewardRouter: [0xA906F338CB21815cBc4Bc87ace9e68c87eF8d8F1](https://arbiscan.io/address/0xA906F338CB21815cBc4Bc87ace9e68c87eF8d8F1)​
* GlpRewardRouter: [0xB95DB5B167D75e6d04227CfFFA61069348d271F5](https://arbiscan.io/address/0xB95DB5B167D75e6d04227CfFFA61069348d271F5)​
* ReferralStorage: [0xe6fab3F0c7199b0d34d7FbE83394fc0e0D06e99d](https://arbiscan.io/address/0xe6fab3f0c7199b0d34d7fbe83394fc0e0d06e99d)​
* GMX-ETH Uniswap Pool: [0x80A9ae39310abf666A87C743d6ebBD0E8C42158E](https://arbiscan.io/address/0x80A9ae39310abf666A87C743d6ebBD0E8C42158E)​

**Avalanche**

* GMX: [0x62edc0692BD897D2295872a9FFCac5425011c661](https://snowtrace.io/address/0x62edc0692BD897D2295872a9FFCac5425011c661)​
* Vault: [0x9ab2De34A33fB459b538c43f251eB825645e8595](https://snowtrace.io/address/0x9ab2De34A33fB459b538c43f251eB825645e8595)​
* Router: [0x5F719c2F1095F7B9fc68a68e35B51194f4b6abe8](https://snowtrace.io/address/0x5F719c2F1095F7B9fc68a68e35B51194f4b6abe8)​
* PositionRouter: [0xffF6D276Bc37c61A23f06410Dce4A400f66420f8](https://snowtrace.io/address/0xffF6D276Bc37c61A23f06410Dce4A400f66420f8)​
* OrderBook: [0x4296e307f108B2f583FF2F7B7270ee7831574Ae5](https://snowtrace.io/address/0x4296e307f108B2f583FF2F7B7270ee7831574Ae5)​
* Reader: [0x67b789D48c926006F5132BFCe4e976F0A7A63d5D](https://snowtrace.io/address/0x67b789D48c926006F5132BFCe4e976F0A7A63d5D)​
* RewardReader: [0x04Fc11Bd28763872d143637a7c768bD96E44c1b6](https://snowtrace.io/address/0x04Fc11Bd28763872d143637a7c768bD96E44c1b6)​
* OrderBookReader: [0xccFE3E576f8145403d3ce8f3c2f6519Dae40683B](https://snowtrace.io/address/0xccFE3E576f8145403d3ce8f3c2f6519Dae40683B)​
* StakedGmx: [0x4d268a7d4C16ceB5a606c173Bd974984343fea13](https://snowtrace.io/address/0x4d268a7d4C16ceB5a606c173Bd974984343fea13)​
* StakedGlp: [0xaE64d55a6f09E4263421737397D1fdFA71896a69](https://snowtrace.io/address/0xaE64d55a6f09E4263421737397D1fdFA71896a69)​
* GlpManager: [0xD152c7F25db7F4B95b7658323c5F33d176818EE4](https://snowtrace.io/address/0xD152c7F25db7F4B95b7658323c5F33d176818EE4)​
* RewardRouter: [0x82147C5A7E850eA4E28155DF107F2590fD4ba327](https://snowtrace.io/address/0x82147C5A7E850eA4E28155DF107F2590fD4ba327)​
* GlpRewardRouter: [0xB70B91CE0771d3f4c81D87660f71Da31d48eB3B3](https://snowtrace.io/address/0xB70B91CE0771d3f4c81D87660f71Da31d48eB3B3)​
* ReferralStorage: [0x827ED045002eCdAbEb6e2b0d1604cf5fC3d322F8](https://snowtrace.io/address/0x827ed045002ecdabeb6e2b0d1604cf5fc3d322f8)​
* GMX-AVAX Trader Joe Pool: [0x0c91a070f862666bBcce281346BE45766d874D98](https://snowtrace.io/address/0x0c91a070f862666bbcce281346be45766d874d98)​

**Bug Bounty**

There is an active bug bounty for the contracts, for more details: <https://gmxio.gitbook.io/gmx/bug-bounty>​

**Swap**

To execute a swap:

* Approve the Router contract for the token and amount you would like to swap
* Call Router.swap with parameters:
  + \_path: [tokenIn, tokenOut]
  + \_amountIn: amount of tokenIn to swap
  + \_minOut: minimum expected output amount
  + \_receiver: address of the receiver of tokenOut
* The function will revert if the amount of tokenOut sent to the receiver is less than *\_*minOut

To get swap amounts before execution:

* Call Reader.getMaxAmountIn with parameters:
  + \_vault: address of the vault
  + \_tokenIn: address of token that will be given
  + \_tokenOut: address of token to be received
  + The max amount of tokenIn that can be swapped will be returned
* Call Reader.getAmountOut with parameters:
  + \_vault: address of the vault
  + \_tokenIn: address of token that will be given
  + \_tokenOut: address of token to be received
  + \_amountIn: amount of tokenIn to swap
  + Two values will be returned, the first is the amount out after fees, and the second is the fee amount
  + The fee amount will be in terms of tokenOut

Tokens have a usdgAmount in the Vault contract used for some calculations, this amount is updated on minting of GLP, redemption of GLP and swaps based on the price of the token at the time. Due to price fluctuations this value may drift slightly from the actual USD value of the tokens in the pool, the usdgAmount is periodically updated to re-align values.

**Query Available Amounts**

The maximum sum of all position sizes is limited by the amount of tokens there are in the pool and any additional caps.

To calculate the available amount of liquidity for long positions:

* indexToken: the address of the token to long
* Available amount in tokens: Vault.poolAmounts(indexToken) - Vault.reservedAmounts(indexToken)
* Available amount in USD: PositionRouter.maxGlobalLongSizes(indexToken) - Vault.guaranteedUsd(indexToken)
* The available liquidity will be the lower of these two values
* PositionRouter.maxGlobalLongSizes(indexToken) can be zero, in which case there is no additional cap, and available liquidity is based only on the available amount of tokens

To calculate the available amount of liquidity for short positions:

* indexToken: the address of the token to short
* collateralToken: the address of the stablecoin token to be used as collateral
* Available amount in tokens: Vault.poolAmounts(collateralToken) - Vault.reservedAmounts(collateralToken)
* Available amount in USD: PositionRouter.maxGlobalShortSizes(indexToken) - Vault.globalShortSizes(indexToken)
* The available liquidity will be the lower of these two values
* PositionRouter.maxGlobalShortSizes(indexToken) can be zero, in which case there is no additional cap, and available liquidity is based only on the available amount of tokens

**Opening / Increasing a Position**

To open or increase the size of an existing position:

* Approve the PositionRouter as a Router plugin for your account
  + Router.approvePlugin(PositionRouter address)
* Approve the Router contract for the token and amount you would deposit as collateral for the position
* Call PositionRouter.createIncreasePosition with parameters:
  + \_path: [collateralToken] or [tokenIn, collateralToken] if a swap is needed
  + \_indexToken: the address of the token you want to long or short
  + \_amountIn: the amount of tokenIn you want to deposit as collateral
  + \_minOut: the min amount of collateralToken to swap for
  + \_sizeDelta: the USD value of the change in position size
  + \_isLong: whether to long or short
  + \_acceptablePrice: the USD value of the max (for longs) or min (for shorts) index price acceptable when executing the request
  + \_executionFee: can be set to PositionRouter.minExecutionFee
  + \_referralCode: [referral](https://gmxio.gitbook.io/gmx/referrals) code for affiliate rewards and rebates
  + \_callbackTarget: an optional callback contract, this contract will be called on request execution or cancellation
* After this transaction is sent a keeper will execute the request, the request will either be executed or cancelled
* If the position cannot be increased for reasons such as the \_acceptablePrice not being fulfillable or there being insufficient liquidity then the request will be cancelled and funds will be sent back to the msg.sender that called PositionRouter.createIncreasePosition
* \_minOut can be zero if no swap is required
* USD values for \_sizeDelta and \_price are multiplied by (10 \*\* 30), so for example to open a long position of size 1000 USD, the value 1000 \* (10 \*\* 30) should be used

**Closing / Decreasing a Position**

To close or decrease an existing position:

* Call PositionRouter.createDecreasePosition with parameters:
  + \_path: [collateralToken] or [collateralToken, tokenOut] if a swap is needed
  + \_indexToken: the index token of the position
  + \_collateralDelta: the amount of collateral in USD value to withdraw
  + \_sizeDelta: the USD value of the change in position size
  + \_isLong: whether the position is a long or short
  + \_receiver: the address to receive the withdrawn tokens
  + \_acceptablePrice: the USD value of the min (for longs) or max (for shorts) index price acceptable when executing the request
  + \_minOut: the min output token amount
  + \_executionFee: can be set to PositionRouter.minExecutionFee
  + \_withdrawETH: only applicable if WETH will be withdrawn, the WETH will be unwrapped to ETH if this is set to true
  + \_callbackTarget: an optional callback contract, this contract will be called on request execution or cancellation
* After this transaction is sent a keeper will execute the request, the request will either be executed or cancelled
* If the position cannot be decreased for reasons such as the \_acceptablePrice not being fulfillable then the request will be cancelled and there will be no change to the position
* \_minOut can be zero if no swap is required

**Positions List**

A list of position details can be retrieved by calling Reader.getPositions

* \_vault: the vault contract address
* \_account: the account of the user
* \_collateralTokens: an array of collateralTokens
* \_indexTokens: an array of indexTokens
* \_isLong: an array of whether the position is a long position

The returned positions will be in the order of the query, for example, given the following inputs:

* \_collateralTokens: [WBTC.address, WETH.address, USDC.address]
* \_indexTokens: [WBTC.address, WETH.address, WBTC.address]
* \_isLong: [true, true, false]

The position details would be returned for

* Long BTC position, positionIndex: 0
* Long ETH position, positionIndex: 1
* Short BTC position, positionIndex: 2

The returned array would be a list of values ordered by the positions:

* size
  + position size in USD
  + value at: positionIndex \* 9
* collateral
  + position collateral in USD
  + value at: positionIndex \* 9 + 1
* averagePrice
  + average entry price of the position in USD
  + value at: positionIndex \* 9 + 2
* entryFundingRate
  + a snapshot of the cumulative funding rate at the time the position was entered
  + value at: positionIndex \* 9 + 3
* hasRealisedProfit
  + 1 if the position has a positive realised profit, 0 otherwise
  + value at: positionIndex \* 9 + 4
* realisedPnl
  + the realised PnL for the position in USD
  + value at: positionIndex \* 9 + 5
* lastIncreasedTime
  + timestamp of the last time the position was increased
  + value at: positionIndex \* 9 + 6
* hasProfit
  + 1 if the position is currently in profit, 0 otherwise
  + value at: positionIndex \* 9 + 7
* delta
  + amount of current profit or loss of the position in USD
  + value at: positionIndex \* 9 + 8

**Buying / Selling GLP**

Buying and selling GLP can be done through the GlpRewardRouter.

To buy GLP, call mintAndStakeGlp

* \_token: the token to buy GLP with
* \_amount: the amount of token to use for the purchase
* \_minUsdg: the minimum acceptable USD value of the GLP purchased
* \_minGlp: the minimum acceptable GLP amount

To sell GLP, unstakeAndRedeemGlp

* \_tokenOut: the token to sell GLP for
* \_glpAmount: the amount of GLP to sell
* \_minOut: the minimum acceptable amount of tokenOut to be received
* \_receiver: the address to send tokenOut to
* Note that GLP can only be redeemed up to the reservedAmount, which is based on the amount of open interest, if the pool has been fully redeemed up to the reservedAmount then redeemers will need to wait for positions to close before further redemptions can be done, in this scenario the borrowing fee APR would be very high so liquidity providers will be incentivised to mint GLP and traders will be incentivised to close positions

The price of GLP can be retrieved from the [GlpManager](https://gmxio.gitbook.io/gmx/contracts#arbitrum).

* The buy price would be getAum(true) / glpSupply
* The sell price would be getAum(false) / glpSupply

glpSupply would be the totalSupply value of [0x4277f8F2c384827B5273592FF7CeBd9f2C1ac258](https://arbiscan.io/address/0x4277f8f2c384827b5273592ff7cebd9f2c1ac258).

**Transferring Staked GLP**

When GLP is bought it is automatically staked and when it is sold it is automatically unstaked, for integrations adding GLP the StakedGlp contract can be used to transfer staked GLP tokens.

​[StakedGlp](https://gmxio.gitbook.io/gmx/contracts" \l "arbitrum) behaves like a regular ERC20 token, the user can call approve on it to approve your contract, then your contract can call transferFrom to transfer the GLP tokens to any receiving account or contract. When transferring, the StakedGlp contract will unstake GLP from the user and stake the GLP for the receiving account, the receiving account or contract would then start earning rewards which can be compounded or claimed by calling handleRewards on the RewardRouter contract.

Since there is a 15 min cooldown duration after minting GLP, this amount of time needs to pass for the user before transferFrom can be called for their account.

**GLP Price**

The price of GLP is based on the total worth of all tokens in the pool and factors in pending profits and losses from all currently opened positions.

* Buy price: glpManager.getPrice(true)
* Sell price: glpManager.getPrice(false)

Since there might be a spread for token pricing, passing in true into the getPrice function returns the maximum price at that point in time, while passing in false returns the minimum price.

For the calculation of pending PnL for shorts the glpManager.shortsTracker.globalShortAveragePrices value should be used instead of vault.globalShortAveragePrices.

**Staking**

The RewardRouter contract handles the necessary actions needed for staking in a single transaction.

When staking GMX:

* The RewardRouter deposits the GMX token into the StakedGmxTracker contract
* The StakedGmxTracker issues itself as a token for each token deposited
* esGMX can similarly be deposited into the StakedGmxTracker
* The StakedGmxTracker distributes esGMX to staked tokens
* After this step, the RewardRouter deposits the StakedGmxTracker tokens into the BonusGmxTracker
* The BonusGmxTracker distributes Multiplier Points to staked tokens
* Finally the BonusGmxTracker tokens are deposited into the FeeGmxTracker which distributes ETH or AVAX to staked tokens

When minting GLP:

* The RewardRouter sends the funds to be deposited to the GlpManager and mints GLP tokens
* The RewardRouter then deposits the GLP tokens to the FeeGlpTracker which distributes ETH or AVAX to the staked tokens
* Finally the RewardRouter deposits the FeeGlpTracker tokens into the StakedGlpTracker which distributes esGMX to staked tokens

Addresses for contracts can be found at <https://github.com/gmx-io/gmx-interface/blob/master/src/Addresses.js>.

To get the deposit balances for an account you can use RewardTracker.depositBalances(account, token), or RewardReader.getDepositBalances(account, depositTokens, rewardTrackers).

To get claimable rewards you can use RewardReader.getStakingInfo(account rewardTrackers), this returns an array of uint256 values in the order:

* Claimable rewards
* Amount of reward token distribution per second
* Average staked amount for account
* Total rewards distributed to account
* Total staked tokens in the rewardTracker

**Transferring a Referral Code**

Steps to transfers a referral code:

1. 1.

Get the hash of the referral code using an online tool like: <https://www.devoven.com/string-to-bytes32> with the "Append Zeros" option checked, e.g. the hash of "code" would be "0x636f646500000000000000000000000000000000000000000000000000000000"

1. 2.

Open the ReferralStorage contract in a block explorer, using the ReferralStorage links above

1. 3.

Navigate to the "Write Contract" tab

1. 4.

Click "Connect to Web3" and connect the account that owns the referral code

1. 5.

Click on "setCodeOwner" and key in the hash from step 1 for code, key in the new owner address for newAccount

1. 6.

Click on "Write" to send the transaction

**Testnet Deployments**

The below contracts were deployed on the Arbitrum Testnet before [http://arbiscan.io](http://arbiscan.io/) was available, note that they are not actively maintained:

* Vault: [0xA4704fBfaf7c89511668052931Ba0f1816D2c9d3](https://testnet.arbiscan.io/address/0xA4704fBfaf7c89511668052931Ba0f1816D2c9d3)​
* Router: [0x526f42EA12E167E36E3De747187f53b6989d121A](https://testnet.arbiscan.io/address/0x526f42EA12E167E36E3De747187f53b6989d121A)