DutchSwap Documentation

Release latest

Adrian Guerrera and the DutchSwap community

Nov 19, 2020

Getting Started

DutchSwap is a smart contract factory and Dutch auction house for ERC20 tokens.

> Historically, Dutch auctions have been used to provide a fair price discovery event for both new and developed products and markets. For those with something to sell, they provide a system where you leave with all your stock sold - at a price you decide you're happy with. For those looking to buy, a way to make sure you're getting the same deal as everybody else in the market. With these goals in mind, DutchSwap aims to update the traditional Dutch auction experience and bring them to the Ethereum community.

A fair token pricing solution for new and existing ERC20 tokens, DutchSwap has you covered from bulb to bloom.

!! Documentation is currently under construction !!

Note: Parts of this documentation reference smart contracts written in `Solidity` and the documentation assumes a basic familiarity with it. You may wish to view the Solidity docs if you have not used it previously.

CHAPTER 1

Features

- Dutch Auctions for ERC20 tokens
- Payments in ETH or any ERC20 of your choosing
- Token minting and dispersal services
- Contract testing via pytest, including trace-based coverage evaluation
- Property-based and stateful testing via hypothesis

The main documentation for the site is organized into the following sections:

1.1 Quickstart

Note: Visit the DutchSwap Venture Studio as you read and try it out for yourself! ",

This page provides a quick overview of DutchSwap. It assumes a level of familiarity with smart contract development, the Ethereum network and ERC20 tokens and will rely on some simple examples. For more in-depth content, continue to the further information under "Getting Started" in the table of contents. **UNDER CONSTRUCTION**

DutchSwap is a smart contract framework for creating Dutch auctions on Ethereum.

The idea of a Dutch auction is to determine a fair price for a batch of items. In particular, new projects and tokens with no previous price are perfect as they allow the market an even playing field for price discovery. Though there are many positives to using a Dutch auction for a fixed supply of any item, this document will cover the creation and auction of a brand new ERC20 token.

Note: To interact with DutchSwap, you'll first need to connect your Ethereum wallet. DutchSwap support many options for this, but we recommend Metamask.

1.1.1 1 - Creating a New Token

The first step to using DutchSwap is having a token to sell. .. note:

```
   If you already have some ERC20 tokens you wish to sell, continue to step 2.

To mint your own new ERC20 token, you need to do the following:
```

- Enter your Token's name
- Add a symbol associated with your token
- Set a total supply

DutchSwap will mint the tokens for you and deposit them into your wallet. After creating your ERC20 tokens, you're ready to set up your auction.

1.1.2 2 - Starting a Dutch Auction

The process for creating an auction for your token is simple, you only need to follow the next steps:

- Click "New Auction" tab on the left side of the DutchSwap studio. You will see a window indicating the steps required to create an auction.
- Enter the details of the ERC20 Token you're wishing to sell.
- **Set your auction details, including:**
  - Amount - quantity of token supply that you want to sell.
  - Prices - set the starting price high and the end price as low as you're willing to go.
  - Dates - pick appropriative start and end dates for your auction.

Congratulation! You have created your auction and have full access to its data. Change it whatever is needed and manage your token sale with comfort.

Note: You must be able to transfer the number of tokens for sale to the auction contract, otherwise it will fail.

Note: The auction starting price must be greater than the ending price. Dutch auctions start high, and drop in price until sold out.

1.1.3 3 - Claiming your tokens

The dutch auction ends when the sale ends and the final price is greater or equal to the minimum price of the seller, or if the auction period finishes without all the tokens being sold.

At the end of the auction, simply click "Claim"

- If the auction was successful, you will claim the tokens you have won at auction.
- If the auction has ended below the minimum, your claim will be for your original tokens.

1.2 Step by step

1.2.1 Connecting your Wallet

Ethereum Wallets

To buy or sell tokens with Dutch Swap, you'll need an Ethereum wallet. The most commonly supported wallet is Metamask.

Below you can see some examples of how you can use metamask.

Metamask

Download and install Metamask into your browser

Note: You can read more about metamask here: https://metamask.io/

1.3 What is a Dutch Auction

First, a little history lesson of how and why Dutch auctions came to be. Dutch auctions arose in the auction houses, cafes and drinking houses of the Netherlands during the Dutch Golden Age. During this period, the Dutch had established themselves as a world power primarily on the back of a maritime trading empire. Dutch traders and merchants had at times sole access to many of the most coveted materials and wares from throughout the world, with the means and infrastructure to enter trade throughout all of Europe.

This unique situation and the massive expanse of trade, financing and market making it brought with it eventually led to the creation of many of the financial institutions we still rely on today - merchant banking, stock markets and futures contracts for example. Crucial in the area of market price discovery in these new and emerging ventures was the creation of the Dutch Auction - originally a mechanism to ensure sellers could move all of their stock for an amount they needed to continue operation, while allowing individuals in the market to dictate a fair investment they were happy with.

We see the growing Ethereum community facing many of the same requirement in how they set prices for the many varied Ethereum and ERC20 token projects out there. We believe that the market should determine the price of a token rather than you or I, that price setting should happen in the open - not in the dark. The Dutch Auction mechanism achieves this fairness for two main reasons.:

- First, at every auction everyone who buys, buys it at the same price.
- Second, the market determines that price itself.

These are the characteristics that have kept Dutch auctions in use for 400 years and why we think we can provide a lot of value to the Ethereum community. Although they are not yet so popular in our world, they have already proven themselves for centuries in the fields of business and trade in particular.

1.3.1 How a DutchSwap Auction Works

- Every auction has a set amount of tokens available to be bought.
- The price is initially set at a high price and during the auction continues to drop.
- When a buyer bids they are setting their total spend and a maximum individual price for the token.

- As the individual price continues to drop, buyers total spend stays the same the amount of token they receive increases.
- The price drops until either all the tokens are sold at auction, or it hits the minimum price set by the seller.

1.3.2 Why would I need a Dutch auction?

Because this type of auction solves the following problems:

- The fairest method for selling more than one of the same object.
- Descending prices ensure bidders will bid early, and honestly.
- Lowest bid price for all means people commit what they think it is worth, yet are likely to get it cheaper.
- Everything is done out in the open, transparency to everyone involved.

Note: A VERY simple example - V wants to sell 100 tokens. X, Y & Z see value behind this token and wish to take part in a dutch auction of the tokens. Bidding begins at $5 per token, and will drop in price $1 every hour. At $4, X commits to $50 worth of tokens - X sees a deal and doesn't want somebody buying all the tokens up! Nobody else bids however and the auction continues. At $2, Y believes the market will jump in so they bid - $50 worth and the auction ends. Both pay their $50 and receive their tokens. X receives twice as many tokens as they would have at $4, Y receives $50 worth of tokens at a price they're happy with. Z, who waited too long, misses out entirely.

The biggest benefit of such auctions is that they are meant to democratize public offerings. As it happens currently, the process for conducting a typical IPO is controlled well before it reaches an open market.

1.3.3 Recent Dutch Auctions

The research and development behind DutchSwap was inspired by the Google IPO which was a dutch auction by OpenIPO and both the Algorand and Gnosis dutch auctions. More details on the Algorand auction mechanism can be found here: Algo Auction We do not support the Algorand refund vault but will do so in v2.

1.4 Auction Process

To set up your auction you need to proceed through some pretty simple actions. Really, it's fairly easy to start trading using Dutch Swap.

Start from the beginning. When you open the corresponding tab with the creation of an auction, you will be greeted by the first field required to fill in. This field is your token. If one already exists, just start typing its name or symbol in the input field and select it from the drop-down list and continue working.

Note: If no such token exists, create one. The creation process is described in a separate paragraph of the documentation.

1.5 Token Types

Most ERC20 tokens can be sold in a dutch Auction. For those without exisitng tokens, our token factory can mint Base Tokens. Base token type is an ERC20 token from the DutchSwap protocol. It is essentially an ERC20 with some advanced features from the ERC777 standard and some bonus code to make them work with the DutchSwap contract.

Dutch auction sales differ from traditional token sales in that the Dutch auction allows the market to set the final token price. The auction process allows tokens to be allocated in an equal and impartial way as all successful bidders pay the same price per token.

The bidding process generates a clearing price for the tokens offered in the auction. Once the token price is determined, all investors who submitted successful bids receive an allocation of tokens at that price.

1.5.1 What is an ERC20 Token

Tokens themselves is a special unit that describes what exactly you sell. It can hold value and be sent and received.

The price per token that you ultimately pay will be less than or equal to the quoted price at the time since the token price decreases over time.

Demand for tokens is calculated as the sum of payments received from users divided by the current price per token. Once demand equals the total amount of tokens available, the auction ends and the token price is locked in.

If the auction concludes at the end of the auction period everyone will receive their tokens at the pre-defined reserve price, also known as the price floor.

1.5.2 Types of Tokens

Most ERC20 tokens can be sold in a Dutch Auction.

The Basic ERC20 Token implements the missing name, short name, decimals and the initial total supply of a standard ERC20 token. No extra features, no extra bells, and whistles. Just standard ERC20 functionality.

Note: For those without existing tokens, our token factory can mint Base Tokens. The base token type is an ERC20 token from the DutchSwap protocol. It is essentially an ERC20 with some advanced features from the ERC777 standard and some modified code to make them work with the Dutch Swap contract.

Exceptions for DutchSwap

Some exceptions include deflationary tokens where there is a difference in total value when transfered, usually from a transfer at the protocol level.

1.6 Creating Tokens

Most ERC20 tokens can be used as the sale token in a Dutch Auction.

Note: This framework is based on swapping ETH or ERC20 tokens in batches using a Dutch auction mechanism. To fully understand the documentation of the solution outlined, you must have a basic knowledge of what a ERC20 token is and how to use Ethereum

---

The most important step towards recognition was the possibility to create one's own custom Dutch Swap token. That gave an opportunity to dozens of projects to see life thanks to additional flexibility of auctions. Now everyone can create their own token with a unique name and fully manage it.

1.6.1 Initial Deployment

To create your one follow the next steps:

- Visit the appropriative page on Dutch Swap
- Enter the reliable values of your token, such as name, symbol, and decimals
- Click the Mint button to create your custom token

Note: Note that your Ropsten network address will be automatically fetched from your account.

- It starts with entering Token's name - should be the unique value(such as Tether, Chainlink).
- Add a symbol for your token. It should be consist of about 3-5 uppercaseletters

---

Note: After creating your special token, you will be able to use it directly in newly created auctions

---

1.6.2 Adding token to Metamask

You can take the new contract address and add it as a custom token in Metamask

Metamask cant keep track of all the new tokens on Ethereum and will need to be added mamually.

1.7 Smart Contracts

1.7.1 Deployed Smart Contracts

Current deployments for V1.3

Ethereum Mainnet

- Fixed ERC20 Factory:

0xA550114ee3688601006b8b9f25e64732eF774934

- Auction Factory:

0x3CB6Fb749a1FD088e1C524cBA27f25B5FDd105c8

Ropsten Test Network

- Fixed ERC20 Factory:

0xDAD930b252bcd95fA5bfaeEa20420283DbfBc94c

- Auction Factory:

0x46f6FbAac7346E4d0A66200A9267F791412d8884

Rinkeby Test Network

- Fixed ERC20 Factory:

0x1C3e1D406E64004416Fd592C55f9eDeD1A76Bae8

- Auction Factory:

0x30E5620794dDe007f9F071344Ecdd44C959Bb4B6

Kovan Test Network

- Fixed ERC20 Factory:

0x1C3e1D406E64004416Fd592C55f9eDeD1A76Bae8

- Auction Factory:

0x2c2a4b9843eC5377f4BC25797E8B3639Da1d09dD

Goerli Test Network

- Fixed ERC20 Factory:

0x1C3e1D406E64004416Fd592C55f9eDeD1A76Bae8

- Auction Factory:

0xC996D2F04DDF8c44A2AbE77262ec78B7BF21203B

1.7.2 Dutch Auction Contract

First we deploy the Dutch Auction Contract to the blockchain. And we use it as a template to supply it to Dutch Auction Factory

The factory contract is explained in *DutchSwap Factory Contract*

The Dutch Auction is deployed using the function *deployDutchAuction* of the factory.

The function *deployDutchAuction* initializes the contract using the function initDutchAuction

initDutchAuction

This initialises the smart contract:

```
function initDutchAuction(
    address _funder,
    address _token,
    uint256 _totalTokens,
    uint256 _startDate,
    uint256 _endDate,
    address _paymentCurrency,
    uint256 _startPrice,
    uint256 _minimumPrice,
```

(continues on next page)

(continued from previous page)

```
    address payable _wallet
)
```

Here,

1._funder: This is the address of the factory that creates the Dutch Auction. The factory needs to be given approval by the funder(msg.sender) to spend the tokens on the funders behalf

2._token: The address of the token

3._totalTokens: The amount of supply of the tokens. It is in wei (ie totalSupply * 10**18)

4._startDate: The start date for the auction

5._endDate: The end date for the auction

6._paymentCurrency: Address of the currency you want to be paid with. Can be ethereum address(0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE) or a token address

7._startPrice: Start Price for the token to start auction(in wei). This should be the maximum price you want your token to be valued at

8._minimumPrice: Minimum price you want the token to be valued at.

9._wallet: The address that you want your payment to be received at if the auction is successfuly. It is also the address that you will receive your tokens at if the auction is not successful.

Get Token Price:

- *function priceFunction()*

Returns price during the auction which proportionaly decreases as the time elapses.

- *function clearingPrice()*

The current clearing price of the Dutch auction. If the auction is successful, it returns the actual token price.

- *function tokenPrice()*

The average price of each token. It is derived by dividing total commitments by total tokens. The total commitment is the amount of tokens or ethers commited to the contract

Get Tokens available

- *function tokensClaimable(address _user)*

The number of tokens the user is able to claim. It is given by the total commitment of the user divided by the price of token(clearing price)

- *function tokensRemaining()*

Total amount of tokens left for auction

- *function totalTokensCommitted()*

Total amount of tokens committed at current auction price

How to Buy a token

- *function commitEth() public payable*

Commit ETH to buy tokens for any address. It calculates the commitment based on the amount of ether we give to the smart contract. If we commit ethers that is greater than the maximum commitment available this function will refund

Finally it will add how much a user has commited to commitments mapping

- *function commitTokens(uint256 _amount)*

Users must approve contract prior to committing tokens to auction. It calculates the commitment based on the amount of token we want to buy with.

Determine the commitments

- *function calculateCommitment(uint256 _commitment)*

Returns the amount able to be committed during an auction. If the commitment is bigger than the maximum commitment it will subtract the surplus.

Getters

- *function auctionSuccessful() public view returns (bool)*

Returns successful if the tokens sold equals total Tokens. That is to say the token price is greater than or equal to the clearing price.

- *function auctionEnded() public view returns (bool)*

Returns bool if auction is successful or time has ended

Finalizing the auction

- *function finaliseAuction () public*

If the auction has successfuly finished above the reserve, then transfer the total commitments to the initialized wallet

If the function has cancelled or failed, transfer total tokens back, ie to initialized wallet

- *function withdrawTokens()*

If the auction has successfuly finished, transfer the claimed tokens to the bidders.

If the auction did not meet the reserved price, return the commited funds back to bidders.

1.7.3 DutchSwap Factory Contract

Functions

- *function initDutchSwapFactory(address _dutchAuctionTemplate, uint256 _minimumFee)*

This function initializes the factory method with the Dutch Auction contract that has been deployed. You can deploy the DutchAuctionContract or use the already deployed address and pass it to _dutchAuctionTemplate

- *function addCustomAuction(address _auction)*

You can add a Dutch Swap Auction you have created without using the *function deployDutchAuction*

- *function removeFinalisedAuction(address _auction)*

Remove the function that has ended or finalised

- **function deployDutchAuction(** address _token, uint256 _tokenSupply, uint256 _startDate, uint256 _endDate, address _paymentCurrency, uint256 _startPrice, uint256 _minimumPrice, address payable _wallet)

This function creates dutch auction and approves the created dutch auction to use the supplied auction_token for the auction.

The parameters to pass are as follows:

1._token: This is the address of ERC20 Token we just created

2._tokenSupply:The supply of total number of tokens for the auction(uint256). This must be in wei(ie totalSupply * 10**18)

3._startDate: The start date for the auction(uint)

4._endDate: The end date for the auction(uint)]

5._paymentCurrency: Address of the currency you want to be paid with. Can be ethereum address(0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE) or a token address

6._startPrice: Start Price for the token to start auction(in wei). This should be the maximum price you want your token to be valued at

7._minimumPrice: Minimum price you want the token to be valued at.

8._wallet: The address that you want your payment to be received at if the auction is successfuly. It is also the address that you will receive your tokens at if the auction is not successful.

1.7.4 Fixed Supply ERC20 Contract

FixedSupplyToken

The FixedSupplyToken smart contract implements all the mandatory ERC20 functions. They are:

totalSupply(), balanceOf(...), transfer(...), transferFrom(...), approve(...) and allowance(...)

Additionally, the approveAndCall(...) functionality is included so that the two operations of executing tokenContract.approve(targetContract, tokens) and targetContract.doSomething(...) (which will execute tokenContract.transferFrom(user, targetContact, tokens)) can be combined into a single approveAndCall(...) transaction. Please only use this functionality with trusted smart contracts, and with checks!

Factory deployTokenContract Function

- *function deployTokenContract(string memory symbol, string memory name, uint8 decimals, uint totalSupply) public payable returns (address token)*

Deploy a new token contract. The account executing this function will be assigned as the owner of the new token contract. The entire totalSupply is minted for the token contract owner.

Parameters:

- symbol: Symbol of the token
- name: Token contract name
- decimals: Decimal places, between 0 and 27. Commonly 18
- totalSupply: The number of tokens that will be minted to the token contract owner's account

1.8 Deployment

1.8.1 Deployed Smart Contracts

Current deployments for V1.3

Ethereum Mainnet

- Fixed ERC20 Factory:

0xA550114ee3688601006b8b9f25e64732eF774934

- Auction Factory:

0x3CB6Fb749a1FD088e1C524cBA27f25B5FDd105c8

Ropsten Test Network

- Fixed ERC20 Factory:

0xDAD930b252bcd95fA5bfaeEa20420283DbfBc94c

- Auction Factory:

0x46f6FbAac7346E4d0A66200A9267F791412d8884

Rinkeby Test Network

- Fixed ERC20 Factory:

0x1C3e1D406E64004416Fd592C55f9eDeD1A76Bae8

- Auction Factory:

0x30E5620794dDe007f9F071344Ecdd44C959Bb4B6

Kovan Test Network

- Fixed ERC20 Factory:

0x1C3e1D406E64004416Fd592C55f9eDeD1A76Bae8

- Auction Factory:

0x2c2a4b9843eC5377f4BC25797E8B3639Da1d09dD

Goerli Test Network

- Fixed ERC20 Factory:

0x1C3e1D406E64004416Fd592C55f9eDeD1A76Bae8

- Auction Factory:

0xC996D2F04DDF8c44A2AbE77262ec78B7BF21203B

1.8.2 Dutchswap Factory Deployment

Environment

Local Environment Setup

This needs to set up with the following requirements:

- Install brownie
- Install Ganache CLI

Scripts

We have already deployed Dutchswap Factory Contract for the respective testnet and also in mainnet

The links to addresses are Deployed Smart Contract page.

If you want to deploy your own Factory Contracts please take these steps:

Deploy ERC20 Token Factory

- We have a token factory smart contract **BokkyPooBahsFixedSupplyTokenFactory**. First lets deploy it:

```
token_factory = BokkyPooBahsFixedSupplyTokenFactory.deploy({'from': accounts[0]})
```

Deploy Dutch Auction Factory:

- First we create a template for dutch auction by deploying DutchSwapAuction smart contract:

```
dutch_auction_template = DutchSwapAuction.deploy({'from': accounts[0]})
```

- We deploy the DutchSwapFactory contract to create a new Dutch Swap Auction:

```
auction_factory = DutchSwapFactory.deploy({"from": accounts[0]})
```

- We initialize our DutchSwapFactory with dutch_auction_template we created:

```
auction_factory.initDutchSwapFactory(dutch_auction_template, 0, {"from":␣
↪accounts[0]})
```

Okay, so we have created a token factory to supply to our Auction and a Auction Factory to create an auction. Please follow this link for further deployment:

DutchSwap Deployment

1.8.3 DutchSwap Deployment

Environment

Local Environment Setup

This needs to set up with the following requirements:

- Install brownie

- Install Ganache CLI

Script

We have already deployed Dutchswap Factory Contract for the respective testnet and also in mainnet

If you want to deploy the factories please deploy Auction factory using ref:*deployment_factory*

The links to addresses are *Deployed Smart Contracts*

Create ERC20 Token:

Only follow the Create ERC20 Token below Steps if you dont have an ERC20Token

- First we need a ERC20 token. We create a token using ERC20 Token Factory smart contract **BokkyPooBahsFixedSupplyTokenFactory**. The address of token factory is found in Deployed Smart Contract page
- Copy the Fixed ERC20 Factory address of the required network and create a token factory using:

```
token_factory = BokkyPooBahsFixedSupplyTokenFactory.at(token_factory_address)
```

- Create a transaction to deploy ERC20Token

```
tx = token_factory.deployTokenContract(SYMBOL,NAME,DECIMALS,NUMBER_OF_AUCTION_
↪TOKENS,{'from': accounts[0], "value": "@value ethers"})
```

Deploy a new token contract. The account executing this function will be assigned as the owner of the new token contract. The entire totalSupply is minted for the token contract owner.

This transaction will create a ERC20 Fixed Supply ERC20 Token with the properties you pass for the values of the parameters The parameters are:

1. SYMBOL: The symbol representing the token
2. NAME: The name of the token created
3. DECIMALS:In ERC20 tokens, that scaling factor is denoted by the value of decimals , which indicates how many 0's there are to the right of the decimal point the fixed-point representation of a token
4. NUMBER_OF_AUCTION_TOKENS: The number of tokens that will be minted to the token contract owner's account
5. @value: The value in ether of the total supplied tokens. This must be atleast the minimumFee(0.1 ethers).

- We need the token to be able to use it. How do we get it? Simple just pass the address of the token we get from above transaction:

```
auction_token = FixedSupplyToken.at(web3.toChecksumAddress(tx.events[
↪'TokenDeployed']['token']))
```

- Okay so we have created a token for which we want to auction. Lets create a auction!

Create Dutch auction

- First we need a Auction Factory which actually creates an Auction for the specified ERC20 Token. We have already deployed DutchSwapFactory at respective addresses found in:

  Deployed Smart Contracts

- Copy the Auction Factory address of the required network and create a Auction Factory using:

```
auction_factory = DutchSwapFactory.at(auction_factory_address)
```

- Before creating an auction we need to approve the Auction Factory to be able to transfer the tokens:

```
auction_token.approve(auction_factory,AUCTION_TOKENS, {"from": accounts[0]})
```

- Now lets create a dutch auction by deploying it using the factory:

```
tx = auction_factory.deployDutchAuction(auction_token, AUCTION_TOKENS, AUCTION_
↪START,AUCTION_END,PAYMENT_CURRENCY, AUCTION_START_PRICE, AUCTION_RESERVE,␣
↪wallet, {"from": accounts[0]})
```

This function creates dutch auction and approves the created dutch auction to use the supplied auction_token for the auction.

The parameters to pass are as follows:

1.auction_token: This is the address of ERC20 Token we just created

2.AUCTION_TOKENS:The supply of total number of tokens for the auction(uint256). This must be in wei(ie totalSupply * 10**18)

3.AUCTION_START: The start date for the auction(uint)

4.AUCTION_END: The end date for the auction(uint)]

5.PAYMENT_CURRENCY: Address of the currency you want to be paid with. Can be ethereum address(0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE) or a token address

6.AUCTION_START_PRICE: Start Price for the token to start auction(in wei). This should be the maximum price you want your token to be valued at

7.AUCTION_RESERVE: Minimum price you want the token to be valued at.

8.wallet: The address that you want your payment to be received at if the auction is successfuly. It is also the address that you will receive your tokens at if the auction is not successful.

- Finally we need the actual address where the auction has been deployed. This is given by:

```
dutch_auction = DutchSwapAuction.at(web3.toChecksumAddress(tx.events[
↪'DutchAuctionDeployed']['addr']))
```

deploy_DutchSwapAuction.py

Okay so all the script mentioned above are put into a deployment script in the file deploy_DutchSwapAuction.py

Please check the code and supply the parameters as per your requirements

All you need to do is run the command:

brownie run deploy_DutchSwapAuction.py

The link for the **deploy_DutchSwapAuction.py**:

deploy_DutchSwapAuction

For local setup in your ganacheCLI you need to modify it a little:

In line 27 of deploy_DutchSwapAuction.py change *USE_EXISTING_FACTORY* to False

1.9 Ways to contribute

Dutch Swap is a community-driven and open source project. There are many ways to contribute to the DutchSwap project, it's as much about the code as it is the community. Here are some way to get involved regardless of your skill set:

- **Be part of the community.** The best way to contribute to DutchSwap and help us grow is simply to use the auctions and promote the experience by word-of-mouth, blog posts, tutorials, videos, test contracts, support on the forums, Discord. You get the point. Being a user and advocate helps spread the word about DutchSwap, which has no marketing budget and relies on its community for adoption.
- **Make auctions.** It's our main focus to convince new projects and especially the industry at large that DutchSwap and Dutch auctions are a relevant option - We need auctions to be made with Dutch Swap. We know that the framework has a lot of potential and just needs usage to draw attention to Dutch Swap. So keep working on your awesome projects, and if you find a project looking for a better method of price discovery for their token - send them our way!
- **Donate.** Dutch Swap is an open sourced project, but it can still benefit from user donations for many things. Most importantly, we use donation money to hire core developers so they can work full-time on the protocol. Full-tme development has been very beneficial to the project so far and ensures a quality product on first offering. If you want to donate some tokens to the project, check our website for details.
- **Get involved in the development.** Developers of all levels can help improve DutchSwap. This can be by contributing code via pull requests, testing the contracts directly on chain or on the git *master* branch, reporting bugs or suggesting enhancements on the issue tracker, improve the official documentation (both the class reference and tutorials) and its translations. Our development team would love to have you on board. The following sections will cover each of those "direct" ways of contributing.

1.9.1 Contributing code

The possibility to study, use, modify and redistribute modifications of the dutch swaps's source code are the fundamental rights that Dutch Swap's MIT license grants you, making it free and open source software.

As such, everyone is entitled to modify Dutch Swap's source code, and send those modifications back to the upstream project in the form of a patch (a text file describing the changes in a ready-to-apply manner) or - in the modern workflow that we use - via a so-called "pull request" (PR), i.e. a proposal to directly merge one or more Git commits (patches) into the main development branch.

Contributing code changes upstream has two big advantages:

- Your own code will be reviewed and improved by other developers, and will be further maintained directly in the upstream project, so you won't have to reapply your own changes every time you move to a newer version. On the other hand it comes with a responsibility, as your changes have to be generic enough to be beneficial to all users, and not just your project; so in some cases it might still be relevant to keep your changes only for your own project, if they are too specific.
- The whole community will benefit from your work, and other contributors will behave the same way, contributing code that will be beneficial to you. At the time of this writing, more than 1000 developers have contributed code changes to the engine!

To ensure good collaboration and overall quality, the Dutch Swap developers enforce some rules for code contributions, for example regarding the style to use in the C++ code (indentation, brackets, etc.) or the Git and PR workflow.

A good place to start is by searching for issues tagged as junior jobs (or Hacktoberfest during October) on GitHub.

See also:

Technical details about the PR workflow are outlined in a specific section, doc_pr_workflow.

Details about the code style guidelines and the `clang-format` tool used to enforce them are outlined in doc_code_style_guidelines.

1.9.2 Testing and reporting issues

Another great way of contributing to the engine is to test development releases or the development branch and to report issues. It is also helpful to report issues discovered in stable releases, so that they can be fixed in the development branch and in future maintenance releases.

Testing development versions

To help with the testing, you have several possibilities:

- Compile the engine from source yourself, following the instructions of the Compiling page for your platform.
- Test official pre-release binaries when they are announced (usually on the blog and other community platforms), such as alpha, beta and release candidate (RC) builds.
- Test "trusted" unofficial builds of the development branch; just ask community members for reliable providers. Whenever possible, it's best to use official binaries or to compile yourself though, to be sure about the provenance of your binaries.

As mentioned previously, it is also helpful to keep your eyes peeled for potential bugs that might still be present in the stable releases, especially when using some niche features of the engine which might get less testing by the developers.

Filing an issue on GitHub

Dutch Swap uses GitHub's issue tracker for bug reports and enhancement suggestions. You will need a GitHub account to be able to open a new issue there, and click on the "New issue" button.

When you report a bug, you should keep in mind that the process is similar to an appointment with your doctor. You noticed *symptoms* that make you think that something might be wrong (the engine crashes, some features don't work as expected, etc.). It's the role of the bug triaging team and the developers to then help make the diagnosis of the issue you met, so that the actual cause of the bug can be identified and addressed.

You should therefore always ask yourself: what is relevant information to give so that other Dutch Swap contributors can understand the bug, identify it and hopefully fix it. Here are some of the most important infos that you should always provide:

- **Operating system.** Sometimes bugs are system-specific, i.e. they happen only on Windows, or only on Linux, etc. That's particularly relevant for all bugs related to OS interfaces, such as file management, input, window management, audio, etc.
- **Hardware.** Sometimes bugs are hardware-specific, i.e. they happen only on certain processors, graphic cards, etc. If you are able to, it can be helpful to include information on your hardware.
- **Dutch Swap version.** This is a must have. Some issues might be relevant in the current stable release, but fixed in the development branch, or the other way around. You might also be using an obsolete version of Dutch Swap and experiencing a known issue fixed in a later version, so knowing this from the start helps to speed up the diagnosis.
- **How to reproduce the bug.** In the majority of cases, bugs are reproducible, i.e. it is possible to trigger them reliably by following some steps. Please always describe those steps as clearly as possible, so that everyone can try to reproduce the issue and confirm it. Ideally, make a demo project that reproduces this issue out of the box, zip it and attach it to the issue (you can do this by drag and drop). Even if you think that the issue is trivial to reproduce, adding a minimal project that lets reproduce it is a big added value. You have to keep in mind that there are thousands of issues in the tracker, and developers can only dedicate little time to each issue.

When you click the "New issue" button, you should be presented with a text area prefilled with our issue template. Please try to follow it so that all issues are consistent and provide the required information.

1.9.3 Contributing to the documentation

There are two separate resources referred to as "documentation" in Dutch Swap:

- **The class reference.** This is the documentation for the complete Dutch Swap API as exposed to GDScript and the other scripting languages. It can be consulted offline, directly in Dutch Swap's code editor, or online at Dutch Swap API. To contribute to the class reference, you have to edit the *doc/base/classes.xml* in Dutch Swap's Git repository, and make a pull request. See doc_updating_the_class_reference for more details.
- **The tutorials and engine documentation and its translations.** This is the part you are reading now, which is distributed in the HTML, PDF and EPUB formats. Its contents are generated from plain text files in the reStructured Text (rst) format, to which you can contribute via pull requests on the godot-docs GitHub repository. See doc_documentation_guidelines for more details.

1.9.4 Contributing translations

To make Dutch Swap accessible to everyone, including users who may prefer resources in their native language instead of English, our community helps translate both the Dutch Swap editor and its documentation in many languages.

See doc_editor_and_docs_localization for more details.

1.10 Community channels

So, where is the Dutch Swap community and where can you ask questions and get help?

A brief overview over these and other channels is also available on the Dutch Swap website.

1.10.1 Discord

Come join our server, we like to do regular break out sessions, and you'll get to meet some of the team.

- Discord

1.10.2 Github

The following is the repos we'll be working from. Pull requests welcomed!

- GitHub

1.10.3 Twitter

Any public communitation - announcements, offers and important information will likely come through the Twitter. Follow us to stay up-to-date!

- Twitter

Note that some of these channels are run and moderated by members of the Dutch Swap community or third parties.