**Opyn v2 FAQ**

**What are options?**

Options give option buyers the right, but not the obligation to buy or sell the option's underlying asset at the strike price by the expiry date.

* An option that gives the option buyer the right to buy the underlying asset is called a *call option*
* An option that gives the option buyer the right to sell the underlying asset is called a *put option*

Options sellers earn a premium in return for taking on this obligation to either buy the underlying from or selling the underlying to the option buyer.

Here are some resources to learn more about options:

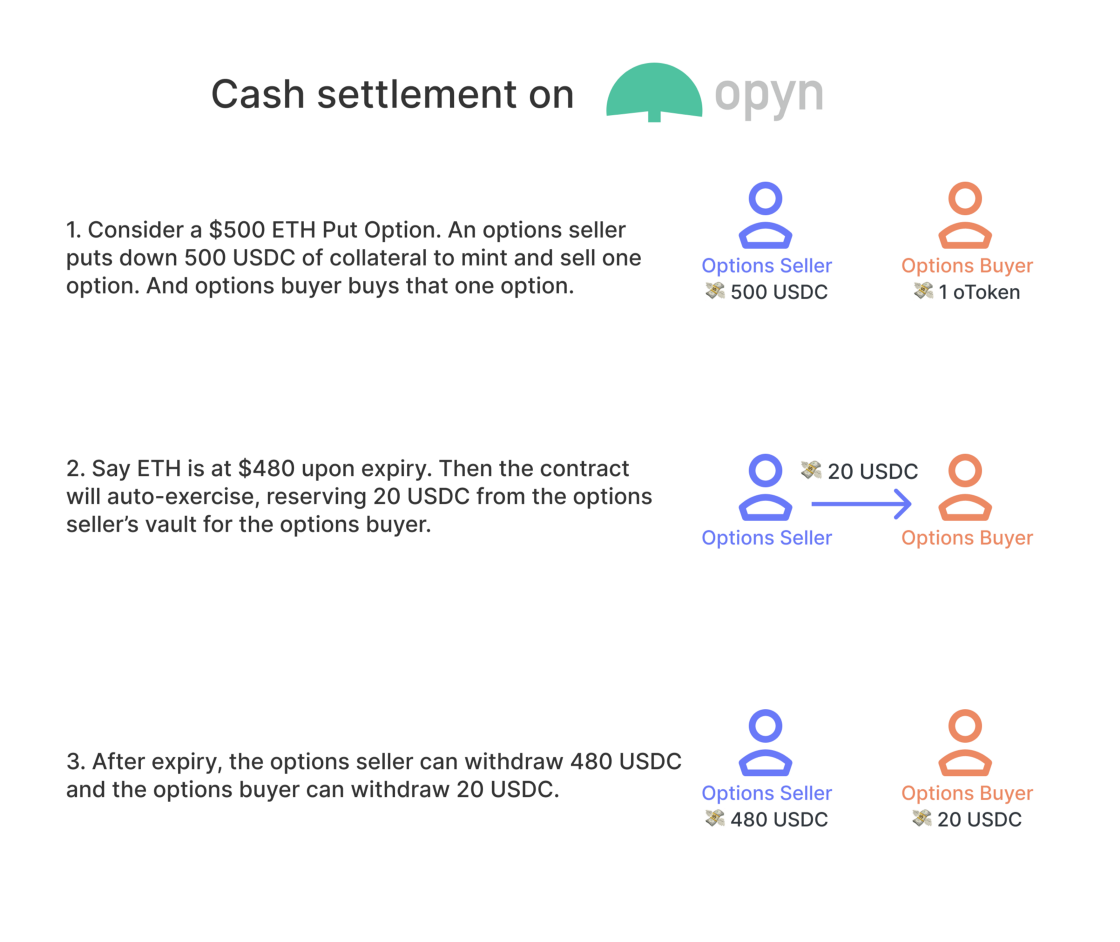
* ​[Khan Academy](https://www.khanacademy.org/economics-finance-domain/core-finance/derivative-securities)​
* ​[OptionAlpha](https://optionalpha.com/members/video-tutorials/options-basics)​
* ​[Investopedia](https://www.investopedia.com/options-basics-tutorial-4583012)​
* ​[Beginner's Guide to DeFi Options](https://medium.com/opyn/a-beginners-guide-to-defi-options-opyn-v2-4d64f91acc84)​

There are also lots of great conversations on our [discord](https://tiny.cc/opyndiscord), and we’re happy to answer any questions :)

**What kind of options are available on Opyn?**

Opyn v2 offers European, cash-settled options that auto-exercise upon expiry.

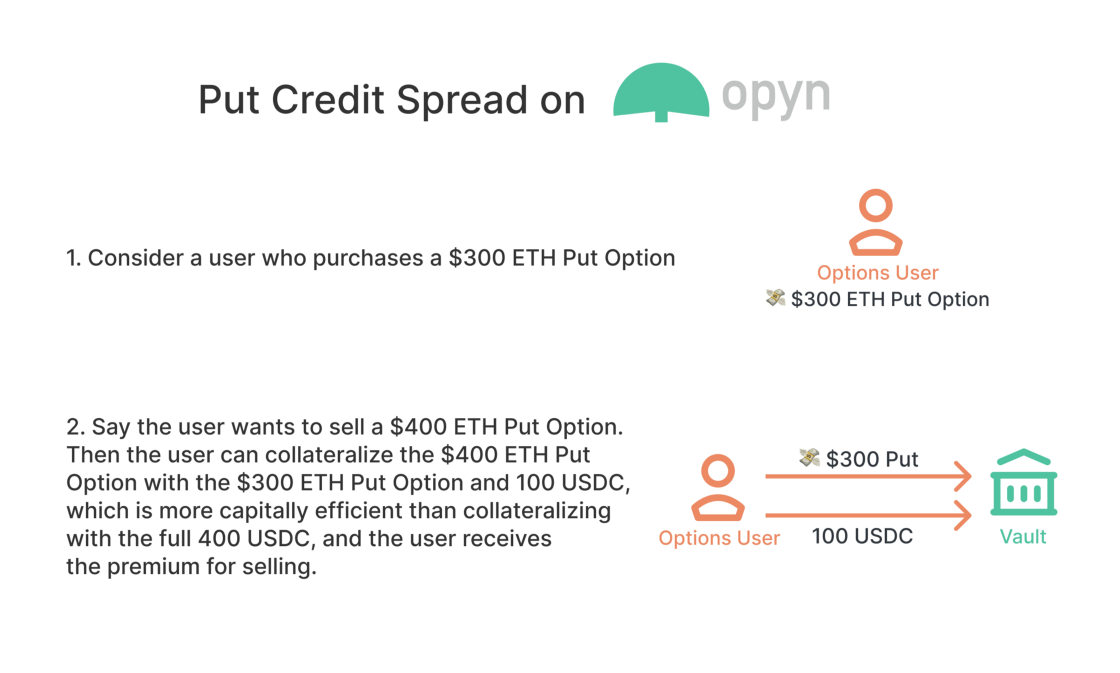
* European options mean that option holders can exercise options only upon expiry
* Cash settlement means that option holders don’t have to provide the underlying asset in order to exercise. Rather, the options are settled in the collateral asset, and option holders receive the difference between the price of the underlying asset at expiry and the strike price from option sellers.

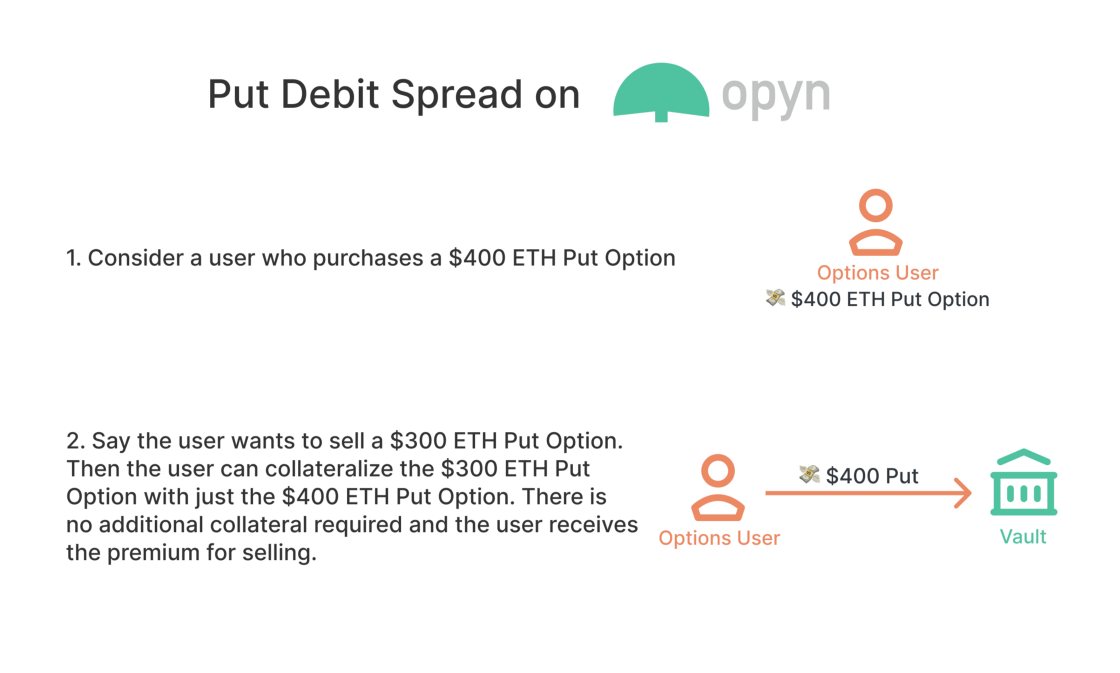


* The protocol now has auto-exercise for [in the money](https://www.investopedia.com/terms/i/inthemoney.asp) options, so option holders don’t need to take action before or at expiration. Upon expiry, proceeds for long and short option holders are calculated and can be redeemed at any point after the proceeds have been finalized with a settlement price. Users can redeem by clicking a "redeem" button on the interface after expiry.

**What is a spread?**

Spreads enable long oTokens to collateralize short oTokens, enabling users to post the max loss of a structure as collateral.





**Trading**

**Where do oTokens trade?**

Opyn options (oTokens) are ERC20s, so they can be trading on any decentralized exchange that follows the ERC20 standard.

On [opyn.co](http://opyn.co/) interface they trade on 0x.

**How are oTokens priced?**

The pricing of each oToken depends on the exchange where it is trading.

On the [opyn.co](http://opyn.co/) interface where oTokens trade on 0x, market makers are placing bids and asks.

**Why can't I change the gas price for buy or sell transactions on the opyn.co interface?**

On the opyn.co interface oTokens trade on 0x. This means that each trade requires a 0x fee which is a function of gas price. That fee needs to be sent in the transaction, so increasing the gas price alters the 0x fee causing the transaction to fail. You can lower the gas price, however lowering the gas price could cause the transaction to take too long, meaning the 0x order will expire and then the transaction will fail. For the transaction to go through we recommend confirming with the pre-set gas price.

**What is market impact?**

Market impact refers to the increase in the average price of a token as the size of a market order gets larger. For larger orders, there might not be enough oTokens available at the best bid or the best ask. In this case, in order to fill the complete order, the price of some of the oTokens will be at a less favorable bid / ask.

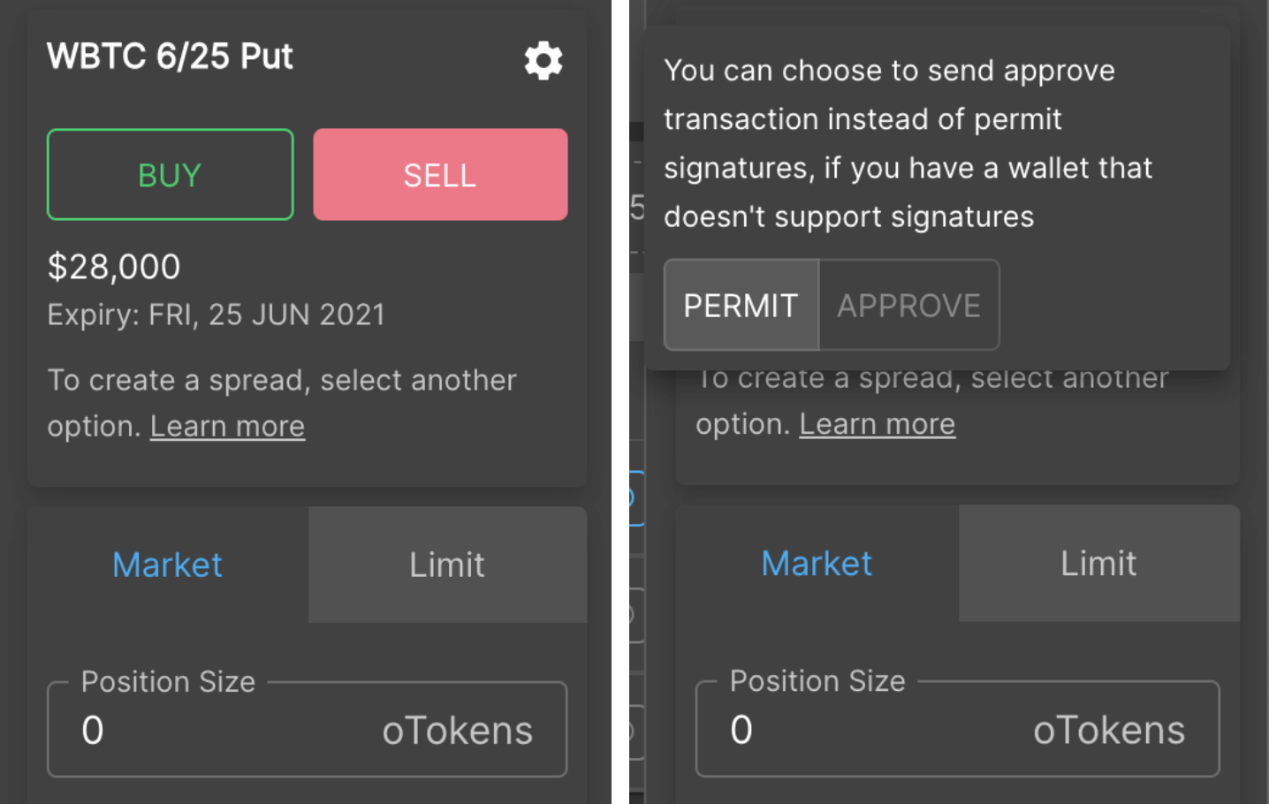
For example, say there is an orderbook with an ask for 15 oTokens at 100 USDC and an ask for 5 oTokens at 105 USDC. Then, if you purchase up to 15 oTokens, you can purchase each oToken for 100 USDC. However, say you wanted to purchase 17 oTokens. Then you could purchase 15 for 100 USDC and 2 for 105 USDC, so the average price for each oToken would be 100.59 USDC, meaning a 0.59% market impact.

**How does the opyn.co interface calculate total return?**

Total return is calculated as the difference between the initial premium and the current premium, where the current premium is the best bid or ask available. In the case that an option's in-the-money amount is greater than the amount you could sell the option for, the in-the-money amount is used to calculate the total return rather than the current premium.

**My hardware wallet isn't working with permit, what do I do?**

Some hardware wallets don't support permit signatures. In order to use a normal "approve" instead of permit signatures, you can click the "gear" icon on the top right of the order ticket and select "approve."



**Do you have any tutorials on how to use the platform?**

Yes! You can check them out here:

* ​[Opyn v2 Tutorial](https://medium.com/opyn/how-to-trade-defi-options-opyn-v2-tutorial-aabc0a323430)​
* ​[Limit Order Tutorial](https://medium.com/opyn/opyn-v2-limit-order-tutorial-10efe115ac50) ​
* Partial Collateralization Tutorial (coming soon!)

**Limit Orders**

**How do limit orders work?**

Limit orders on opyn.co go through 0x and placing limit orders on Opyn is free, the taker of the order pays fees.

* Users must lock up collateral for sell limit orders
* Users can cancel limit orders at any time by returning to the orderbook
* Cancellation of limit orders costs gas
* Orders will be cancelled automatically after the order deadline, and users will not need to pay gas for expired orders
* Orders may be fully filled or partially filled by other users or market makers
* Executed limit orders will appear in the Opyn dashboard
* Users will be notified on screen when a limit order executes, not when a limit order expires unfilled
* Limit orders may not execute immediately. Cancellation of orders and filling orders cost gas, which can cause orders that are crossed to sometimes occur due to gas and 0x fees

To learn more about how to create limit orders on opyn.co, check out [this tutorial](https://medium.com/opyn/opyn-v2-limit-order-tutorial-10efe115ac50).

**Why are small limit orders less likely to be taken?**

Takers have to pay a [0x protocol fee](https://0x.org/docs/guides/v3-upgrade-guide#protocol-fee) when they buy or sell options. Takers have to pay this fee with each order they fill. For example:

* If there was an ask for 5 oTokens and the taker wanted to purchase 5 oTokens they would only have to pay the 0x protocol fee once.
* However, if there were 5 asks for 1 oToken each, the taker would have to pay the 0x protocol fee 5 times.

Because of the 0x protocol fee, orders are filled based on their total cost (bid/ask price + 0x protocol fee), to reduce the total cost to the taker.

**Building on Gamma**

**Can I create a new oToken?**

Yes, anyone can create new oTokens if a product has been whitelisted. A product is a combination of specifying the underlying asset, strike asset, and collateral asset for an option and whether it's a call or a put. For any of these whitelisted products, anyone can create a new option, specifying the strike and expiry. Expiration times are currently fixed to 8:00 AM UTC to prevent fragmentation of liquidity across a variety of expirations within the same day.

**What can I build on Gamma?**

Opyn v2 allows you to create put and call options with spreads for capital efficiency, and is great for applications ranging from protection and hedging to taking views on different cryptocurrencies. Options are an incredibly versatile financial instrument - in fact you can create [any financial payoff using just put and call options.](https://www.youtube.com/watch?v=rMsu4v-UlkA&feature=youtu.be&ab_channel=MITOpenCourseWare)​

Check out [these use cases](https://opyn.gitbook.io/opyn/getting-started/use-cases) to learn more about what you can build, and [jump into Discord - we'd love to jam on #dev](https://tiny.cc/opyndiscord)!

**Where can I get help?**

Please join the #dev channel in the Opyn community [Discord server](https://tiny.cc/opyndiscord). We're always happy to help, so don't hesitate to ask questions!

**Under the Hood**

**What is the Gamma Protocol?**

‌The [Gamma Protocol](https://github.com/opynfinance/GammaProtocol) is Opyn v2, the most capital efficient on-chain options protocol. You can access the smart contracts here.

**What are oTokens?**

‌oTokens are ERC20 tokens that represent options that you have bought or sold. Each oToken corresponds to one unit of the underlying asset. Eg. 1 oETHp is a put option on 1 ETH

**How does auto-exercise work?**

The protocol now has auto-exercise for in the money options, so option holders don’t need to take action before or at expiration. Upon expiry, proceeds for long and short option holders are calculated and can be redeemed at any point after the proceeds have been finalized with a settlement price.

(Please note that auto-exercise is ONLY available for v2. For v1, you must exercise before expiry here. You can see the [v1 FAQ here](https://opyn.gitbook.io/opynv1/).)

At the time of expiry, Chainlink nodes provide a live ETHUSD price to the Chainlink aggregator. Opyn's contracts have a short locking period as it takes a short amount of time for the Chainlink oracle price for the expiry time of the options to be available. After the locking period has passed, an oracle price can be submitted. After a price has been submitted, there is a dispute period in which the price can be disputed. If the price is disputed, the disputer can update with a new price. After the dispute period has passed, the settlement value of options are finalized and users can redeem option proceeds or settle their vaults.

The locking period and dispute time periods are different for different assets. Currently WETH-USDC options are available on Opyn v2. For WETH-USDC options, the locking period is 5 minutes and the dispute period is 1 hour. So while the options all expire at 8:00 UTC and are settled with the price of the underlying at 8:00 UTC, users can start to redeem at 10:05 UTC.

**How do the oracles work?**

Cash settlement requires an oracle to determine the payout at expiry. The Gamma protocol architecture is generalizable to allow for different oracles for different assets. We will initially be launching ETH-USDC options collateralized with USDC. These will use the Chainlink oracle to get the ETH price. There are no liquidations needed as max loss is posted as collateral.

**What is a flash mint?**

Since vault collateralization is checked at the end of a transaction, you can mint options without collateral as long as they are burned before the end of the transaction.

Some use cases for flash mint include:

* Arbitrage between dexes or exchanges
* Allows order of operations of transactions not to matter, you can mint and sell options first, and then add collateral in the same transaction
* Allows users to do interesting operations like deposit partial collateral and use the premium received as the rest of the collateral (ie sell options before they are fully collateralized and then fully collateralize using the proceeds of the sale)
* Probably many more cool things that we haven't thought of yet!

All of these actions have to be done atomically, within a single transaction. Collateralization is checked at the end of all of these operations/actions and if the collateralization is not correct, it is as if the prior actions/operations never happened and the transaction reverts.

**What is an operator?**

Operators are a smart contract feature that allow users to delegate control of their vaults to a third party smart contract. This could be a smart contract that rolls over their options for them, a fund manager to do trades for them, or a vast possibility of other interactions that developers can build on top of the protocol.

Operators have full control over user funds and can take any action on behalf of a user. The only action operators cannot take is to add new operators or remove new operators.

**Partial Collateralization and Liquidations**

**How does partial collateralization work?**

Users can now choose if they want to fully collateralize or partially collateralize a minted option. If a user chooses to partially collateralize their trade:

* For calls, less than 1 underlying asset can be posted as collateral
* For puts, less than strike can be posted as collateral

With partial collateralization, excess capital is now free to either standby risk free or be deployed in another trade.

Benefits of Partial Collateralization

* Increased upside leverage
* Enhanced capital efficiency
* More portfolio flexibility

Risks Associated with Partial Collateralization

* Greater downside leverage
* Under-collateralized positions will lead to forced liquidations
* Possible to lose 100% of original investment prior to expiration

Margin is calculated using only a spot price, a shock to spot parameter, a conservative assumption on vol/premium, and time to expiry to give a worst case bound on the option premium. Check out [this post](https://medium.com/opyn/partially-collateralized-options-now-in-defi-b9d223eb3f4d) to learn more.

**How do liquidations work?**

Sellers must maintain a minimum amount of collateral in their vault to secure the options they have sold. If a seller fails to do so, their vault may be liquidated and their collateral will be seized and auctioned off to repay their debts. The liquidation mechanism is a reverse dutch auction that is triggered via a Chainlink pricer with a specific timestamp.

The reverse dutch auction serves as the price discovery mechanism for Opyn liquidations.The reverse dutch auction starts at a low price and then the price increases over time - liquidators will execute the trade when it is profitable. Check out [this post](https://medium.com/opyn/partially-collateralized-options-now-in-defi-b9d223eb3f4d) to learn more.

**Why is there a minimum size required for partially collateralized positions?**

In order to ensure the safety of the system and make sure liquidators are incentivized to liquidate vaults even in high gas environments, we must have a minimum size required to create a partially collateralized vault. This is similar to MakerDAO's dust, which is the minimum requirement for opening a Maker vault to mint DAI. The current minimums are 1 ETH for selling call options and 2500 USDC for selling put options.

**Opyn v1**

**What is happening to Opyn v1?**

Opyn v1 laid the foundation for DeFi options as the first live ERC20 options protocol. For the first time, anyone could create, buy, and sell options on any ERC20 token. Opyn v1 will remain live on the Ethereum network, providing a venue for American, physically settled options. We will continue to launch ERC20 options on Opyn v1.

**Where can I find more info on v1?**

You can access the Opyn v1 [developer docs and FAQ here](http://opyn.gitbook.io/opyn/). You can interact with the smart contracts via Etherscan, the [Opyn v1 interface](http://v1.opyn.co/), and [OpynMonitor](http://opynmonitor.xyz/), a community built interface.

**Security**

**What is the security of the Gamma protocol?**

‌The security of the Opyn protocol is our highest priority. Our team has created a protocol that we believe is safe and dependable, and has been audited by OpenZeppelin and formally verified by Certora. All smart contract code is publicly verifiable. You can find the OpenZeppelin audit report here, the Certora report here, and you can find our [bug bounty here](https://opyn.gitbook.io/opyn-v2/get-started/security#bug-bounty-program).

We encourage our users to be mindful of risk and only use funds they can afford to lose. Options are complex instruments that when understood correctly can be powerful hedges. Smart contracts are still new and experimental technology. We want to remind our users to be optimistic about innovation while remaining cautious about where they put their money.

**Does the protocol have any privileged roles?**

The protocol currently has the following privileged roles:

* Owner: can whitelist/blacklist collateral and oToken addresses, update/upgrade modules, set addresses for other roles in the system, and manage oracle parameters.
* Full Pauser: can fully pause the system in case of an emergency.
* Partial Pauser: can pause all actions other than redeem oToken and settle vault.
* Farmer: can withdraw any token excess balance in the pool.
* Pricer: can submit oracle prices
* Disputer: can dispute oracle prices

**Help! I can’t access Opyn!**

‌Opyn’s smart contracts are on the Ethereum blockchain and are thus always available. If Metamask, or the [Opyn](http://opyn.co/) interface are unavailable, you can always [access Opyn through the smart contracts](https://opyn.gitbook.io/opyn-v2/get-started/abis-smart-contract-addresses)​[.](https://opyn.gitbook.io/opyn/abis-smart-contract-addresses) We also encourage the community to build interfaces to the Opyn v2 protocol.

**Introduction**

**What are Options**

Options give option buyers the right, but not the obligation to buy or sell the option's underlying asset at the strike price by the expiry date. Options sellers earn a premium in return for taking on this obligation to either buy the underlying from or selling the underlying to the option buyer.

* An option that gives the option buyer the right to buy the underlying asset is called a *call option*
* An option that gives the option buyer the right to sell the underlying asset is called a *put option*

Here are some resources to learn more about options:

* ​[Khan Academy](https://www.khanacademy.org/economics-finance-domain/core-finance/derivative-securities)​
* ​[OptionAlpha](https://optionalpha.com/members/video-tutorials/options-basics)​
* ​[Investopedia](https://www.investopedia.com/options-basics-tutorial-4583012)​

There are also lots of great conversations on our [discord](https://tiny.cc/opyndiscord), and we’re happy to answer any questions :)

**Why do people use options**

People often use options for the following three uses cases:

* Hedging
* Leverage
* Yield

Let's break down how.

**Hedging**

People often buy put options in order to hedge downside risk.

For example if ETH is currently $2000 and you're worried about ETH price crashing, you can buy a put option with a strike price of $1800. Then say ETH crashes to $1000, you can still sell your ETH for $1800 using your put option. While you own the put option, you've basically created a price floor of $1800 as the lowest amount you'll sell your ETH at.

**Leverage**

People often buy call options to gain leverage without having any liquidation risk.

For example if ETH is currently at $2000 and you think ETH price is going to increase, you could buy a call option with a strike price of $2500. This call option would likely cost less than $100 compared to buying 1 ETH at $2000. Then say ETH runs up to $4000, you buy ETH at $2500 using your call option. There is no liquidation risk, because you are not putting down any collateral to borrow - you are simply paying a premium to buy an option.

**Yield**

People often sell put or call options to earn "high yield".

In order to sell a put or call option, you put down collateral, mint an option, and sell that option. When you sell the option you earn a premium. Because selling options is higher risk than say lending in a money market, since you could be exercised on if the option hits the strike price, you earn a higher premium for taking on that risk, which people often view as "high yield".

For example if you think ETH isn't going to move too much, you could sell a covered call on ETH earning the premium ("high yield" on ETH), and if you aren't exercised, can leave with your original ETH collateral and your premium. However, it is important to remember the additional risk that comes with these positions.

**Protocol Introduction**

Opyn V2 (Gamma Protocol), allows anyone to buy, sell, and create options on any ERC20 asset.

Gamma Protocol is the most capital efficient and flexible options protocol allowing for:

* ​[Partially collateralized](https://medium.com/opyn/partially-collateralized-options-now-in-defi-b9d223eb3f4d) options
* ​[Spreads](https://opyn.gitbook.io/opyn/#what-is-a-spread)
* ​[Flash Minting](https://opyn.gitbook.io/opyn/#what-is-a-flash-mint) oTokens
* ​[Operators](https://opyn.gitbook.io/opyn/#what-is-an-operator) (useful for rolling over vaults, creating perpetual positions, and more)
* Option strategies
  + In addition to buying & selling single options outlined above you can also hedge / leverage / earn yield with combined positions like [straddles & strangles](https://www.investopedia.com/ask/answers/05/052805.asp), [the wheel](https://seekingalpha.com/instablog/1046492-markus-heitkoetter/5514813-wheel-option-strategy-example), [principal protected notes](https://www.investopedia.com/terms/p/principalprotectednote.asp) and more

Opyn v2 options are **cash settled, European** options.

* Cash settled: options are settled in the collateral asset, and option holders receive the difference between the price of the underlying asset at expiry and the strike price from option sellers.
* European: option holders can exercise options only upon expiry

**Use Cases**

**Ideas to Build**

Options are an incredibly versatile financial instrument - in fact you can create [any financial payoff using just put and call options.](https://www.youtube.com/watch?v=rMsu4v-UlkA&feature=youtu.be&ab_channel=MITOpenCourseWare) See the introduction to learn more about [why people use options](https://opyn.gitbook.io/opyn/getting-started/introduction#why-do-people-use-options).

We're excited to see what you'll build! Here are some ideas to get started:

Join #devon [**Discord**](https://discord.gg/ugAv3SH) to jam on these ideas and more!

**Hedging for Uniswap LPs**

Uniswap LPs can help reduce impermanent loss in ETH:Stablecoin pools using [straddles](https://www.investopedia.com/terms/s/straddle.asp) (put and call with same strike) and [strangles](https://www.investopedia.com/terms/s/strangle.asp) (put and call with different strike).

**Liquidation Saver**

With the [partial collateralization update](https://medium.com/opyn/partially-collateralized-options-now-in-defi-b9d223eb3f4d), users who choose to partially collateralize positions are at risk of liquidation. Say 10 ETH is required to fully collateralize a position and 4 ETH to partially collateralize. You could create a smart contract into which a user deposits the 10 ETH, where 4 ETH go to collateralizing the options position and the other 6 ETH earn interest (eg. in Yearn). If the option position gets close to undercollateralization, the contract can pull some ETH from Yearn to top up the option collateral. You can [view this doc to see how liquidations work](https://www.notion.so/opynopyn/Gamma-Protocol-Liquidations-1ffd204e403245199a433b98c5cc613b).

**Liquidator Bot**

The [partial collateralization update](https://medium.com/opyn/partially-collateralized-options-now-in-defi-b9d223eb3f4d) includes liquidations, so you can build a liquidator bot to liquidate undercollateralized positions.

* ​[How liquidations work](https://www.notion.so/opynopyn/Gamma-Protocol-Liquidations-1ffd204e403245199a433b98c5cc613b)​
* ​[Reference Liquidator Bot Implementation](https://github.com/opynfinance/Gamma-Liquidator)​

**Liquidation Interface**

The [partial collateralization update](https://medium.com/opyn/partially-collateralized-options-now-in-defi-b9d223eb3f4d) includes liquidations, so you can build an interface that allows anyone to liquidate positions, even if they haven't built a liquidator bot. You can [view this doc to see how liquidations work](https://www.notion.so/opynopyn/Gamma-Protocol-Liquidations-1ffd204e403245199a433b98c5cc613b).

**Rollovers**

Allow users to rollover their options from one expiry until the next. For example, if a user holds on to an option that expires on Oct. 30, give them the ability to have that option automatically rollover to expire on a date in the future eg. Nov 30. You can accomplish this relatively simply using the new “operator” functionality in Opyn v2, where users can delegate vault actions to another smart contract.

**Portfolio Managers**

With Opyn v2’s new “[operator](https://opyn.gitbook.io/opyn/get-started/protocol-overview-and-glossary-of-terms#operators)” functionality, users can delegate out portfolio management to dedicated portfolio managers. These managers could be individuals or smart contracts that employ specific strategies.

**Structured Products**

You can use options in combination with other financial primitives to build interesting [structured products](https://www.investopedia.com/articles/optioninvestor/07/structured_products.asp). For example, you could create a [principal protected note](https://www.investopedia.com/terms/p/principalprotectednote.asp), where you attach a call or put option to an ERC-20. One way this could work to go to a money market (eg. Compound, Aave), look at the fixed rate lending rates, and deposit an amount (say 0.99 USDC) that yields 1 USDC at expiry. Then you could use the remaining 0.01 USDC to buy a call option. The user’s upside exposure would be based on the 0.01 and the price of a call option. Another strategy could be [the wheel](https://seekingalpha.com/instablog/1046492-markus-heitkoetter/5514813-wheel-option-strategy-example), where you sell puts to collect premium, and if the puts are exercised, sell calls. There is a set of [developer templates](https://opyn.gitbook.io/perp-vault/) available to get started with building these kinds of products.

**Physical Settlement Operator**

Opyn v2 options are cash settled in their collateral asset. However, some users prefer to received the underlying asset if they are exercised. This has been a request especially for put sellers who would prefer to receive ETH rather than USDC cash settlement. You can build an [operator](https://opyn.gitbook.io/opyn/get-started/protocol-overview-and-glossary-of-terms#operators) that takes a put seller's USDC and purchases ETH with it, if exercised.

**Auto Redeem Operator**

Currently, Opyn users must return to [Opyn.co](http://opyn.co/) to redeem their collateral after an option expires. Using Opyn's [operator](https://opyn.gitbook.io/opyn/get-started/protocol-overview-and-glossary-of-terms#operators) function, you can build a smart contract that automatically redeems oTokens for users after expiration.

**OTC oTokens Interface**

To avoid slippage, a lot of large oToken users are looking for ways to conduct OTC trades for oTokens. You could facilitate this using 0x as a settlement layer, building a simple interface for parties to interact with each other while preserving anonymity — this could be something similar to what [Boxswap](https://boxswap.io/) does for OTC NFT trading.

**Simplified Interface for options beginners**

Options can be intimidating. You can create an interface that breaks down options into simple steps (i.e. Do you want to earn income or speculate? → 2 Do you think the price of ETH will be higher or lower on X date, etc.) to help new options traders better understand how to trade options. This could also be a beginner options interface that has more educational content displaying on the front end. The purpose of this interface could be to make options trading more approachable.

**Advanced Interface for quant traders（量化交易员）**

Advanced options traders need access to calculations, figures, and charts that might be too confusing for beginner or intermediate traders. Examples of more advanced trading platforms are [Deribit](https://www.deribit.com/), [dYdX](https://dydx.exchange/), [LedgerX](https://www.ledgerx.com/options) or more traditional options trading platforms such as [ThinkorSwim](https://www.tdameritrade.com/tools-and-platforms/thinkorswim/desktop.page).

**Position Builder**

To help users better understand options, you could create an options "position builder," by asking basic questions (e.g. do you think ETH will be above $2000 on March 18). Once a user has gone through the position builder steps, allow the trader to buy or sell the option by directly linking them to that option options on [Opyn.co](http://opyn.co/)​

**Gain & Loss Calculator**

To help users better understand options' potential gain / loss under various market conditions, you could create a gain loss calculator / simulator for each option position on [Opyn.co](http://opyn.co/).

**Options Trading Leaderboard (e.g.** [**gamified leaderboards**](https://matcha.xyz/moolah)**)**

To add a gamification element to options trading, you could create a leaderboard that ranks users' addresses by trading volume, # of contracts traded, or another metric.

**SDK for Gamma Protocol**

You could create a python / js / rust library to interact with Gamma Protocol. This could incldue setting up some basic architecture, testing frameworks, and having wrapping functions to batch actions to help other developers integrating with opyn.

**Volatility Oracle**

Using put and call options you can develop a volatility oracle like the [VIX](https://www.investopedia.com/ask/answers/021015/what-cboe-volatility-index-vix.asp), which tracks volatility in traditional finance.

**Vesting Call Options Interface**

Create and interface of people to deploy vesting call options, similar to equity options that startups typically use to incentivize employees.This allows DAOs to grant vesting options to any contributor. [Opyn V1 physically settled options](https://opyn.gitbook.io/opynv1/) work better for this use case. [Here are instructions](https://gist.github.com/alexisgauba/cf055f521bcfded018116f3f37074f67) on how to set this up using Etherscan, but an interface would make this much easier and more accessible to projects.

**Existing Projects**

You can check out the projects that have integrated Opyn here! A number of them have open-source codebases that you can use to learn from as well.

**Gamma Portal**

Gamma Portal is an open-sourced alternative front end to interact with [Gamma protocol](https://github.com/opynfinance/GammaProtocol)​

* ​[Site](https://gammaportal.xyz/#/)
* ​[Github](https://github.com/antoncoding/gamma-portal)

**Ribbon**

Ribbon Finance uses financial engineering to create structured products that deliver sustainable yield

* ​[Site](https://www.ribbon.finance/)
* ​[Github](https://github.com/ribbon-finance)

Gamma protocol

# Taking Actions

## Protocol Actions

The main way to interact with the protocol is through a single function called [operate](https://github.com/opynfinance/GammaProtocol/blob/386386bac50e24816931190a243e1f220d043c29/contracts/core/Controller.sol#L411). This operate function takes in [actions](https://github.com/opynfinance/GammaProtocol/blob/master/contracts/libs/Actions.sol) as parameters, where each action specifies what you'd like to do (eg. add collateral, mint options).

This pattern allows you to string together multiple actions into one operate call to achieve many interactions with one call (eg. creating a short position by putting down collateral and minting options).

Additionally, collateralization is only checked at the end of an operation, allowing for flash mint functionality, where you mint an option, sell it, and then use the proceeds（收入） from selling as collateral.

For how to sell oTokens once they've been minted, or ways to buy oTokens [see Trading oTokens](https://opyn.gitbook.io/opyn/get-started/trading-otokens). oToken trades are not done through the core protocol, but through any venue that supports ERC20 trading.

## Action Structure

To use the operate function in [Controller](https://github.com/opynfinance/GammaProtocol/blob/master/contracts/core/Controller.sol), you need to input an array of actions. An [Action struct](https://github.com/opynfinance/GammaProtocol/blob/386386bac50e24816931190a243e1f220d043c29/contracts/libs/Actions.sol#L29) is defined as follows:

struct ActionArgs {

ActionType actionType;

address owner;

address secondAddress;

address asset;

uint256 vaultId;

uint256 amount;

uint256 index;

bytes data;

}

* actionType: type of action that is being performed on the system.
* owner: vault owner's address
* vaultId: the id of the vault to be modified
* secondAddress: another address field (usage dependent on action type)
* asset: asset that is being transferred.
* amount: amount of asset
* index: always 0 for the current version.
  + each vault can hold multiple short / long / collateral assets but we are restricting the scope to only 1 of each in this version. In future versions this would be the index of the short / long / collateral asset that needs to be modified
* data: used for arbitrary function calls

## Available Actions

To create an action, you must specify actionType and **the fields required for that specific action**. You can leave the other fields as default values.

//types of actions available

enum ActionType {

OpenVault,

MintShortOption,

BurnShortOption,

DepositLongOption,

WithdrawLongOption,

DepositCollateral,

WithdrawCollateral,

SettleVault,

Redeem,

Liquidate,

Call

}

### OpenVault

Open a new vault for an account.

#### Required Fields:

* owner : the vault owner.
* vaultId: the new vault Id to open. For the first vault, the vaultId is 1.

JavaScript

const ZERO\_ADDRESS = "0x0000000000000000000000000000000000000000"

const args = [{

actionType: ActionType.OpenVault,

owner: "0xcc5d905b9c2c8c9329eb4e25dc086369d6c7777c",

secondAddress: ZERO\_ADDRESS,

asset: ZERO\_ADDRESS,

vaultId: 1, // open the first vault

amount: 0,

index: 0,

data: ZERO\_ADDRESS,

}]

await controller.operate(args)

### DepositCollateral

Deposit collateral asset into a vault. This allow you to mint options against the vault.

#### Required Fields:

* owner : the vault owner.
* vaultId: the vault id to deposit into
* secondAddress: the address depositing the collateral assets. It can only be either msg.senderor the owner (if msg.sender is an [**operator**](https://opyn.gitbook.io/opyn/get-started/protocol-overview-and-glossary-of-terms#operators) of owner).
* asset: the address of the **collateral** you want to deposit.
* index: 0
* amount: amount to deposit

JavaScript

const ZERO\_ADDRESS = "0x0000000000000000000000000000000000000000"

const USDC = "0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48"

const owner = "0xcc5d905b9c2c8c9329eb4e25dc086369d6c7777c"

const args = [{

actionType: ActionType.DepositCollateral,

owner: owner,

secondAddress: owner,

asset: USDC,

vaultId: 1, // deposit to the first vault

amount: 100 \* 1e6, // deposit 100 USDC

index: 0,

data: ZERO\_ADDRESS,

}]

//approve collateral to margin pool

await ERC20(USDC).approve(marginPool.address, 100 \* 1\*6)

await controller.operate(args)

### WithdrawCollateral

Withdraw collateral from a vault. (Opposite of DepositCollateral)

#### Required Fields:

* owner : the vault owner.
* vaultId: the vault id to withdraw collateral from
* secondAddress: destination address to withdraw collateral to.
* asset: the address of the **collateral** you want to withdraw
* index: 0.
* amount: amount to withdraw

JavaScript

const ZERO\_ADDRESS = "0x0000000000000000000000000000000000000000"

const USDC = "0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48"

const owner = "0xcc5d905b9c2c8c9329eb4e25dc086369d6c7777c"

const args = [{

actionType: ActionType.WithdrawCollateral,

owner: owner,

secondAddress: owner, // withdraw to the owner address

asset: USDC,

vaultId: 1,

amount: 100 \* 1e6, // withdraw 100 USDC

index: 0,

data: ZERO\_ADDRESS,

}]

await controller.operate(args)

### DepositLongOption

Deposit an oTokens into a vault. This allows you to create [spreads](https://opyn.gitbook.io/opyn/#what-is-a-spread) (using an option to collateralize other options).

#### Required Fields:

* owner : the vault owner.
* vaultId: the vault id to deposit into
* secondAddress: the address depositing the oTokens. It can only be either msg.senderor the owner (if msg.sender is an [**operator**](https://opyn.gitbook.io/opyn/get-started/protocol-overview-and-glossary-of-terms#operators) of owner).
* asset: the address of the **oToken** you want to deposit
* index: 0.
* amount: amount to deposit

JavaScript

const ZERO\_ADDRESS = "0x0000000000000000000000000000000000000000"

const ETHUSD200Put = "0xde99ea535749f02da84d13e6f8253291e32d3a7f"

const owner = "0xcc5d905b9c2c8c9329eb4e25dc086369d6c7777c"

const args = [{

actionType: ActionType.DepositLongOption,

owner: owner,

secondAddress: owner,

asset: ETHUSD200Put,

vaultId: 1, // deposit some option to the first vault

amount: 100 \* 1e8, // deposit 100 oToken

index: 0,

data: ZERO\_ADDRESS,

}]

// approve oToken collateral to margin pool

await ERC20(ETHUSD200Put).approve(marginPool.address, 100 \* 1\*8)

await controller.operate(args)

### WithdrawLongOption

Withdraw a long oTokens from a vault. (Opposite of DepositLongOption)

#### Required Fields:

* owner : the vault owner.
* vaultId: the vault id to withdraw from
* secondAddress: destination address to withdraw long oTokens to.
* asset: the address of the **oToken** you want to withdraw
* index: 0.
* amount: amount to withdraw

JavaScript

const ZERO\_ADDRESS = "0x0000000000000000000000000000000000000000"

const ETHUSD200Put = "0xde99ea535749f02da84d13e6f8253291e32d3a7f"

const owner = "0xcc5d905b9c2c8c9329eb4e25dc086369d6c7777c"

const args = [{

actionType: ActionType.WithdrawLongOption,

owner: owner,

secondAddress: owner,

asset: ETHUSD200Put,

vaultId: 1,

amount: 100 \* 1e8, // deposit 100 oToken

index: 0,

data: ZERO\_ADDRESS,

}]

await controller.operate(args)

### MintShortOption

Mint oTokens from a vault.

#### Required Fields:

* owner : the vault owner.
* vaultId: the vault id to mint from
* secondAddress: destination address that receives the minted oTokens (most often the owner who is minting)
* asset: the address of the oToken you want to mint
* index: 0.
* amount: amount to mint

JavaScript

const ZERO\_ADDRESS = "0x0000000000000000000000000000000000000000"

const ETHUSD250Put = "0xde99ea535749f02da84d13e6f8253291e32d3a7f"

const owner = "0xcc5d905b9c2c8c9329eb4e25dc086369d6c7777c"

const args = [{

actionType: ActionType.MintShortOption,

owner: owner,

secondAddress: owner,

asset: ETHUSD250Put,

vaultId: 1, // mint from the first vault

amount: 100 \* 1e8, // mint 100 oToken

index: 0,

data: ZERO\_ADDRESS,

}]

await controller.operate(args)

### BurnShortOption

Burn oTokens from a vault.

#### Required Fields:

* owner : the vault owner.
* vaultId: the vault id to mint from
* secondAddress: address from which the oTokens are transferred (most often owner address)
* asset: the address of the oToken you want to burn
* index: 0.
* amount: amount to burn

JavaScript

const ZERO\_ADDRESS = "0x0000000000000000000000000000000000000000"

const ETHUSD250Put = "0xde99ea535749f02da84d13e6f8253291e32d3a7f"

const owner = "0xcc5d905b9c2c8c9329eb4e25dc086369d6c7777c"

const args = [{

actionType: ActionType.BurnShortOption,

owner: owner,

secondAddress: owner,

asset: ETHUSD250Put,

vaultId: 1, // burn from the first vault

amount: 100 \* 1e8, // burn 100 oToken

index: 0,

data: ZERO\_ADDRESS,

}]

await controller.operate(args)

### SettleVault

Settle a vault after expiry. If the vault contains minted oTokens and collateral, this function will payout the excess collateral based on spot price at expiry.

#### Required Fields:

* owner : the vault owner.
* vaultId: the vault id to mint from
* secondAddress: the address to receive collateral payout.

JavaScript

const ZERO\_ADDRESS = "0x0000000000000000000000000000000000000000"

const owner = "0xcc5d905b9c2c8c9329eb4e25dc086369d6c7777c"

const args = [{

actionType: ActionType.SettleVault,

owner: owner,

secondAddress: owner,

asset: ZERO\_ADDRESS,

vaultId: 1, // settle the first vault

amount: 0,

index: 0,

data: ZERO\_ADDRESS,

}]

await controller.operate(args)

### Redeem

Redeem expired oTokens for collateral. All oTokens are auto-exercised at expiry, and any oToken holder can use this redeem function to claim the payout if the oToken expires in the money.

#### Required Fields:

* asset: the oToken to redeem.
* secondAddress: the address to receive collateral payout.
* amount: amount of oTokens to redeem

JavaScript

const ZERO\_ADDRESS = "0x0000000000000000000000000000000000000000"

const ETHUSD250Put = "0xde99ea535749f02da84d13e6f8253291e32d3a7f"

const holder = "0xcc5d905b9c2c8c9329eb4e25dc086369d6c7777c"

const args = [{

actionType: ActionType.Redeem,

owner: ZERO\_ADDRESS,

secondAddress: holder,

asset: ZERO\_ADDRESS,

vaultId: 0,

amount: 1e8, // redeem 1 oToken

index: 0,

data: ZERO\_ADDRESS,

}]

await ERC20(ETHUSD250Put).approve(marginPool.address, 1\*8)

await controller.operate(args)

### Liquidate

Liquidate a vault in the danger zone by bringing oTokens (either by buying or minting) and redeeming collateral. This is a great use case for [flash-minting](https://opyn.gitbook.io/opyn/#what-is-a-flash-mint). You can view this blog for a [high level overview](https://medium.com/opyn/partially-collateralized-options-now-in-defi-b9d223eb3f4d) of liquidations and this doc for [more specifics](https://www.notion.so/opynopyn/Gamma-Protocol-Liquidations-1ffd204e403245199a433b98c5cc613b).

To find out if a vault is in the danger zone, you can call isLiquidatable() in the [Controller](https://opyn.gitbook.io/opyn/getting-started/abis-smart-contract-addresses), passing in the vault owner, vault id, and the pricer round id.

* If the vault is not liquidatable it will return false, 0, 0
* If the vault is liquidatable it will return true, amount of USDC to repay 1 oToken, USDC dust amount
  + The dust amount is the minimum amount that must remain

You can find the pricer roundId by calling latestRoundData() from the [Chainlink price feed](https://docs.chain.link/docs/get-the-latest-price/).

#### Required Fields:

* owner: the address of the vault owner to liquidate
* secondAddress: the liquidator address
* vaultId: the vault to liquidate
* amount: the amount of oToken to pay back
* data: the roundId to liquidate with, encoded

JavaScript

const ZERO\_ADDRESS = "0x0000000000000000000000000000000000000000"

const vaultOwner = "0xcc5d905b9c2c8c9329eb4e25dc086369d6c7777c"

const liquidator = "0xC257274276a4E539741Ca11b590B9447B26A8051"

const roundId = "0x0000000000000000000000000000000000000000000000000000000000000003"

const args = [{

actionType: ActionType.Liquidate,

owner: vaultOwner,

secondAddress: liquidator,

asset: ZERO\_ADDRESS,

vaultId: 0, // liquidate vault 0

amount: 1e8, // liquidate 1 oToken

index: 0,

data: roundId,

}]

await controller.operate(args)

### Call

Call an arbitrary contract. The protocol is currently under **restricted mode**, where we maintain a whitelist of Callee addresses that can receive calls from the Controller during an operate function call. In the future, anyone can create arbitrary Callee contracts and call it with other actions in one transaction.

#### Required fields:

* secondAddress: callee receiver contract.

#### Optional fields

* owner : the vault owner.
* vaultId: the vault id to mint from
* amount: amount of ETH to attach to the call
* data: other arbitrary data to call the contract with

**Example using** [**PermitCallee**](https://github.com/opynfinance/GammaProtocol/blob/master/contracts/external/callees/PermitCallee.sol)​

* The action arguments will differ based on any given callee's requirements

JavaScript

const ZERO\_ADDRESS = "0x0000000000000000000000000000000000000000"

const owner = "0xcc5d905b9c2c8c9329eb4e25dc086369d6c7777c"

const signedMessage = await signERC2612Permit(wallet.provider, tokenId, owner, spenderAddress.toString(), value, maxDeadline, nonce);

const data = ethers.utils.defaultAbiCoder.encode(

['address', 'address', 'address', 'uint256', 'uint256', 'uint8', 'bytes32', 'bytes32'],

[tokenId, owner, spenderAddress, value.toString(), maxDeadline, signedMessage.v, signedMessage.r, signedMessage.s],

)

​

const args = [{

actionType: ActionType.Call,

owner: owner,

secondAddress: ZERO\_ADDRESS,

asset: ZERO\_ADDRESS,

vaultId: 0,

amount: 0,

index: 0,

data: data,

}]

await controller.operate(args)

## Combining Actions

The [operate](https://github.com/opynfinance/GammaProtocol/blob/386386bac50e24816931190a243e1f220d043c29/contracts/core/Controller.sol#L411) function in Controller contract takes in an array of actions. This allows you to combine actions! We'll outline some common combinations here.

#### Constraints

1. 1.

In the first version, we restrict all the actions in a single operate call to have the same owner and vaultId field if it's an action on a vault, except SettleVault. This mean you cannot create multiple vaults nor deposit assets into multiple vaults in a single transaction.

1. 2.

Also as mentioned in the Call section, we will start the protocol under restricted mode that only whitelisted callee contracts can be called during an operate call.

### Create Short Position

Combines the [OpenVault](https://opyn.gitbook.io/opyn/get-started/actions#openvault), [DepositCollateral](https://opyn.gitbook.io/opyn/get-started/actions#depositcollateral), and [MintShortOption](https://opyn.gitbook.io/opyn/get-started/actions#mintshortoption) actions to create a short position by opening a vault, supplying collateral and minting options from a vault.

JavaScript

const ZERO\_ADDRESS = "0x0000000000000000000000000000000000000000"

const USDC = "0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48"

const owner = "0xcc5d905b9c2c8c9329eb4e25dc086369d6c7777c"

const ETHUSD250Put = "0xde99ea535749f02da84d13e6f8253291e32d3a7f"

const args = [

//Open Vault

{

actionType: ActionType.OpenVault,

owner: "0xcc5d905b9c2c8c9329eb4e25dc086369d6c7777c",

secondAddress: ZERO\_ADDRESS,

asset: ZERO\_ADDRESS,

vaultId: 1, // open the first vault

amount: 0,

index: 0,

data: ZERO\_ADDRESS,

},

//Deposit Collateral

{

actionType: ActionType.DepositCollateral,

owner: owner,

secondAddress: owner,

asset: USDC,

vaultId: 1, // deposit to the first vault

amount: 100 \* 1e6, // deposit 100 USDC

index: 0,

data: ZERO\_ADDRESS,

},

//Mint oTokens

{

actionType: ActionType.MintShortOption,

owner: owner,

secondAddress: owner,

asset: ETHUSD250Put,

vaultId: 1, // mint from the first vault

amount: 100 \* 1e8, // mint 100 oToken

index: 0,

data: ZERO\_ADDRESS,

}

]

//approve collateral to margin pool

await ERC20(USDC).approve(marginPool.address, 100 \* 1\*6)

await controller.operate(args)

### Create Spread

Combines [DepositLongOption](https://opyn.gitbook.io/opyn/get-started/actions#depositlongoption), [DepositCollateral](https://opyn.gitbook.io/opyn/get-started/actions#depositcollateral) (if needed), and [MintShortOption](https://opyn.gitbook.io/opyn/get-started/actions#mintshortoption) to create a spread.

You can learn more about spreads [here](https://opyn.gitbook.io/opyn/#what-is-a-spread). For debit spreads you would not need to deposit collateral beyond the long option, but for credit spreads you would. Below we show a credit spread.

JavaScript

const ZERO\_ADDRESS = "0x0000000000000000000000000000000000000000"

const USDC = "0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48"

const owner = "0xcc5d905b9c2c8c9329eb4e25dc086369d6c7777c"

const ETHUSD250Put = "0xde99ea535749f02da84d13e6f8253291e32d3a7f"

const args = [

//Open Vault

{

actionType: ActionType.OpenVault,

owner: "0xcc5d905b9c2c8c9329eb4e25dc086369d6c7777c",

secondAddress: ZERO\_ADDRESS,

asset: ZERO\_ADDRESS,

vaultId: 1, // open the first vault

amount: 0,

index: 0,

data: ZERO\_ADDRESS,

},

//Deposit Long Option

{

actionType: ActionType.DepositLongOption,

owner: owner,

secondAddress: owner,

asset: ETHUSD200Put,

vaultId: 1, // deposit some option to the first vault

amount: 100 \* 1e8, // deposit 100 oToken

index: 0,

data: ZERO\_ADDRESS,

},

//Deposit Collateral

{

actionType: ActionType.DepositCollateral,

owner: owner,

secondAddress: owner,

asset: USDC,

vaultId: 1, // deposit to the first vault

amount: 100 \* 1e6, // deposit 100 USDC

index: 0,

data: ZERO\_ADDRESS,

},

//Mint oTokens

{

actionType: ActionType.MintShortOption,

owner: owner,

secondAddress: owner,

asset: ETHUSD250Put,

vaultId: 1, // mint from the first vault

amount: 100 \* 1e8, // mint 100 oToken

index: 0,

data: ZERO\_ADDRESS,

}

]

// approve oToken collateral to margin pool

await ERC20(ETHUSD200Put).approve(marginPool.address, 100 \* 1\*8)

//approve USDC collateral to margin pool

await ERC20(USDC).approve(marginPool.address, 100 \* 1\*6)

await controller.operate(args)

### Close Position

In order to close a position, you must buy back oTokens from an exchange if you previously sold your oTokens ([see Trading oTokens](https://opyn.gitbook.io/opyn/get-started/trading-otokens)), and then you can combine the [BurnShortOption](https://opyn.gitbook.io/opyn/get-started/actions#burnshortoption) and [WithdrawCollateral](https://opyn.gitbook.io/opyn/get-started/actions#withdrawcollateral) actions to burn those oTokens and redeem collateral.

If you are trying to close out a spread, you will also need to call [WithdrawLongOption](https://opyn.gitbook.io/opyn/get-started/actions#withdrawlongoption) to redeem your long option after you've burned oTokens.

JavaScript

const ZERO\_ADDRESS = "0x0000000000000000000000000000000000000000"

const ETHUSD250Put = "0xde99ea535749f02da84d13e6f8253291e32d3a7f"

const owner = "0xcc5d905b9c2c8c9329eb4e25dc086369d6c7777c"

const args = [

//Burn Short Option

{

actionType: ActionType.BurnShortOption,

owner: owner,

secondAddress: owner,

asset: ETHUSD250Put,

vaultId: 1, // burn from the first vault

amount: 100 \* 1e8, // burn 100 oToken

index: 0,

data: ZERO\_ADDRESS,

},

//Withdraw collateral

{

actionType: ActionType.WithdrawCollateral,

owner: owner,

secondAddress: owner, // withdraw to the owner address

asset: USDC,

vaultId: 1,

amount: 100 \* 1e6, // withdraw 100 USDC

index: 0,

data: ZERO\_ADDRESS,

}

]

await controller.operate(args)

### Permit, Deposit, and Mint

This is an example of combining the [Call](https://opyn.gitbook.io/opyn/get-started/actions#call) action with other actions, in this case the common [DepositCollateral](https://opyn.gitbook.io/opyn/get-started/actions#depositcollateral) and [MintShortOption](https://opyn.gitbook.io/opyn/get-started/actions#mintshortoption) actions required to create a short position.

The call action facilitates arbitrary calls to whitelisted callees. This uses the [PermitCallee](https://github.com/opynfinance/GammaProtocol/blob/master/contracts/external/callees/PermitCallee.sol) to [let users sign transactions](https://eips.ethereum.org/EIPS/eip-2612) to allow a contract to spend their ERC20 funds vs. having to send an approve transaction.

JavaScript

const ZERO\_ADDRESS = "0x0000000000000000000000000000000000000000"

const USDC = "0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48"

const owner = "0xcc5d905b9c2c8c9329eb4e25dc086369d6c7777c"

const ETHUSD250Put = "0xde99ea535749f02da84d13e6f8253291e32d3a7f"

const signedMessage = await signERC2612Permit(wallet.provider, tokenId, owner, spenderAddress.toString(), value, maxDeadline, nonce);

const data = ethers.utils.defaultAbiCoder.encode(

['address', 'address', 'address', 'uint256', 'uint256', 'uint8', 'bytes32', 'bytes32'],

[tokenId, owner, spenderAddress, value.toString(), maxDeadline, signedMessage.v, signedMessage.r, signedMessage.s],

)

const args = [

//Permit collateral to margin pool

{

actionType: ActionType.Call,

owner: owner,

secondAddress: ZERO\_ADDRESS,

asset: ZERO\_ADDRESS,

vaultId: 0,

amount: 0,

index: 0,

data: data,

},

//Deposit Collateral

{

actionType: ActionType.DepositCollateral,

owner: owner,

secondAddress: owner,

asset: USDC,

vaultId: 1, // deposit to the first vault

amount: 100 \* 1e6, // deposit 100 USDC

index: 0,

data: ZERO\_ADDRESS,

},

//Mint oTokens

{

actionType: ActionType.MintShortOption,

owner: owner,

secondAddress: owner,

asset: ETHUSD250Put,

vaultId: 1, // mint from the first vault

amount: 100 \* 1e8, // mint 100 oToken

index: 0,

data: ZERO\_ADDRESS,

}

]

await controller.operate(args)

**Trading oTokens**

Opyn options (oTokens) are ERC20s, so they can be traded on any decentralized exchange that follows the ERC20 standard.

oTokens on [opyn.co](http://opyn.co/) currently trade on [0x](https://0x.org/), but [v1.opyn.co](https://v1.opyn.co/#/) oTokens traded on [Uniswap](http://uniswap.org/), and projects integrating with Opyn have traded oTokens on [Balancer](https://twitter.com/aaveaave/status/1286371566708826112?lang=en) and [Airswap](https://trader.airswap.io/).

To learn how to take other protocol actions eg. add collateral, mint oTokens, settlement after expiry, please see [Taking Actions](https://opyn.gitbook.io/opyn/get-started/actions).

**Trade on 0x**

oTokens on [opyn.co](http://opyn.co/) currently trade on [0x](https://0x.org/). You can view existing bids + asks, and buy / sell oTokens using [0x API](https://0x.org/docs/api).

You can check out a full example implementation using 0x API at [Gamma Portal's open source codebase.](https://github.com/antoncoding/gamma-portal/blob/master/src/utils/0x-utils.ts) ​

**Get bids and asks**

See [full example on Gamma Portal](https://github.com/antoncoding/gamma-portal/blob/master/src/utils/0x-utils.ts)​

TypeScript

/\*\*

\* get oToken:WETH stats (v1) for all options

\* @param {Array<{addr:string, decimals:number}>} options

\* @param {{addr:string, decimals:number}} quoteAsset

\* @return {Promise<Array<

\* >}

\*/

export async function getBasePairAskAndBids(

oTokens: OToken[],

networkId: SupportedNetworks,

): Promise<OTokenOrderBook[]> {

const filteredOTokens = oTokens // await filter0xAvailablePairs(networkId, oTokens);

// 0x has rate limit of 6 request / 10 sec, will need to chuck array into 6 each

const BATCH\_REQUEST = 6

const COOLDOWN = networkId === 1 ? 0.5 : 2

​

const batchOTokens = filteredOTokens.reduce(

(prev: OToken[][], curr) => {

if (prev.length > 0 && prev[prev.length - 1].length >= BATCH\_REQUEST) {

return [...prev, [curr]]

} else {

const copy = [...prev]

copy[copy.length - 1].push(curr)

return copy

}

},

[[]],

)

​

let final: OTokenOrderBook[] = []

​

for (const batch of batchOTokens) {

const [bestAskAndBids] = await Promise.all([

Promise.map(batch, async ({ id: oTokenAddr }: OToken) => {

const { asks, bids } = await getOTokenUSDCOrderBook(networkId, oTokenAddr)

return {

id: oTokenAddr,

asks: asks.filter(record => isValidAsk(record)),

bids: bids.filter(record => isValidBid(record)),

}

}),

sleep(COOLDOWN \* 1000),

])

final = final.concat(bestAskAndBids)

}

​

return final

}

**Buy or sell oTokens**

See [full example on Gamma Portal](https://github.com/antoncoding/gamma-portal/blob/ec2e8c87c31bb8b4d62d8d99c96b3c3cb75f1e5a/src/hooks/use0xExchange.tsx#L20)​

TypeScript

//approve oToken to 0x if selling, USDC to 0x if buying

​

//fill orders

const fillOrders = useCallback(

async (orders: SignedOrder[], amounts: BigNumber[]) => {

if (!orders.length) return toast.error('No Orders selected')

const exchange = new web3.eth.Contract(abi, addresses[networkId].zeroxExchange)

​

const signatures = orders.map(order => order.signature)

​

const gasPrice = getGasPriceForOrders(orders)

const feeInEth = getProtocolFee(orders).toString()

const amountsStr = amounts.map(amount => amount.toString())

​

console.log(`only filling first order la, orders[0]`, orders[0])

​

await exchange.methods

.batchFillLimitOrders(orders, signatures, amountsStr, false)

.send({

from: user,

value: web3.utils.toWei(feeInEth, 'ether'),

gasPrice: web3.utils.toWei(gasPrice.toString(), 'gwei'),

})

.on('transactionHash', notifyCallback)

​

track('fill-order')

},

[networkId, getProtocolFee, getGasPriceForOrders, notifyCallback, toast, user, web3, track],

)

**Limit Orders**

See [full example on Gamma Portal](https://github.com/antoncoding/gamma-portal/blob/ec2e8c87c31bb8b4d62d8d99c96b3c3cb75f1e5a/src/hooks/use0xExchange.tsx#L20)​

TypeScript

//Create limit order

const createOrder = useCallback(

async (

makerToken: string,

takerToken: string,

makerAmount: BigNumber,

takerAmount: BigNumber,

expiry: number,

takerAddress?: string,

) => {

const salt = new BigNumber(Date.now()).integerValue().toString()

const taker = takerAddress ? takerAddress : '0x0000000000000000000000000000000000000000'

const order = new v4orderUtils.LimitOrder({

chainId: networkId,

makerToken,

takerToken,

makerAmount,

takerAmount,

maker: user,

taker,

sender: '0x0000000000000000000000000000000000000000',

expiry: new BigNumber(expiry).integerValue(),

salt,

})

track('create-order')

const signature = await order.getSignatureWithProviderAsync(

web3.currentProvider as any,

v4orderUtils.SignatureType.EIP712,

)

​

return {

...order,

signature,

}

},

[networkId, user, web3, track],

)

//Broadcast limit order

const broadcastOrder = useCallback(

async (order: SignedOrder) => {

const url = `${httpEndpoint}sra/v4/order`

const res = await fetch(url, {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify(order),

})

if (res.status === 200) return toast.success('Order successfully broadcasted')

const jsonRes = await res.json()

if (jsonRes.validationErrors) return toast.error(jsonRes.validationErrors[0].reason)

​

toast.error(JSON.stringify(jsonRes))

},

[httpEndpoint, toast],

)

​

//Cancel limit orders

const cancelOrders = useCallback(

async (orders: SignedOrder[], callback: Function) => {

const exchange = new web3.eth.Contract(abi, addresses[networkId].zeroxExchange)

​

await exchange.methods

.batchCancelLimitOrders(orders)

.send({

from: user,

})

.on('transactionHash', notifyCallback)

track('cancel-order')

callback()

},

[web3, notifyCallback, networkId, track, user],

)

**Additional Notes**

Some other factors to keep in mind when integrating with 0x:

* Since the 0x protocol fee is determined by the gas price, you must set the gas price in the frontend for your users and warn them not to change the gas price. If they change the gas price, it will change the 0x protocol fee and cause the transaction to revert.
* 0x orders typically expire in less than 5 minutes. To prevent users from getting reverts due to orders expiring, we recommend filtering out orders that will expire in one minute or less, and setting the default gas price to fastest.

**Trade on Airswap**

Many projects integrating with Opyn (eg. [Ribbon](https://ribbon.finance/), [Fontis](http://fontis.finance/), [Optional](http://optional.finance/)) have chosen to use Airswap for making OTC oToken trades with market makers.

These projects run options strategies where they buy or sell options in bulk at predetermined times, making OTC trading a better choice.

You can check out an example implementation using Airswap at [Ribbon's open source codebase](https://github.com/ribbon-finance/structured-products/blob/e66d36b8ef22bcfbc00f20bdd7031a6d418f5221/contracts/instruments/RibbonThetaVault.sol#L472).

**Create and Sign Airswap Order**

Use [Airswap utils](https://www.npmjs.com/package/@airswap/utils) to create an order.

JavaScript

import {createOrder, signTypedDataOrder} from '@airswap/utils';

​

const createSignedOrder = async (

) => {

const order = createOrder({

signer: {

wallet: "0x2343...a456", //buyer address

token: ethers.constants.AddressZero, //payment token address eg. ETH

amount: 100000000000000000, //payment amount

},

sender: {

wallet: "0x234...A3", //seller address

token: "0x2343...a456", //oToken address

amount: 100000000000000000, //amount of oTokens to sell

},

affiliate: {

wallet: ethers.constants.AddressZero, //can set a fee here if desired, here it is 0

},

});

const signedOrder = await signTypedDataOrder(order, "0x243...452", "0xw423...234f");

return signedOrder;

};

**Execute Trade**

Use [Airswap's swap function](https://docs.airswap.io/reference/swap#solidity), submitting the order from above

**System Roles & Terms**

**Admin Roles**

|  |  |
| --- | --- |
| **Role** | **Action** |
| Owner | Can whitelist/blacklist collateral and oToken addresses, update/upgrade modules, set addresses for other roles in the system, and manage oracle parameters. |
| Partial Pauser | Can pause all actions other than redeem oToken and settle vault |
| Full Pauser | Can fully pause the system in case of an emergency. |
| Disputer | Price disputer for the oracle contract that can dispute the price submitted by Pricer contracts, and has the privilege to override with a new price. |
| Pricer | Can submit oracle prices |
| Farmer | Can withdraw any token excess balance in the pool. |

**Operators**

Operators allow users to delegate control of their vaults to a third party smart contract or EOA. This could be a smart contract that rolls over their options for them, an EOA fund manager to do trades for them, or a vast possibility of other interactions that developers can build on top of the protocol.

Operators have full control over user funds and can take any action on behalf of a user. The only action operators cannot take is to add new operators or remove new operators. Every account can define multiple operators.

An example use of an existing operator, is the [PayableProxy contract](https://github.com/opynfinance/GammaProtocol/blob/master/contracts/external/proxies/PayableProxyController.sol) which takes ETH from users and converts it to WETH to use in the system.

**Glossary of Terms**

|  |  |
| --- | --- |
| **Name** | **Description** |
| oToken | An ERC20 representation of option token. |
| Strike Price | Strike price with 8 decimals |
| Strike | Asset that the strike price is denominated in |
| Collateral | Asset that is held as collateral against short/written options |
| Underlying | Asset that the option references |
| Product | A unique combination of underlying, strike, collateral and Put/Call. Ex: ETH:USDC:USDC:PUT |
| Put Option | A put option gives the buyer the right but not the obligation to sell the underlying ERC20 at strike price. |
| Call Option | A call option gives the buyer the right but not the obligation to buy the underlying ERC20 at strike price. |
| Expiry | The timestamp that option payout is determined. After the expiry timestamp, the system will enable buyers and sellers to withdraw their expected payout. |
| Vault | An option seller needs to open a vault to mint option tokens. The options minted are saved as “short” otokens in a vault.  A seller can deposit collateral assets into a vault to mint a oToken, or add some long oTokens to reduce the amount of collateral needed for creating the short position. |
| oToken ID | Each oToken has a unique ID which is represented by Hash (underlyingAsset, strikeAsset, collateralAsset, expiry, strikePrice, isPut). |