Kleros “Address Tag” Query Registry Guidelines

Version 2.3.0

The “Address Tag” query registry is a collection of modules that can be used to dynamically construct the Public Name Tags associated with ranges of contract addresses on EVM-compatible chains. This complements the original ‘Address Tag’ registry for single entries, and is meant to allow for large amounts of contract tags of a similar nature to be included in a single entry. The data from these registries are then used by blockchain explorers and wallets to provide contract insights for their users.

For brevity, some explanations in this document may refer to the Project Name and Public Name Tag in a single string separated by a colon (e.g. “*Curve.fi: CRV Token”*), even though they are two distinct fields in practice.

> The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

A **major issue** is a bug, deficiency or vulnerability which:
- Prevents the application from operating as expected, or
- Carries a risk of security, privacy breach, financial loss, or harm for the user.

A **minor issue** is a bug, deficiency or vulnerability which:
- Does not prevent the application from operating as expected, and
- Does not carry a risk of security, privacy breach, financial loss, or harm for the user.

Elements Required for Submission

An entry into the list must be composed of :

Field Name	***Description***	***Example***
Github Repository URL	The URL of the repository containing the function that returns the Contract Tags. The repository name must be in the kebab case (hyphen-case).	• *https://github.com/0xbuidler/uniswap-v3-lp-tags.git* • *https://github.com/vbuterin/oz-erc721-tags.git*

Commit hash	The hash of the specific commit for this repository to be referenced.	• *3f4a02a* • *c9dd0dc*
Chain ID	The EVM Chain ID of the chain of the contracts being retrieved by the function in this module.	• *1* • *137* • *100*
Description	A field used to describe the range of contracts being curated here, specifying (if applicable) the version, type and purpose of the contracts that are returned.	• *The Uniswap v2 LP contracts for Ethereum Mainnet.*

Module Acceptance Criteria

The criteria for a new retrieval module entry to be accepted in this registry are as follows:

- The module must make use of the data from the decentralized The Graph Network.
- Repository structure and guidelines:
    - The easiest way to build a compliant repository is to clone the repository in the Reference section below (caution: the code provided is just an example and does not form part of this policy).
    - Each repository should have one file named *main.mts* within the *src/* folder as its entry point with all logic contained within it. It must be written in Typescript and structured as an ES6 module. The code must compile without error. It must not exceed 500 lines in length.
    - The code must import and make use of the *ContractTag* and *ITagService* interfaces in the NPM package 'atq-types'.
    - The code may use the NPM packages *axios* or *node-fetch* for data retrieval. No other packages are allowed.
    - Only Yarn may be used as package manager in the module (if needed).
    - The code must not use *'this*' to reduce complexity associated with potential unexpected behavior when the code is imported and used in a different context.
    - The module must export exactly one and only one function called ***returnTags***, which must:
        - Be defined as async.

- Take in exactly two arguments.
    - The first argument must be the Chain ID in decimal form (e.g. ‘100’ for Gnosis Chain instead of the hexadecimal value ‘0x64’) rendered as a String:
        - The Chain ID must be explicitly handled. If the Chain ID is not one that the function can handle, the function must throw with a descriptive error message (e.g. ‘Unsupported Chain ID: [ChainID].’)
    - The second argument must be the API Key for the query service:
        - API Keys must not be included in the code or pulled from environmental variables, and must be provided to the returnTags function.
- Return an array of *ContractTags* (e.g. *Promise<ContractTag []’>*).
- If there is any issue with the execution or the provided arguments, the function must throw an Error and revert, instead of returning an array with array/incomplete results or looping/hanging indefinitely.
- The ContractTags returned must be constructed dynamically using data from the subgraph query responses and may not contain any hard coded entries (as those belong in the original Address Tag Registry).
    - It is not allowed to artificially limit or skip any of the entries returned from the subgraph queries unless the entry contains data that would lead to invalid entries. If so, indicate clearly in the comments why a specific subset of the entries are skipped.
    - Some contracts may be associated with two or more projects at the same time (e.g. The contract for a Safe smart contract wallet is designed by Safe, but created and used by another project). In such cases, it is allowed to use any associated project as the *‘Project Name’*, as long as all are mentioned in the *‘Public Note*’.
    - Some contracts may not be associated with any specific projects at all (e.g. some ERC20/ERC721/ERC1155 tokens). In such cases, the name of the token can be used as the project itself.
        - The “*User Interface /Website Link*” field may be left as an empty string if there is no feasible way to pull that information for each and every contract in a trustless and permissionless manner.

- Besides the pre-defined queries in the module and the usage of the API Key in the authorisation header, the code must not store or send any additional information to local or remote destinations.
- It is strongly recommended to use ESLint for linting the code to maintain code quality and consistency.
    - Do note that this is a recommendation and that linting issues must not result in a rejection of the entry.
- Every address tag returned by the ***returnTags*** function:
    - must fulfill the requirements of the Address Tags Registry for individual contracts at the time of submission unless specified otherwise in this policy.
        - An ellipsis ('...') may be used to indicate that part of a field is truncated in order to stay within the applicable length limits.
    - must not have its contract address already be included in another entry in this registry for the same chain.
        - However, it is acceptable for contracts covered by an entry to be already included in the original Address Tags Registry.
- The *package.json* file should have **at least** the following keys with these exact values (for dependencies you may exclude *axios* or *node-fetch* if not used):

```
{
    ...
    "type": "module",
    "main": "main.mjs",
    "license": "MIT",
    "dependencies":
    {
        "atq-types": "^1.1.5",
        "axios": "^1.6.8",
        "node-fetch": "^3.3.2"
    },
    "scripts":
    {
        "build": "tsc"
    }
    ...
```

```
}
```

- The *tsconfig.json* file **must be exactly this (excluding comments)**:

```
{
    "compilerOptions":
    {
        "target": "ESNext",
        "module": "NodeNext",
        "moduleResolution": "NodeNext",
        "types":
        [
            "node"
        ],
        "outDir": "./dist",
        "esModuleInterop": true,
        "forceConsistentCasingInFileNames": true,
        "strict": true,
        "skipLibCheck": true
    }
}
```

- Other requirements
  - Each module must only be used to retrieve a specific category of contracts of a specific protocol. Otherwise, a separate module must be created.

> ***Examples***:
> - *A module for retrieving all liquidity pool contracts for Uniswap v3 on Ethereum Mainnet.*
> - *A module for retrieving all 'Light Curate' contracts for Kleros Curate on Gnosis Chain*

  - The endpoints used must point to one of the gateway endpoints of the decentralized Graph Network (e.g. https://gateway-arbitrum.network.thegraph.com/api/[api-key]/deployments/id/QmYayB5NBkDuGmgJNz1B9kH3ySYfA1iLBz8X8Jv8qcobSQ), which contains the **specific deployment ID**. Even if the subgraph ID is the only identifier mentioned in a project's official documentation, you must use the latest deployment ID of that subgraph in the module.

        - Only official subgraphs published on the official documentation of the project in question.
            - Unless there is explicit mention by a project that all subgraphs published by a specific address are linked to them, this assumption cannot be made.
            - In the absence of that, only subgraphs published by Messari (i.e. *subgraphs.messari.eth*) and Builders DAO may be used.
        - If the query is meant to query an ever-expanding number of contracts (e.g. Kleros Curate contracts, DEX liquidity pool contracts), query pagination must be used.
            - If query pagination is used, only cursor-based pagination is allowed (i.e. as opposed to offset pagination, windowed queries).
            - Measures must be taken to ensure that all relevant entries are returned, the cursor implementation follow these rules:
                - Unique and sequential fields should be used as a first choice for the cursor field.
                - In the absence of the above, the entity's 'id' field can be used, provided the block is fixed across all iterative queries.
        - Be free of Major Issues as defined in the preamble, while Minor Issues should be avoided (though acceptable).
    - New entries with the same repository URL and Chain ID can be accepted if it is a commit hash that's later than the existing one. Once accepted, it will be considered to supersede earlier entr(ies) with the same repository URL and Chain ID.
    - Issues with the code that do not affect the correctness of the results returned for the specific Chain ID in the submission in question are acceptable and must not lead to the rejection of a submission.
    - Referencing these contracts by The Graph, the subgraph in question used in the entry must be available.
        - It will be considered unavailable if both of the following are true
            - **Zero Curation signal** (i.e. zero GRT signalled by curators on the *L2Curation* contract) and **Zero Active Allocation** (i.e. zero GRT staked by indexers on the *L2Staking* contract) for the subgraph's

deployment ID for at least **72 hours** continuously prior to the time of challenge

- There have been **no valid Proofs of Indexing (POI)** submitted for any allocation of that Deployment ID within the last **7 days** prior to the time of challenge.

- For both of the above, The Graph's 'Upgrade Indexer' is not considered a valid indexer.

Removal Request

A request to remove an accepted entry from the list can be made at any time by anyone submitting a deposit.

To assist the jury, the removal requester **should** justify why the entry should be removed by either:

- Showing that there is an updated entry for the same repository with a higher commit, rendering the entry obsolete.
- Providing evidence that one of the above-cited acceptance criteria is not (or no longer) fulfilled by the entry.
- Or explaining why a change in the underlying contract parameters or context justifies the adjustment of any part of the tag submission in order to prevent potential user confusion.

In any case, if a challenge is raised, jurors are expected to review the entire code (i.e. main.mts) for all possible issues, even if the issues were not explicitly raised by the challenger. This supersedes court policies instructing jurors to refuse-to-arbitrate if a limited scope is not defined.

References

Custom helper GPT

Use this custom GPT to help you with checking and creating your module before submission: https://chat.openai.com/g/g-fVvq1bQrb-kleros-atq-quick-check . Take note to do your own additional checks as the results from the custom GPT is purely advisory.

Sample module

Check out the repository here for an example of how the module should be implemented: https://github.com/gmkung/balancer-v2-pools-atq-module.git