Lens Protocol¶

The Lens Protocol is a decentralized, non-custodial social graph. Lens implements unique, on-chain social interaction mechanisms analogous to commonly understood Web2 social media interactions, but significantly expanded with unique functionality that empower communities to form and participants to own their own social graph.

Setup¶

For now only Linux and macOS are known to work

We are now figuring out what works for Windows, instructions will be updated soon

(feel free to experiment and submit PR's) The environment is built using Docker Compose, note that your.env file must have the RPC URL of the network you want to use, and an optionalMNEMONIC andBLOCK_EXPLORER_KEY , defined like so, assuming you choose to use Mumbai network:

MNEMONIC="MNEMONIC YOU WANT TO DERIVE WALLETS FROM HERE" MUMBAI_RPC_URL="YOUR RPC URL HERE" BLOCK_EXPLORER_KEY="YOUR BLOCK EXPLORER API KEY HERE" With the environment file set up, you can move on to using Docker:

export

USERID

UID

&&

docker-compose build &&

docker-compose run --name lens contracts-env bash If you need additional terminals:

docker exec

-it lens bash From there, have fun!

Here are a few self-explanatory scripts:

npm run test npm run coverage npm run compile Cleanup leftover Docker containers:

USERID

UID

docker-compose down

Protocol Overview¶

The Lens Protocol transfers ownership of social graphs to the participants of that graph themselves. This is achieved by creating direct links betweenprofiles and theirfollowers , while allowing fine-grained control of additional logic, including monetization, to be executed during those interactions on a profile-by-profile basis.

Here's how it works...

Profiles¶

Any address can create a profile and receive an ERC-721Lens Profile NFT. Profiles are represented by aProfileStruct :

/* *@notice A struct containing profile data. * *@param pubCount The number of publications made to this profile. * @param followNFT The address of the followNFT associated with this profile, can be empty.. * @param followModule The address of the current follow module in use by this profile, can be empty. * @param handle The profile's associated handle. * @param uri The URI to be displayed for the profile NFT.* / struct ProfileStruct { uint256 pubCount; address followNFT; address followModule; string handle; string uri; } Profiles have a specific URI associated with them, which is meant to include metadata, such as a link to a profile picture or a display name for instance, the JSON standard for this URI is not yet determined. Profile owners can always change their follow module or profile URI.

Publications¶

Profile owners canpublish to any profile they own. There are threepublication types:Post ,Comment andMirror . Profile owners can also set and initialize theFollow Module associated with their profile.

Publications are on-chain content created and published via profiles. Profile owners can create (publish) three publication types, outlined below. They are represented by aPublicationStruct :

/* * *@notice A struct containing data associated with each new publication. * * @param profileIdPointed The profile token ID this publication points to, for mirrors and comments. * @param pubIdPointed The publication ID this publication points to, for mirrors and comments. * @param contentURI The URI associated with this publication. * @param referenceModule The address of the current reference module in use by this profile, can be empty. * @param collectModule The address of the collect module associated with this publication, this exists for all publication. * @param collectNFT The address of the collectNFT associated with this publication, if any.* / struct PublicationStruct { uint256 profileIdPointed; uint256 pubIdPointed; string contentURI; address referenceModule; address collectModule; address collectNFT; }

Publication Types¶

Post¶

This is the standard publication type, akin to a regular post on traditional social media platforms. Posts contain:

1. A URI, pointing to the actual publication body'smetadata
2. JSON, including any images or text.
3. An uninitialized pointer, since pointers are only needed in mirrors and comments.

Comment¶

This is a publication type that points back to another publication, whether it be a post, comment or mirror, akin to a regular comment on traditional social media. Comments contain:

1. A URI, just like posts, pointing to the publication body'smetadata
2. JSON.
3. An initialized pointer, containing the profile ID and the publication ID of the publication commented on.

Mirror¶

This is a publication type that points to another publication, note that mirrors cannot, themselves, be mirrored (doing so instead mirrors the pointed content). Mirrors have no original content of its own. Akin to a "share" on traditional social media. Mirrors contain:

1. An empty URI, since they cannot have content associated with them.
2. An initialized pointer, containing the profile ID and the publication ID of the mirrored publication.

Profile Interaction¶

There are two types of profile interactions: follows and collects.

Follows¶

Wallets can follow profiles, executing modular follow processing logic (in that profile's selected follow module) and receiving aFollow NFT . Each profile has a connected, uniqueFollowNFT contract, which is first deployed upon successful follow. Follow NFTs are NFTs with integrated voting and delegation capability.

The inclusion of voting and delegation right off the bat means that follow NFTs have the built-in capability to create a spontaneous DAO around any profile. Furthermore, holding follow NFTs allows followers tocollect publications from the profile they are following (except mirrors, which are equivalent to shares in Web2 social media, and require following the original publishing profile to collect).

Collects¶

Collecting works in a modular fashion as well, every publication (except mirrors) requires aCollect Module to be selected and initialized. This module, similarly to follow modules, can contain any arbitrary logic to be executed upon collects. Successful collects result in a new, unique NFT being minted, essentially as a saved copy of the original publication. There is one deployed collect NFT contract per publication, and it's deployed upon the first successful collect.

When a mirror is collected, what happens behind the scenes is the original, mirrored publication is collected, and the mirror

publisher's profile ID is passed as a "referrer." This allows neat functionality where collect modules that incur a fee can, for instance, reward referrals. Note that theCollected event, which is emitted upon collection, indexes the profile and publication directly being passed, which, in case of a mirror, is different than the actual original publication getting collected (which is emitted unindexed).

Alright, that was a mouthful! Let's move on to more specific details about Lens's core principle: Modularity.

Lens Modularity¶

Stepping back for a moment, the core concept behind modules is to allow as much freedom as possible to the community to come up with new, innovative interaction mechanisms between social graph participants. For security purposes, this is achieved by including a whitelisted list of modules controlled by governance.

To recap, the Lens Protocol has three types of modules:

1. Follow Modules
2. contain custom logic to be executed upon follow.
3. Collect Modules
4. contain custom logic to be executed upon collect. Typically, these modules include at least a check that the collector is a follower.
5. Reference Modules
6. contain custom logic to be executed upon comment and mirror. These modules can be used to limit who is able to comment and interact with a profile.

Note that collect and reference modules shouldnot assume that a publication cannot be re-initialized, and thus should include front-running protection as a security measure if needed, as if the publication data was not static. This is even more prominent in follow modules, where it can absolutely be changed for a given profile.

Lastly, there is also aModuleGlobals contract which acts as a central data provider for modules. It is controlled by a specific governance address which can be set to a different executor compared to the Hub's governance. It's expected that modules will fetch dynamically changing data, such as the module globals governance address, the treasury address, the treasury fee as well as a list of whitelisted currencies.

Upgradeability¶

This iteration of the Lens Protocol implements a transparent upgradeable proxy for the central hub to be controlled by governance. There are no other aspects of the protocol that are upgradeable. In an ideal world, the hub will not require upgrades due to the system's inherent modularity and openness, upgradeability is there only to implement new, breaking changes that would be impossible, or unreasonable to implement otherwise.

This does come with a few caveats, for instance, theModuleGlobals contract implements a currency whitelist, but it is not upgradeable, so the "removal" of a currency whitelist in a module would require a specific new module that does not query theModuleGlobals contract for whitelisted currencies.