# Aleph Documentation

*Release 1.1*

**Friedrich Lindenberg**

**Jun 20, 2017**

Contents

> Truth cannot penetrate a closed mind. If all places in the universe are in the Aleph, then all stars, all lamps, all sources of light are in it, too.
>
> —The Aleph, Jorge Luis Borges

**Aleph** is a tool for indexing large amounts of both unstructured (PDF, Word, HTML) and structured (CSV, XLS, SQL) data for easy browsing and search. It is built with investigative reporting as a primary use case. Aleph allows cross-referencing mentions of well-known entities (such as people and companies) against watchlists, e.g. from prior research or public datasets.

Here's some key features:

- Web-based UI for search across large document and data sets.
- Watchlist editor for making custom sets of entities to be tracked.
- Equal support for structured (i.e. tabular) and unstructured (i.e. textual) sources.
- Importers include a local filesystem traverser, web crawlers and a SQL query importer.
- Document entity tagger (regular expressions-based, and optionally using NLP).
- Support for OCR, unpacking Zip/RAR/Tarballs, language and encoding detection.
- Entity watchlist importers for OpenNames and Investigative Dashboard.
- OAuth authorization and access control on a per-source and per-watchlist basis.
- Excel export for search result sets.

CHAPTER 1

# Documentation

The documentation for Aleph is available at aleph.readthedocs.io. Feel free to edit the source files in the `docs` folder and send pull requests for improvements.

To build the documentation, please install the dependencies first and run `make docs`:

```
(host)$ docker-compose run app bash
(app) $ pip install -r requirements-docs.txt
(app) $ make docs
```

Now you can browse the documentation locally at `http://lvh.me:8000/docs/_build/html/`:

```
(host)$ make docs-web
```

CHAPTER 2

# Support

Aleph is used by multiple organisations, including Code for Africa, OCCRP and OpenOil. For coordination, the following mailing list exists: aleph-search

If you find any errors or issues using Aleph please file an issue on GitHub or contact the mailing list.

CHAPTER 3

# Table of contents

## Installation

Aleph requires a couple of services in order to operate. To make it easier for development and deployments it uses Docker containers. Below you will find the installation steps on how to install Aleph locally for development and production environment.

### Prerequisites

Before we continue, you will need to have Docker and `docker-compose` installed. Please refer to their manual to learn how to set up Docker and docker-compose.

You will also need to edit a configuration file to provide some credentials for the external services. This includes the OAuth credentials to allow Google users to login to Aleph and an email server credentials. Email server support is optional for development purposes.

Inside the same repository you will find a file called `aleph.env.tmpl`. This is a template of the configuration file. Make a copy of this file named `aleph.env` and follow the steps below to edit it.

To get the OAuth credentials please visit the Google Developers Console. There you will need to create an API key. In the **Authorized redirect URIs** section, use this URL:

```
http://lvh.me:13376/api/1/sessions/callback/google
```

Save the client ID and the client secret as `ALEPH_OAUTH_*` values.

Finally you will need to provide a value for the `ALEPH_SECRET_KEY`. A good example of a value is the output of `openssl rand -hex 24`.

### Development installation steps

Insider the Aleph repository you will find a `Dockerfile` and a `docker-compose.dev.yml` files. These are used to build a container with the application and start the relevant services.

To proceed run:

1. `make build` to start the application and relevant services. You can leave this open to have access to the development logs.
2. `make upgrade` to run the latest database migrations and create/update any indexes.
3. Open `http://lvh.me:13376/` in your browser and proceed with the login.

Your repository is mounted inside the docker container under the name `aleph_app`. You can access these services anytime by running `make shell`.

### Building from a clean state

You can also build the Aleph images locally. This could be useful while working on the Dockerfile changes and new dependency upgrades.

Aleph provides two commands to build the images. First one is `make base`, this will build the `alephdata/base` image (this is an intermediary image with system-level dependencies for Aleph). The second one is `make build`, this will build the `alephdata/aleph` image (this will generate a production ready image).

## Front-end

Aleph is transitioning the front-end codebase towards a more modern architecture and while this is still a work-in-progress, some of the features already landed and should make the front-end development easier.

An LTS version of Node.js with NPM is required before we continue. First you will need to install the development packages (at the moment the build tool uses Webpack 2): `npm install .`. If you are using Docker, none of this is required.

In order to build the front-end you will need to run: `make assets`. The front-end assets are always built when you start the application.

If you are working on the front-end, you will need to start the assets watcher in parallel:

```
make assets-dev
```

While working on the front-end development, make sure you disable browser cache!

## Production deployment

Aleph runs on PostgreSQL and ElasticSearch along with a couple of system tools like OpenOffice, ImageMagik, Tesseract and wkhtmltopdf. For a full list of system dependencies please review the `aleph_base` Dockerfile.

If you decide to not use Docker compose, you will have to provide all these dependencies and services and change the configuration file accordingly. An application only Docker image is also available at `alephdata/aleph`.

Finally, aleph is optimized to use certain Amazon Web Services: SQS and S3. To enable AWS features, you will need to set the AWS key ID and access key in the configuration file. Amazon SQS support is available for task queueing. Where S3 is available for file uploads.

### Upgrading

Aleph does not provide automatic upgrades. You will have to download the new version Docker images or checkout the latest version using Git first.

Once you have the latest version, you can run the command bellow to upgrade the existing installation.

```
make upgrade
```

## Configuration

Most of the Aleph configuration is handled via a set of values in a Python configuration file. The defaults are documented in the default_settings.py file and can be overridden by creating a configuration file named `settings.py` in the aleph base directory.

While using Docker, the config file, in turn, is largely configured using environment variables in accordance with 12 factor principles. These environment variables can be found also in docker_settings.py.

### Feature options

- `TIKA_URI` - when enabled, this will use Apache Tika to extract content from PDF files, rather than the built-in `pdfminer` and `tesseract` modules. The URI must point to a Tika server endpoint, which is also responsible for handling OCR.

  **Note:** using Tika with OCR'd documents may yield different results from the built-in mechanism and OCR may not be performed on the same sections of a document's content (See: #104).

## Running tests

To run the tests, assuming you already have the `docker-compose` up and ready, run `make test`.

This will create a new database and run all the tests.

The test settings can also be configured by making a copy of the `test_settings.py.tmpl` file to `test_settings.py` and editing it to match your configuration. You must then set the environment variable `ALEPH_TEST_SETTINGS` to point to the absolute path of that settings file.

# Usage

Aleph main features are available through the web interface. Data loading and other maintenance operations are provided via command line tools. In order to understand the concepts and how it works, please refer to the chapters bellow.

## Use cases

Below are some of common use cases Aleph was designed to cover. To learn more about the needs of such a tool, please refer to the notes about the page on user needs in investigative journalism.

There's also a *glossary* describing the keywords used in Aleph.

Consider some common use-cases like:

- As a journalist, I want to combine different types of facets which represent document and entity metadata.
- As a journalist, I want to see a list of documents that mention a person/org/topic so that I can sift through the documents.
- As a journalist, I want to intersect sets of documents that mention multiple people/orgs/topics so that I can drill down on the relationships between them.
- As a data importer, I want to routinely crawl and import documents from many data sources, including web scrapers, structured sources and filesystems.

- As a data importer, I want to associate metadata with documents and entities so that users can browse by various facets.

## Crawling data

One way to get the data into Aleph is to provide files and folders it can crawl and load the content to the database.

### From files and folders

Aleph provide a tool to process all the files and folders using an input path. Some files such as archives (ZIP packages or Tarfiles) will be treated as as *virtual* folders and all their content will be imported under its name.

It is important to note that this method of loadig data provides very limited ways of including metadata (ex.: document titles, source URLs or document languages). To overcome this, read about *Metafolders*.

To use this tool, run

```
docker-compose run app python aleph/manage.py crawldir <DIRECTORY|FILE PATH>
```

optional `crawldir` parameters:

- `-f,--foreign_id COLLECTION_IMPORT_ID` look up the import id in the settings tab of a collection
- `-c,--country COUNTRY`
- `-l,--language LANGUAGE`

It is important to mention that importing the same directory multiple times will not duplicate the source files as long as the base path of the crawl is identical. The base path of the file is used to identify the document source.

### From SQL

This method was removed in Aleph 1.1 to make place for future graph loaders. The new loaders will provide support for this feature.

If you are interested in bringing this feature back, please get in touch with us. The code is still available and can be packaged as a plugin.

### Metafolders

Is a format developed for Aleph to bulk-import many documents while retaining relevant metadata. It can be used in scrapers and data cleaning scripts. Aleph provides also the tools to work with the format and process Metafolder files.

Metafolder files can be generated easily using the Python metafolder library, and the krauler web crawling/scraping tool.

For the metafolder items to be loaded it is important to include the information on the document source which they are to be associated with.

An example with minimal metadata is available below. To learn more about metadata fields available in Aleph, please check the Glossary.

```
{
  "title": "Document title (not required)",
    "collection": {
      "label": "The Banana Republic Leaks",
      "foreign_id": "banana:republic"
```

```
    }
}
```

Now this folder can be loaded using:

```
docker-compose run app python aleph/manage.py metafolder <YOUR METAFOLDER PATH>
```

# Extending Aleph (Plugins)

Aleph's functionality can be extended via a system of plug-ins which are small Python modules.

Plugins can be located both within the main aleph code base, or in an external Python module. They are registered and activated using a distutils entrypoint:

An example:

```
from setuptools import setup, find_packages

setup(
    name='aleph_occrp',
    version='0.2',
    [...]
    entry_points={
        'aleph.init': [
            'occrpext = aleph_occrp:register'
        ]
        'aleph.crawlers': [
            'my_crawler = aleph_occrp.my_crawler:CrawlerClass'
        ]
    }
)
```

See the main setup.py for a real example.

The supported entry points include:

- `aleph.init`, for simple functions to be executed upon system startup.
- `aleph.crawlers`, for *Crawlers*
- `aleph.ingestors` to support additional file type imports.
- `aleph.analyzers`, which are run to extract structured metadata from documents after they have been imported.

## Signals

The documentation for this part is missing at the moment.

## Custom SCSS

An additional environment variable, `CUSTOM_SCSS_PATH`, can be used to specify the path to a SCSS file which will be imported into the application upon start. The given path must be absolute, or relative to the run-time location of Aleph. An example would be:

```
docker-compose run -e CUSTOM_SCSS_PATH=my.scss app make web
```

## Creating new crawlers

Custom crawlers are useful to directly import large amounts of data into the system. This can make sense for custom scrapers or crawlers where the indirection of using Metafolders is not desirable.

Crawlers are Python classes and exposed via the `entry_point` of a Python package. To develop a custom crawler, start by setting up a separate Python package with it's own `setup.py` (learn more).

A basic crawler will extend the relevant `Crawler` class from Aleph and implement its `crawl()` method, below you can find an example:

```
from aleph.crawlers import DocumentCrawler

class ExampleCrawler(DocumentCrawler):
    COLLECTION_ID = 'example'

    def crawl(self):
        for i in range(0, 1000):
            meta = self.metadata()
            meta.foreign_id = 'example-doc:%s' % i
            meta.title = 'Document Number %s' % i
            meta.mime_type = 'application/pdf'
            url = 'https://example.com/documents/%s.pdf' % i
            self.emit_url(meta, url)
```

Besides `emit_url`, results can also be forwarded using the `emit_file(meta,file_path)` method. If a crawler creates collections, it can use `emit_collection(collection,entity_search_terms)` which will start a partial re-index of documents.

To support indexing only new documents on incremental/update crawls, you can use `self.skip_incremental`:

```
    if self.skip_incremental(foreign_id):
        logger.info("Skipping known %s", foreign_id)
        return
```

In order to make sure that Aleph can find the new crawler, it must be added to the `setup.py` of your package, see above how plugins work:

```
setup(
    name='mypackage',
    ...
    entry_points={
        'aleph.crawlers': [
            'example = mypackage.example:ExampleCrawler'
        ]
    }
)
```

Finally, you must ensure that the plugin package is installed in your `aleph` docker container (or using your deployment method), for example by extending the `Dockerfile` to include the plugin package. Once this is ready, run the crawler from inside the container:

```
docker-compose run app python aleph/manage.py crawl example
```

### Custom OAuth

It's possible to hook into the login code to support other providers, but you need to handle the creation of user and group roles through some specific code. This is the code used at OCCRP for OAuth via the Investigative Dashboard (it requires the use of plugins to be activated:

```
from aleph import signals

@signals.handle_oauth_session.connect
def handle_occrp_oauth(sender, provider=None, session=None):
    from aleph.model import Role
    if 'investigativedashboard.org' not in provider.base_url:
        return
    me = provider.get('api/2/accounts/profile/')
    user_id = 'idashboard:user:%s' % me.data.get('id')
    role = Role.load_or_create(user_id, Role.USER,
                               me.data.get('display_name'),
                               email=me.data.get('email'),
                               is_admin=me.data.get('is_admin'))
    role.clear_roles()
    for group in me.data.get('groups', []):
        group_id = 'idashboard:%s' % group.get('id')
        group_role = Role.load_or_create(group_id, Role.GROUP,
                                         group.get('name'))
        role.add_role(group_role)
    session['user'] = role.id
```

## Mapping Files

One of Aleph's primary task is bulk entity loading from structured datasets. Aleph has a YAML syntax to represent the database operations necessary to do this. The syntax is then translated into a Python `dict` under the hood.

Dataset paths are stored in a master YAML file, the location of which should be specified in `DATASETS_YAML`, in the format

```
datasets:
    include:
        - path/to.yml
```

Mappings are loaded with `aleph loaddataset <mapping name>`. The mapping's name is the top-level property of a mapping file. A mapping consists of information identifying the datasets, their sources, their types, and possible access information, and a list of database queries:

```
<mapping name>:
    label: <the dataset or sets being mapped>
    info_url: <source url, if available>
    category: <the type--company, land, etc.--of entities being loaded>
    roles:
        - <user roles>
        - ...
    queries:
        ...
```

- MAPPING NAME: A string to identify this mapping. Is used with `aleph loaddataset` to load entities from a set of tables.

- LABEL: The name of the source.
- INFO_URL: The source url.
- CATEGORY: What kind of record the source provides (company, land, etc.)
- ROLES: User roles that are permitted to access this information. When no credentials are required, 'guest' suffices.
- QUERIES: The database operations that construct entities and links from DB records.

## Queries

A basic query specifies an entity to build out of the data in a given table. We can extend this operation by adding additional tables and filters, and by identifying links between different entities. Entities and links have schemas specified in `aleph/schema.yaml` (such as Entity, LegalEntity, Company, Directorship, etc.) that provide structure for the objects and relations that we're interested in loading.

A query has the following form:

```
- database: <database uri>
  tables:
      - table: <table>
        alias: <table alias>
      - ...
  (joins:)
      - left: <table or alias>.<column>
        right: <table or alias>.<column>
      - ...
  (filters/filters_not:)
      - <table>.<column>: <value>
      - ...
  (where/where_not:)
      - <table>.<column> <SQL predicate> <value>
  entities:
      <entity name>:
          schema: <schema type>
          keys:
              - <table>.<column>
              - ...
          properties:
              <property name>:
                  column: <table>.<column>
                  (literal:) <string>
              ...
  (links:)
      - schema: <schema type>
        source: <table>
        target: <table>
        properties:
            <property name>:
                column: <table>.<column>
                (literal:) <string>
      - ...
```

Note that - is interpreted as an item in a list. Tables, keys, joins, and filters can all be referred to in the singular if only one item is needed (i.e. `table: <table name>`) without -.

- DATABASE: A database uri. Example: `postgresql://user:password@postgres/database`
- CSV_URL: Alternative to DATABASE. Supported protocols: http, https, file
- TABLES: A table or list of tables with (optional) aliases.
- JOINS: Join(s) to perform (note that under the hood, this is a WHERE a = b operation; may change in the future) (optional).
- FILTERS/FILTERS_NOT: Filter by column values (optional).
- WHERE/WHERE_NOT: Filter by SQL where statement (optional).
- ENTITIES:
  - SCHEMA: An entity schema as found in `aleph/schema.yaml`.
  - KEYS: Values to use for a unique identifier (sha1 digest of properties).
  - PROPERTIES: Entity properties, as specified by the chosen schema.
    - PROPERTY:
      - COLUMN: Specifies a column value to use.
      - LITERAL: Or you may provide a literal string value to use (optional).
- LINKS: (optional; schema, keys, and properties are defined as with entities)
  - SOURCE: The source of the link is its object–if we are linking a person to a company as its director, for instance, we refer to the company as the source.
  - TARGET: The entity that is the subject of the link. In the above example, the company's director would be the target of the link.

## Notes for making a good mapping (chill the join)

Mappings are *not* intended to perform expensive cleanup operations. Data should be reasonably structured and indexed where necessary. This also means keeping queries reasonable. In general, when linking, it's best practice to fully specify entities in their own queries, and then to join them using a skeleton representation. That means that rather than do this

```
- database: some://database/uri
  tables:
    - first_table
    - second_table
  joins:
    - left: first_table
      right: second_table
  entities:
      first_entity:
          schema: LegalEntity
          keys:
              - first_table.name
              - first_table.address
              - first_table.id_number
          properties:
              name:
                  column: first_table.name
              address:
                  column: first_table.address
```

```
            id_number:
                column: first_table.id_number
    second_entity:
        schema: LegalEntity
        keys:
            - second_table.name
            - second_table.address
            - second_table.id_number
        properties:
            name:
                column: second_table.name
            address:
                column: second_table.address
            id_number:
                column: second_table.id_number
links:
    - schema: link
      source: first_entity
      target: second_entity
```

we prefer

```
- database: some://database/uri
  tables:
    - first_table
  entities:
    first_entity:
        schema: LegalEntity
        keys:
            - first_table.name
            - first_table.address
            - first_table.id_number
        properties:
            name:
                column: first_table.name
            address:
                column: first_table.address
            id_number:
                column: first_table.id_number

- database: some://database/uri
  tables:
    - second_table
  entities:
    second_entity:
        schema: LegalEntity
        keys:
            - second_table.name
            - second_table.address
            - second_table.id_number
        properties:
            name:
                column: second_table.name
            address:
                column: second_table.address
            id_number:
                column: second_table.id_number
```

```
- database: some://database/uri
  tables:
    - first_table
    - second_table
  entities:
    first_entity:
      schema: LegalEntity
      keys:
          - first_table.name
          - first_table.address
          - first_table.id_number
      properties:
          name:
              column: first_table.name
    second_entity:
      schema: LegalEntity
      keys:
          - second_table.name
          - second_table.address
          - second_table.id_number
      properties:
          name:
              column: second_table.name
  links:
    - schema: link
      source: first_entity
      target: second_entity
```

In order to avoid key collision, it's also best to specify several key columns containing identifying information (like name, address, and id_number above).

# Glossary

Here is some of the vocabulary used by the application.

## Documents

These are the basic search results in Aleph. They can either be *text documents* (in which case they are expected to have a number of pages and are shown as a PDF in the user interface), or *tabular documents* (in which case they can have multiple sheets, each with a number of records. They will be shown as tables in the user interface).

## Collections

These are units documents and entity data used to group items in Aleph. This can be the name of an imported directory, database, or something more abstract, such as the name of an organisation or web site that has been ingested. It can also be used to group documents and entities relevant to a particular investigation.

## Foreign IDs

These exist both for documents and collections. They are a little piece of text used to identify the origin of information, e.g. the URL of a crawled document, or the path of an imported folder. Used to avoid duplicate imports when importing

the same content twice.

## Metafolder

Represents a format and tools to store a set of documents before importing them into Aleph. It's benefit is storing *metadata* alongside the actual files that are to be imported, while separating Aleph from previous workflow stages (e.g. the scraping of a web site).

Metafolders can be generated using the Python metafolder library, and the krauler web crawling/scraping tool.

## Crawlers

Little plug-ins to the Aleph engine which import data into the system. The included crawlers are very flexible, such as `DirectoryCrawler` or `MetaFolderCrawler`, but specific crawlers can be programmed that will import data from a specific source.

## Ingestors

Plug-ins to the Aleph engine which accept crawled files and attempt to extract text pages or tabular rows from them so that they can be imported into the system. Ingestors for common file formats like Word documents, PDF files or CSV spreadsheets are included, but more exotic types can be supported by programming additional ingestors.

## Analyzers

Is what Aleph runs after the documents have been ingested. They are used to extract additional information from a document. Examples include the extraction of entities and language detection. Again, new analyzers can be added through the plug-in system.

## Plugins

To Aleph, plugins are Python classes in a Python distutils package which are exposed via the `entry_points` mechanism. These include crawlers, ingestors and analyzers. See `setup.py` in the repository root for examples.

## Metadata

This is used to describe the content of individual documents in Aleph.

Common metadata fields include:

- `title`: a short document title (will be extrapolated from the file name, if none is given).
- `summary`: an optional short description, usually less than 200 characters.
- `file_name`: the basename of the imported file, e.g. `Source_Data.xlsx`. If not provided, this will be guessed from the `source_url` or `source_path`)
- `foreign_id`: see above, e.g. a source URL, or foreign systems ID
- `extension`: e.g. `pdf`, `csv`, without the dot
- `mime_type`: e.g. `text/csv`, `application/pdf`
- `source_url`: e.g. `http://source.com/documents/Source_Data.xlsx`

- `source_path`: local import path, e.g. `/tmp/Source_Data.xlsx`
- `languages`: a list of lowercase two-letter ISO 3166-1 language codes, e.g `['ja','en']`
- `countries`: a list of lowercase two-letter ISO country codes, e.g `['jp','en']`
- `dates`: a list of ISO 8601 dates relevant to the document, eg. `['2001-01-28']`
- `keywords`: a list of key phrases.
- `emails`: email addresses extracted from the text.
- `headers`: a hash of the headers received upon download of the document via HTTP.
- `content_hash`: a SHA-1 checksum of the data (automatically generated).

Other fields can be added, but they will not usually be shown in the user interface.

# External resources and links

Mostly external resources worth mentioning.

## Articles

Below you can find a curated list of articles and blog posts about the purpose behind this project.

## Alternatives and related projects

Aleph is one of many document processing and search tools targeted at journalists, activists etc. Many of these are similar in scope, Aleph aims to distinguish itself by providing entity cross-referencing and seamless support for both tabular and textual data.

- DocumentCloud, the biggest document hosting site for journalistic content, including OCR and organisation/project-level access control.
- DARPA MEMEX, a coordinated research to make domain-specific deep web search engines.
- Overview Project, document mining tool with plugin architecture, both hosted & local
- Transparency Toolkit, LookingGlass is an indexing server for JSON documents with support for theming, used mainly for scraped social media profiles.
- resourcecontracts.org, visual browser for resource (oil, mining, etc.) contract documents.
- ICIJ Extract, Java-based OCR and content extraction pipeline used for large-scale leaks.
- Hoover, Python-based search engine.
- mma-dexter, used by Media Monitoring Africa to do content classification and guided entity extraction of South African media.
- Omecca, eprints, fedora, dspace

**Defunct but interesting**

- datawi.re, doc mining as a timeline
- analice.me, document management and data extraction tool by Hacks/Hackers Buenos Aires.
- Unveillance, harlo's git-annex-based topic modelling tool

**Framework-ey stuff**

- OpenCalais, LingPipe, AlchemyAPI
- Apache Airflow
- nltk, patterns

### Text mining bookmarks

Here's some relevant Python text mining bookmarks:

- https://bugzilla.redhat.com/show_bug.cgi?id=191060#c1
- https://github.com/deanmalmgren/textract/blob/master/textract/parsers/pptx_parser.py
- https://github.com/chardet/chardet
- https://github.com/PyYoshi/cChardet (a faster character detection library for Python)
- http://poppler.freedesktop.org/
- http://www.unixuser.org/~euske/python/pdfminer/index.html
- https://mstamy2.github.io/PyPDF2/#documentation
- http://pybrary.net/pyPdf/pythondoc-pyPdf.pdf.html
- https://svn.apache.org/viewvc/httpd/httpd/branches/2.2.x/docs/conf/mime.types?view=annotate
- pdfminer.six
- tesserocr

## Research topics

Topics in need of more research.

### Domain model ideas

- Each imported document is either tabular or textual. It has many records, i.e. data rows or document pages.
- An entity (such as a person, organisation, or topic) is like a permanent search query; each entity can have multiple actual search terms associated with it (`selectors`).
- Documents matching an entity after that entity has been created yield notifications if a user is subscribed.

### Decentralized pipeline ideas

The idea is to pick various subsets of functionality out of a larger continuum of possible tasks related to document and data processing and try to modularise or separate to components and stages of the processing pipelines. This way these could become re-usable while deployed in the differing contexts of the various tools.

Related:

## Entity merging

De-dupe TODO:

1. merge identifiers
2. merge properties
3. merge names, make merged names into a.k.a's
4. merge collections
5. update references
6. update alerts
7. delete source entities
8. update source entities
9. update target entity

## Open design questions

### Entity graph model

The idea is about making a formal instead of a (i.e.) corporate graph. Let's make a messy one that has all the attributes we can derive from our internal data structure. Then let's use it as a recommendation engine, rather than an academic research object :)

### Authorisation

Making sure that users can only see the parts of the graph to which they have explicit access is the hardest part of this. Every node in the graph needs to be associated with one or many collections, and every user querying the database has access to several hundred collections. The following are options for modelling this:

- Make each `Collection` a node and connect it to all its subjects using `PART_OF` relationships. Query these links at the same time as the actual data.
- Add labels to each node to express the `Collections` that it belongs to. This fails because it is impossible to do an OR search on node labels in Neo4J.

Neo4J Lead Generation Patterns:

```
MATCH (c:Collection)<-[:PART_OF]-(src)
MATCH pth = (src)-[*1..3]-(dest)
MATCH (nc:Collection)
WHERE
    all(n IN nodes(pth) WHERE (n)<-[:PART_OF]-(nc))
    AND nc.id IN [250]
RETURN src, pth, dest
LIMIT 10;
```

```
MATCH (c:Collection)<-[:PART_OF]-(src)
MATCH pth = (src)-[*1..3]-(dest)
MATCH (nc:Collection)

    all(n IN nodes(pth) WHERE (n)<-[:PART_OF]-(nc))
```

```
RETURN pth
LIMIT 10;
```

```
MATCH (c:Collection)<-[:PART_OF]-(src)
MATCH pth = (src)-[*1..3]-(dest)
MATCH (nc:Collection)
WHERE
    c.alephCollection = 250
    AND nc.alephCollection IN [250, 39]
    AND all(n IN nodes(pth) WHERE (n)<-[:PART_OF]-(nc))
RETURN pth
LIMIT 10;
```

### Model ideas

- Actor (actorName, actorFingerprint, actorLegalEntity, actorCompany, actorPerson)
  - UNDER_JURISDICTION Country
  - PART_OF Collection
  - LOCATED_AT Address
  - REACHABLE_AT PhoneNumber
  - REACHABLE_AT EMail
  - AUTHORED Document
  - BORN_AT Date
  - DIED_AT Date
  - FOUNDED_AT Date
  - DISSOLVED_AT Date
- Country (countryName, countryCode)
- Collection (collectionId, collectionName)
- Document (documentTitle, documentId, documentType)
  - MENTIONS Actor
  - MENTIONS PhoneNumber
  - MENTIONS EMail
  - PART_OF Collection
  - MENTIONS Date
- PhoneNumber (phoneNumber)
  - LOCATED_IN Country
- Address (addressText)
  - LOCATED_IN Country
- EMail (emailAddress)
  - LOCATED_IN Country

- Date (yearMonthDay)

### Indexing notes

Neo4J queries can go from instantaneous to horrible based on the existance of an index, much quicker than Postgres. Here's the current indexing strategy:

```
MERGE (n) SET n:Aleph;
MERGE (n:Collection) REMOVE n:Aleph;

DROP INDEX ON :Entity(id);
DROP INDEX ON :Phone(id);
DROP INDEX ON :Email(id);
DROP INDEX ON :Document(id);
DROP INDEX ON :Address(id);

DROP INDEX ON :Entity(fingerprint);
DROP INDEX ON :Phone(fingerprint);
DROP INDEX ON :Email(fingerprint);
DROP INDEX ON :Document(fingerprint);
DROP INDEX ON :Address(fingerprint);

CREATE CONSTRAINT ON (n:Aleph) ASSERT n.id IS UNIQUE;
CREATE INDEX ON :Aleph(fingerprint);
```

### Loading external graph data

The purpose of this function is to add structured graph data - such as company registries, contract or concessions info, or financial transactions to the graph database backing aleph. It will then make this graph data available as recommendations and through the scene editor.

Medium-term, the intention is to make the mappings used by this component into user-editable parts of the aleph interface, such that any tabular data uploaded can be woven into the graph.

Mapping file:

```
## Database configuration URL:
# Can also be defined as DATABASE_URI in the environment.
database: postgresql://localhost/database

## Destination collection configuration:
collection: my_collection_foreign_id
```

### Use case: African mining concessions

- Which company holds the most concessions across all datasets?
- Longest chains of ownership -
- Can we track them back to Exhibit 21 structures, who is the BO?
- Can we make links to offshore datasets (PP, OL, BS, PA)?

#### Use case: Moldavian linkages

- Small networks that have a large extent of control of Moldavian economy.
- Small networks connected to political actors (e.g. Parliament).
- Clusters within the larger economy
- Public contracts that connect to PEPs
- Public contracts that connect to the procurement blacklist

#### Use case: PEPs and companies – across all registers.

- Run all PEPs from EP & Aleph against all offshore registers and point out the ultimate children in an ownership chain.

#### Use case: EU transparency data

- Show all advisory group member companies and persons that also were awarded EU-wide contracts.

## Sphinx AutoAPI Index

This page is the top-level of your generated API documentation. Below is a list of all items that are documented here.

### aleph

### aleph.analyze

`aleph.analyze.`**`analyze_document`**(*document*)
Run analyzers (such as NER) on a given document.

`aleph.analyze.`**`analyze_document_id`**(*document_id*)
Analyze a document after looking it up by ID.

`aleph.analyze.`**`analyze_documents`**(*collection_id*)

`aleph.analyze.`**`install_analyzers`**()
Download linguistic resources for the analyzers.

### aleph.analyze.analyzer

#### Analyzer

**class** `aleph.analyze.analyzer.`**`Analyzer`**

#### Imports

- object

**__init__**(*self*, *document*)

**finalize**(*self*)

**on_text**(*self*, *text*)

**prepare**(*self*)

## aleph.analyze.corasick_entity

### AhoCorasickEntityAnalyzer

**class** aleph.analyze.corasick_entity.**AhoCorasickEntityAnalyzer**

#### Imports

- <UNKNOWN>

**finalize**(*self*)

**on_text**(*self*, *text*)

**prepare**(*self*)

### AutomatonCache

**class** aleph.analyze.corasick_entity.**AutomatonCache**

#### Imports

- object

**__init__**(*self*)

**_generate**(*self*)

**generate**(*self*)

## aleph.analyze.language

### LanguageAnalyzer

**class** aleph.analyze.language.**LanguageAnalyzer**

#### Imports

- <UNKNOWN>

**finalize**(*self*)

**identifier**(*self*)

**on_text**(*self*, *text*)

**prepare**(*self*)

## aleph.analyze.polyglot_entity

### PolyglotEntityAnalyzer

**class** aleph.analyze.polyglot_entity.**PolyglotEntityAnalyzer**

#### Imports

- <UNKNOWN>

**finalize**(*self*)

**on_text**(*self*, *text*)

**prepare**(*self*)

## aleph.analyze.regex

### EMailAnalyzer

**class** aleph.analyze.regex.**EMailAnalyzer**

#### Imports

- aleph.analyze.regex.RegexAnalyzer

**on_match**(*self*, *match*)

### PhoneNumberAnalyzer

**class** aleph.analyze.regex.**PhoneNumberAnalyzer**

#### Imports

- aleph.analyze.regex.RegexAnalyzer

**on_match**(*self*, *match*)

### RegexAnalyzer

**class** aleph.analyze.regex.**RegexAnalyzer**

#### Imports

- <UNKNOWN>

**finalize**(*self*)

**on_text**(*self*, *text*)

**prepare**(*self*)

## aleph.archive

aleph.archive.**archive_from_config**(*config*)

## aleph.archive.archive

### Archive

**class** aleph.archive.archive.**Archive**

#### Imports

- object

**_get_prefix**(*self*, *content_hash*)

**archive_file**(*self*, *file_path*)
Import the given file into the archive.

**cleanup_file**(*self*, *content_hash*)

**generate_url**(*self*, *content_hash*)

**load_file**(*self*, *content_hash*)

**upgrade**(*self*)
Run maintenance on the store.

## aleph.archive.file

### FileArchive

**class** aleph.archive.file.**FileArchive**

#### Imports

- <UNKNOWN>

**__init__** (*self*, *config*)

**_locate_key** (*self*, *content_hash*)

**archive_file** (*self*, *file_path*)
Import the given file into the archive.

**load_file** (*self*, *content_hash*)

## aleph.archive.s3

### S3Archive

**class** aleph.archive.s3.**S3Archive**

#### Imports

- <UNKNOWN>

**__init__** (*self*, *config*)

**_get_local_prefix** (*self*, *content_hash*)

**_locate_key** (*self*, *content_hash*)

**archive_file** (*self*, *file_path*)

**cleanup_file** (*self*, *content_hash*)
Delete the local cached version of the file.

**generate_url** (*self*, *content_hash*)

**load_file** (*self*, *content_hash*)

**upgrade** (*self*)
Make sure bucket policy is set correctly.

## aleph.authz

aleph.authz.**get_public_roles**()
Roles which make a collection to be considered public.

### Authz

**class** aleph.authz.**Authz**

#### Imports

- object

### Summary

Hold the authorization information for a user.

This is usually attached to a request, but can also be used separately, e.g. in the context of notifications.

**__init__**(*self*)

**__repr__**(*self*)

**_collection_check**(*self*, *collection*, *action*)

**check_roles**(*self*, *roles*)

**collection_public**(*self*, *collection*)

**collection_read**(*self*, *collection*)
Check if a given collection can be read.

**collection_write**(*self*, *collection*)
Check if a given collection can be written.

**collections_intersect**(*self*, *action*, *colls*)
Intersect the given and the available set of collections.

This will return all available collections if the given set is empty and the `default_all` argument is `True`.

**require**(*self*, *pred*)

**session_write**(*self*)

## aleph.core

aleph.core.**configure_alembic**(*config*)

aleph.core.**create_app**()

aleph.core.**get_app_name**()

aleph.core.**get_app_secret_key**()

aleph.core.**get_app_title**()

aleph.core.**get_app_url**()

aleph.core.**get_archive**()

aleph.core.**get_config**(*name*)

aleph.core.**get_datasets**()

aleph.core.**get_es**()

aleph.core.**get_es_index**()

aleph.core.**get_language_whitelist**()

aleph.core.**get_schemata**()

aleph.core.**get_upload_folder**()

aleph.core.**url_for**()
Generate external URLs with HTTPS (if configured).

## aleph.crawlers

aleph.crawlers.**execute_crawler**(*crawler_id*)

aleph.crawlers.**execute_scheduled**()

aleph.crawlers.**get_exposed_crawlers**()
Return all crawlers which can be run automatically via the web UI.

## aleph.crawlers.crawler

### Crawler

**class** aleph.crawlers.crawler.**Crawler**

#### Imports

- object

**get_id**(*cls*)

**__init__**(*self*)

**__repr__**(*self*)

**collection**(*self*)

**crawl**(*self*)

**create_document**(*self*)

**execute**(*self*)

**increment_count**(*self*)

**load_collection**(*self*, *data*)

**make_meta**(*self*)

**save_data**(*self*, *data*)
Store a lump object of data to a temporary file.

**save_response**(*self*, *res*)
Store the return data from a requests response to a file.

**skip_incremental**(*self*, *foreign_id*)

**to_dict**(*self*)

### CrawlerException

**class** aleph.crawlers.crawler.**CrawlerException**

#### Imports

- exceptions.Exception

#### CrawlerMetadata

**class** aleph.crawlers.crawler.**CrawlerMetadata**

##### Imports

- <UNKNOWN>

**__init__** (*self*, *data*)

#### DocumentCrawler

**class** aleph.crawlers.crawler.**DocumentCrawler**

##### Imports

- aleph.crawlers.crawler.Crawler

**emit_file** (*self*, *document*, *file_path*)

**emit_url** (*self*, *document*, *url*)

**execute** (*self*)

#### RunLimitException

**class** aleph.crawlers.crawler.**RunLimitException**

##### Imports

- aleph.crawlers.crawler.CrawlerException

### aleph.crawlers.documentcloud

#### DocumentCloudCrawler

**class** aleph.crawlers.documentcloud.**DocumentCloudCrawler**

##### Imports

- <UNKNOWN>

**crawl** (*self*)

**crawl_document** (*self*, *document*)

#### SourceAfricaCrawler

**class** aleph.crawlers.documentcloud.**SourceAfricaCrawler**

#### Imports

- aleph.crawlers.documentcloud.DocumentCloudCrawler

## aleph.crawlers.schedule

### CrawlerSchedule

**class** aleph.crawlers.schedule.**CrawlerSchedule**

#### Imports

- object

**__init__** (*self*, *name*)

**__unicode__** (*self*)

**check_due** (*self*, *crawler_id*)

**to_dict** (*self*)

## aleph.crawlers.web

### AlephKrauler

**class** aleph.crawlers.web.**AlephKrauler**

#### Imports

- <UNKNOWN>

**__init__** (*self*, *crawler*)

**emit** (*self*, *page*)

### WebCrawler

**class** aleph.crawlers.web.**WebCrawler**

#### Imports

- <UNKNOWN>

**crawl** (*self*)

**emit** (*self*, *page*)

**get_content** (*self*, *page*)

## aleph.crawlers.webdav

### WebDAVCrawler

**class** aleph.crawlers.webdav.**WebDAVCrawler**

#### Imports

- <UNKNOWN>

**crawl** (*self*)

**crawl_collection** (*self*, *url*, *auth*)

**crawl_document** (*self*, *url*, *auth*)

## aleph.datasets

### Dataset

**class** aleph.datasets.**Dataset**

#### Imports

- object

#### Summary

A dataset describes one set of data to be loaded.

**__init__** (*self*, *name*, *data*)

**__repr__** (*self*)

**countries** (*self*)

**queries** (*self*)

**to_dict** (*self*)

### DatasetSet

**class** aleph.datasets.**DatasetSet**

#### Imports

- object

**__init__** (*self*, *datasets*)

**__iter__** (*self*)

**__repr__** (*self*)

**get** (*self*, *name*)

## aleph.datasets.formatting

### Formatter

**class** aleph.datasets.formatting.**Formatter**

#### Imports

- object

**__init__** (*self*, *template*)

**apply** (*self*, *record*)

## aleph.datasets.mapper

### EntityMapper

**class** aleph.datasets.mapper.**EntityMapper**

#### Imports

- aleph.datasets.mapper.Mapper

**__init__** (*self*, *query*, *name*, *data*)

**to_index** (*self*, *record*)

### LinkMapper

**class** aleph.datasets.mapper.**LinkMapper**

#### Imports

- aleph.datasets.mapper.Mapper

**__init__** (*self*, *query*, *data*)

**to_index** (*self*, *record*, *entities*)

### Mapper

**class** aleph.datasets.mapper.**Mapper**

#### Imports

- object

**__init__** (*self*, *query*, *data*)

**__repr__** (*self*)

**compute_key** (*self*, *record*)

**compute_properties** (*self*, *record*)

**refs** (*self*)

**to_index** (*self*, *record*)

### MapperProperty

**class** aleph.datasets.mapper.**MapperProperty**

#### Imports

- object

**__init__** (*self*, *mapper*, *name*, *data*, *schema*)

**__repr__** (*self*)

**get_values** (*self*, *record*)

## aleph.datasets.query

### CSVQuery

**class** aleph.datasets.query.**CSVQuery**

#### Imports

- aleph.datasets.query.Query

#### Summary

Special case for entity loading directly from a CSV URL

**__init__** (*self*, *dataset*, *data*)

**__repr__** (*self*)

**check_filters** (*self*, *data*)

**iterrows** (*self*)
Iterate through the table applying filters on-the-go.

**read_csv** (*self*, *csv_url*)

**read_local_csv** (*self*, *path*)

**read_remote_csv** (*self*, *csv_url*)

### DBQuery

**class** aleph.datasets.query.**DBQuery**

#### Imports

- aleph.datasets.query.Query

**__init__** (*self*, *dataset*, *data*)

**apply_filters** (*self*, *q*)

**compose_query** (*self*)

**engine** (*self*)

**from_clause** (*self*)

**get_column** (*self*, *ref*)

**get_table** (*self*, *ref*)

**iterrows** (*self*)
Compose the actual query and return an iterator of Record.

**mapped_columns** (*self*)
Determine which columns must be selected.

This will check entity and link mappings for the set of columns actually used in order to avoid loading superfluous data.

**meta** (*self*)

### Query

**class** aleph.datasets.query.**Query**

#### Imports

- object

#### Summary

A dataset describes one set of data to be loaded.

**__init__** (*self*, *dataset*, *data*)

**__repr__** (*self*)

**active_refs** (*self*)

#### QueryTable

**class** aleph.datasets.query.**QueryTable**

##### Imports

- object

##### Summary

A table to be joined in.

**__init__**(*self*, *query*, *data*)

**__repr__**(*self*)

## aleph.datasets.util

aleph.datasets.util.**finalize_index**(*data*, *schema*)
Apply final denormalisations to the index.

## aleph.default_settings

aleph.default_settings.**env_bool**(*name*)
Extract a boolean value from the environment consistently.

aleph.default_settings.**env_list**(*name*)
Extract a list of values from the environment consistently.

Multiple values are expected to be separated by a colon (':'), like in the UNIX $PATH variable.

## aleph.events

aleph.events.**log_event**(*request*)

aleph.events.**save_event**(*action*, *path*, *source_ip*, *query*, *data*, *role_id*)

## aleph.ext

aleph.ext.**get_analyzers**()

aleph.ext.**get_crawlers**()

aleph.ext.**get_extensions**(*section*)

aleph.ext.**get_ingestors**()

aleph.ext.**get_init**()

## aleph.index

## aleph.index.admin

aleph.index.admin.**delete_doc_type**(*doc_type*)

aleph.index.admin.**delete_index**()

aleph.index.admin.**flush_index**()
Run a refresh to apply all indexing changes.

aleph.index.admin.**init_search**()

aleph.index.admin.**upgrade_search**()
Add any missing properties to the index mappings.

## aleph.index.collections

aleph.index.collections.**delete_collection**(*collection_id*)
Delete all documents from a particular collection.

## aleph.index.datasets

aleph.index.datasets.**_index_updates**(*entities*, *links*)
Look up existing index documents and generate an updated form.

This is necessary to make the index accumulative, i.e. if an entity or link gets indexed twice with different field values, it'll add up the different field values into a single record. This is to avoid overwriting the document and losing field values. An alternative solution would be to implement this in Groovy on the ES.

aleph.index.datasets.**delete_dataset**(*dataset_name*)
Delete all entries from a particular dataset.

aleph.index.datasets.**index_items**(*entities*, *links*)
Index a set of links or entities.

## aleph.index.documents

aleph.index.documents.**delete_document**(*document_id*)

aleph.index.documents.**index_document**(*document*)

aleph.index.documents.**index_document_id**(*document_id*)

## aleph.index.entities

aleph.index.entities.**delete_collection_entities**()

aleph.index.entities.**delete_entity**(*entity_id*)
Delete an entity from the index.

aleph.index.entities.**delete_entity_references**(*entity_id*)
Delete all entities associated with any collection.

This is used by the `indexentities` management command in order to clear out any leftover entities in the index.

aleph.index.entities.**document_updates**(*q*, *entity_id*)

aleph.index.entities.**get_count**(*entity*)
Inaccurate, as it does not reflect auth.

aleph.index.entities.**index_entity**(*entity*)
Index an entity.

aleph.index.entities.**update_entity_references**(*entity*)
Same as above but runs in bulk for a particular entity.

## aleph.index.leads

aleph.index.leads.**delete_entity_leads**(*entity_id*)
Delete all entity-related leads from the index.

aleph.index.leads.**index_lead**(*lead*)
Index a lead.

## aleph.index.mapping

## aleph.index.records

aleph.index.records.**clear_records**(*document_id*)
Delete all records associated with the given document.

aleph.index.records.**generate_records**(*document*)
Generate index records, based on document rows or pages.

aleph.index.records.**index_records**(*document*)

## aleph.index.util

aleph.index.util.**bulk_op**(*iter*)

aleph.index.util.**merge_docs**(*old*, *new*)
Exend the values of the new doc with extra values from the old.

aleph.index.util.**query_delete**(*query*)
Delete all documents matching the given query inside the doc_type(s).

aleph.index.util.**remove_nulls**(*data*)
Remove None-valued keys from a dictionary, recursively.

## aleph.ingest

aleph.ingest.**get_manager**()
Get an ingestor manager, as a singleton instance.

aleph.ingest.**ingest**(*document_id*)
Process a given document by extracting its contents. This may include creating or updating child documents.

aleph.ingest.**ingest_document**(*document*, *file_path*)
Given a stub document and file path, extract information. This does not attempt to infer metadata such as a file name.

aleph.ingest.**ingest_url**(*self*, *document_id*, *url*)
Load the given URL into the document specified by document_id.

aleph.ingest.**reingest_collection**(*collection*)

## aleph.ingest.manager

### DocumentManager

**class** aleph.ingest.manager.**DocumentManager**

#### Imports

•<UNKNOWN>

#### Summary

Handle the process of ingesting documents.

This includes creating and flushing records, setting document state and dispatching child ingestors as needed.

**__init__**(*self*, *config*, *archive*)

**after**(*self*, *result*)

**before**(*self*, *result*)

**get_cache**(*self*, *key*)

**handle_child**(*self*, *parent*, *file_path*)

**ingest_document**(*self*, *document*)
Ingest a database-backed document.

First retrieve it's data and then call the actual ingestor.

**set_cache**(*self*, *key*, *value*)

## aleph.ingest.result

### DocumentResult

**class** aleph.ingest.result.**DocumentResult**

#### Imports

•<UNKNOWN>

#### Summary

Wrapper to link a Document to an ingestor result object.

**__init__**(*self*, *manager*, *document*)

**_emit_iterator_rows**(*self*, *iterator*)

**emit_page**(*self*, *index*, *text*)
Emit a plain text page.

**emit_pdf_alternative**(*self*, *file_path*)

**emit_rows**(*self*, *iterator*)
Emit rows of a tabular iterator.

**update**(*self*)
Apply the outcome of the result to the document.

## aleph.logic

#### Summary

Aleph logic layer.

This package contains high-level business logic functions for parts of aleph. This is stuff that would ordinarily live in the model but that also depends on components (like ES search) which in turn themselves depend on the model.

## aleph.logic.alerts

aleph.logic.alerts.**check_alerts**()
Go through all users and execute their alerts.

aleph.logic.alerts.**check_role_alerts**(*authz*)

aleph.logic.alerts.**format_results**(*alert*, *results*)

## aleph.logic.collections

aleph.logic.collections.**analyze_collection**(*collection_id*)
Re-analyze the elements of this collection, documents and entities.

aleph.logic.collections.**delete_collection**(*collection_id*)

aleph.logic.collections.**update_collection**(*collection*)
Create or update a collection.

## aleph.logic.datasets

aleph.logic.datasets.**load_dataset**(*dataset*)
Index all the entities and links in a given dataset.

aleph.logic.datasets.**load_rows**(*dataset*, *query*, *rows*)
Load a single batch of QUEUE_PAGE rows from the given query.

## aleph.logic.distance

aleph.logic.distance.**entity_distance**(*entity*, *other*)

aleph.logic.distance.**pred_best_jw**(*a*, *b*, *field*)
Find the closest jaro-winkler match.

aleph.logic.distance.**pred_matching_elem**(*a*, *b*, *field*)
Find the closest jaro-winkler match.

aleph.logic.distance.**pred_token_overlap**(*a*, *b*, *field*)
Find the closest jaro-winkler match.

## aleph.logic.documents

aleph.logic.documents.**delete_document**(*document*)

aleph.logic.documents.**update_document**(*document*)

## aleph.logic.entities

aleph.logic.entities.**combined_entity**(*entity*)
Use EntityIdentity mappings to construct a combined model of the entity with all data applied.

aleph.logic.entities.**delete_entity**(*entity*)

aleph.logic.entities.**delete_pending**()
Deletes any pending entities.

aleph.logic.entities.**fetch_entity**(*entity_id*)
Load entities from both the ES index and the database.

aleph.logic.entities.**generate_entity_references**(*entity*)

aleph.logic.entities.**reindex_entities**()

aleph.logic.entities.**reindex_entity**(*entity*)

aleph.logic.entities.**update_entity**(*entity*)

aleph.logic.entities.**update_entity_full**(*entity_id*)
Perform update operations on entities.

## aleph.logic.leads

aleph.logic.leads.**generate_leads**(*entity_id*)
Compute likely duplicates of a given entity and index these leads.

aleph.logic.leads.**update_lead**(*entity*, *match*, *judgement*)

## aleph.logic.permissions

aleph.logic.permissions.**update_permission**(*role*, *collection*, *read*, *write*)
Update a roles permission to access a given collection.

## aleph.manage

aleph.manage.**alerts**()
Generate alert notifications.

aleph.manage.**analyze**(*foreign_id*)
Re-analyze documents in the given collection (or throughout).

aleph.manage.**collections**()
List all collections.

aleph.manage.**crawl**(*name*)
Execute the given crawler.

aleph.manage.**crawldir**(*directory*)
Crawl the given directory.

aleph.manage.**deletedataset**(*name*)

aleph.manage.**deletepending**()
Deletes any pending entities and related items.

aleph.manage.**evilshit**()
EVIL: Delete all data and recreate the database.

aleph.manage.**flush**(*foreign_id*)
Reset the crawler state for a given collecton.

aleph.manage.**index**()
Index documents in the given collection (or throughout).

aleph.manage.**indexentities**()
Re-index all the entities.

aleph.manage.**init**()
Create or upgrade the search index and database.

aleph.manage.**installdata**()
Create or upgrade the search index and database.

aleph.manage.**loaddataset**(*name*)
Index all the entities in a given dataset.

aleph.manage.**main**()

aleph.manage.**reingest**(*foreign_id*)
Re-ingest documents in the given collection.

aleph.manage.**resetindex**()
Re-create the ES index configuration, dropping all data.

aleph.manage.**updateentities**()
Re-index all the entities.

aleph.manage.**upgrade**()
Create or upgrade the search index and database.

## aleph.model

aleph.model.**create_system_roles**()

aleph.model.**upgrade_db**()

## aleph.model.alert

### Alert

**class** aleph.model.alert.**Alert**

#### Imports

- <UNKNOWN>
- <UNKNOWN>

#### Summary

A subscription to notifications on a given query.

**by_role** (*cls*, *role*)

**create** (*cls*, *data*, *role*)

**exists** (*cls*, *query*, *role*)

**by_id** (*cls*, *id*)

**dedupe** (*cls*, *entity_id*)

**__repr__** (*self*)

**delete** (*self*)

**is_same** (*self*, *other*)

**label** (*self*)

**to_dict** (*self*)

**to_query** (*self*)

**update** (*self*)

## aleph.model.cache

### Cache

**class** aleph.model.cache.**Cache**

#### Imports

- <UNKNOWN>

### Summary

Store OCR computation results.

**get_cache**(*cls*, *key*)

**set_cache**(*cls*, *key*, *value*)

**__repr__**(*self*)

**__unicode__**(*self*)

## aleph.model.collection

### Collection

**class** aleph.model.collection.**Collection**

### Imports

- <UNKNOWN>
- <UNKNOWN>
- <UNKNOWN>
- <UNKNOWN>

**find**(*cls*)

**create**(*cls*, *data*)

**by_foreign_id**(*cls*, *foreign_id*)

**__repr__**(*self*)

**__unicode__**(*self*)

**get_document_count**(*self*)

**get_entity_count**(*self*)

**is_public**(*self*)

**pending_entities**(*self*)
Generate a ranked list of the most commonly used pending entities. This is used for entity review.

**roles**(*self*)

**to_dict**(*self*)

**touch**(*self*)

**update**(*self*, *data*)

## aleph.model.common

aleph.model.common.**make_textid**()

aleph.model.common.**merge_data**(*base*, *merge*)
Merge two objects such that values in base are kept and updated only if merge has additional info.

aleph.model.common.**object_key**(*obj*)
Generate a checksum for a nested object or list.

### DatedModel

**class** aleph.model.common.**DatedModel**

#### Imports

•object

**all**(*cls*)

**by_id**(*cls*, *id*)

**all_ids**(*cls*)

**all_by_ids**(*cls*, *ids*)

**delete**(*self*)

**to_dict**(*self*)

### IdModel

**class** aleph.model.common.**IdModel**

#### Imports

•object

**to_dict**(*self*)

### ModelFacets

**class** aleph.model.common.**ModelFacets**

#### Imports

•object

**facet_by**(*cls*, *q*, *field*)

### SoftDeleteModel

**class** aleph.model.common.**SoftDeleteModel**

#### Imports

- aleph.model.common.DatedModel

**all** (*cls*)

**all_ids** (*cls*)

**delete** (*self*)

**to_dict** (*self*)

### UuidModel

**class** aleph.model.common.**UuidModel**

#### Imports

- object

**to_dict** (*self*)

## aleph.model.document

### Document

**class** aleph.model.document.**Document**

#### Imports

- <UNKNOWN>
- <UNKNOWN>
- <UNKNOWN>

**is_crawler_active** (*cls*, *crawler_id*)

**by_keys** (*cls*)
Try and find a document by various criteria.

**crawler_stats** (*cls*, *crawler_id*)

**crawler_last_run** (*cls*, *crawler_id*)

**__init__** (*self*)

**__repr__** (*self*)

**delete** (*self*)

**delete_records** (*self*)

**delete_references** (*self*)

**delete_tags** (*self*)

**insert_records** (*self*, *sheet*, *iterable*)

**text_parts** (*self*)

**to_dict** (*self*)

**to_index_dict** (*self*)

**update** (*self*, *data*)

**update_meta** (*self*)

## aleph.model.document_record

### DocumentRecord

**class** aleph.model.document_record.**DocumentRecord**

#### Imports

- <UNKNOWN>

#### Summary

A record reflects a row or page of a document.

**find_records** (*cls*, *document_id*, *ids*)

**__repr__** (*self*)

**text_parts** (*self*)
Utility method to get all text snippets in a record.

**to_dict** (*self*)

## aleph.model.document_tag

### DocumentTag

**class** aleph.model.document_tag.**DocumentTag**

#### Imports

- <UNKNOWN>
- <UNKNOWN>

#### Summary

A record reflects an entity or tag extracted from a document.

**delete_by** (*cls*)

**__repr__** (*self*)

### DocumentTagCollector

**class** aleph.model.document_tag.**DocumentTagCollector**

#### Imports

- object

#### Summary

Utility class to collect and aggregate tags from a particular process.

This is useful when many tags about the same documented are emitted by a particular source.

**__init__**(*self*, *document*, *origin*)

**__len__**(*self*)

**emit**(*self*, *text*, *type*)
Create a tag, this can be called multiple times with the same text.

**save**(*self*)
Flush all existing tags from this origin and store new ones.

## aleph.model.entity

### Entity

**class** aleph.model.entity.**Entity**

#### Imports

- <UNKNOWN>
- <UNKNOWN>
- <UNKNOWN>

**filter_collections**(*cls*, *q*)

**by_id_set**(*cls*, *ids*)

**delete_dangling**(*cls*, *collection_id*)
Delete dangling entities.

Entities can dangle in pending state while they have no references pointing to them, thus making it impossible to enable them. This is a routine cleanup function.

**save**(*cls*, *data*, *collection*)

**by_foreign_id**(*cls*, *foreign_id*, *collection_id*)

**latest**(*cls*)

**__repr__**(*self*)

**__unicode__**(*self*)

**delete** (*self*)

**delete_identities** (*self*)

**delete_references** (*self*)

**merge** (*self*, *other*)

**regex_terms** (*self*)

**schema** (*self*)

**terms** (*self*)

**to_dict** (*self*)

**to_index** (*self*)

**to_ref** (*self*)

**update** (*self*, *entity*)

## aleph.model.entity_identity

### EntityIdentity

**class** aleph.model.entity_identity.**EntityIdentity**

#### Imports

- <UNKNOWN>
- <UNKNOWN>
- <UNKNOWN>

**judgements_by_entity** (*cls*, *entity_id*)

**by_entity_match** (*cls*, *entity_id*, *match_id*)

**entity_ids** (*cls*, *entity_id*)

**save** (*cls*, *entity_id*, *match_id*, *judgement*)

**__repr__** (*self*)

## aleph.model.event_log

### EventLog

**class** aleph.model.event_log.**EventLog**

#### Imports

- <UNKNOWN>
- <UNKNOWN>
- <UNKNOWN>

**emit** (*cls*, *action*, *path*)

**__repr__** (*self*)

**__unicode__** (*self*)

## aleph.model.metadata

### Metadata

**class** aleph.model.metadata.**Metadata**

#### Imports

- object

#### Summary

Handle all sorts of metadata normalization for documents.

**__init__** (*self*)

**add_country** (*self*, *country*)

**add_date** (*self*, *obj*)

**add_domain** (*self*, *domain*)

**add_email** (*self*, *email*)

**add_keyword** (*self*, *kw*)

**add_language** (*self*, *language*)

**add_phone_number** (*self*, *number*)

**add_url** (*self*, *url*)

**author** (*self*, *author*)

**columns** (*self*, *columns*)

**countries** (*self*, *countries*)

**dates** (*self*, *dates*)

**domains** (*self*, *domains*)

**emails** (*self*, *emails*)

**encoding** (*self*, *encoding*)

**extension** (*self*, *extension*)

**file_name** (*self*, *file_name*)

**file_size** (*self*, *file_size*)

**file_title** (*self*)

The file title is a human-readable interpretation of the file name. It is used for labelling or as a backup title. It should not be used to generate an actual file system path.

**has_meta** (*self*, *name*)

**headers** (*self*, *headers*)

**keywords** (*self*, *keywords*)

**languages** (*self*, *languages*)

**mime_type** (*self*, *mime_type*)

**pdf_version** (*self*, *pdf_version*)

**phone_numbers** (*self*, *phone_numbers*)

**source_url** (*self*, *source_url*)

**summary** (*self*, *summary*)

**tables** (*self*, *tables*)

**title** (*self*, *title*)

**to_meta_dict** (*self*)
Generate ElasticSearch form.

**update_meta** (*self*)

**urls** (*self*, *urls*)

## aleph.model.permission

### Permission

**class** aleph.model.permission.**Permission**

#### Imports

- <UNKNOWN>
- <UNKNOWN>
- <UNKNOWN>

#### Summary

A set of rights granted to a role on a resource.

**grant_collection** (*cls*, *collection_id*, *role*, *read*, *write*)

**grant_foreign** (*cls*, *collection*, *foreign_id*, *read*, *write*)

**by_collection_role** (*cls*, *collection_id*, *role*)

**to_dict** (*self*)

## aleph.model.reference

### Reference

**class** aleph.model.reference.**Reference**

#### Imports

- <UNKNOWN>
- <UNKNOWN>
- <UNKNOWN>

**index_references**(*cls*, *document_id*)
Helper function to get reference data for indexing.

**__repr__**(*self*)

**to_dict**(*self*)

## aleph.model.role

### Role

**class** aleph.model.role.**Role**

#### Imports

- <UNKNOWN>
- <UNKNOWN>
- <UNKNOWN>

#### Summary

A user, group or other access control subject.

**by_api_key**(*cls*, *api_key*)

**load_id**(*cls*, *foreign_id*)
Load a role and return the ID.

If type is given and no role is found, a new role will be created.

**by_prefix**(*cls*, *prefix*)
Load a list of roles matching a name, email address, or foreign_id.

**Parameters** **pattern** (*str*) – Pattern to match.

**by_email**(*cls*, *email*)

**notifiable**(*cls*)

**load_or_create**(*cls*, *foreign_id*, *type*, *name*)

**by_foreign_id**(*cls*, *foreign_id*)

**all_groups**(*cls*)

**authenticate_using_ldap**(*cls*, *identifier*, *password*)
Autheticates using user LDAP identifier and password.

**Parameters**

- **identifier** (*str*) – LDAP ID.
- **password** (*str*) – LDAP password.

**Returns** A matched role.

**Return type** *Role*

**__repr__** (*self*)

**__unicode__** (*self*)

**add_role** (*self*, *role*)
Adds an existing role as a membership of a group.

**check_password** (*self*, *secret*)
Checks the password if it matches the role password hash.

**Parameters** **secret** (*str*) – The password to be checked.

**Return type** bool

**clear_roles** (*self*)
Removes any existing roles from group membership.

**set_password** (*self*, *secret*)
Hashes and sets the role password.

**Parameters** **secret** (*str*) – The password to be set.

**to_dict** (*self*)

**update** (*self*, *data*)

## aleph.model.tabular

### Tabular

**class** aleph.model.tabular.**Tabular**

#### Imports

•object

**__init__** (*self*)

**__repr__** (*self*)

**add_column** (*self*, *label*)

**columns** (*self*)

**sheet** (*self*)

**sheet_name** (*self*)

**to_dict** (*self*)

### TabularColumn

**class** aleph.model.tabular.**TabularColumn**

#### Imports

- object

**__init__**(*self*, *schema*, *data*)

**__repr__**(*self*)

## aleph.model.validate

aleph.model.validate.**is_collection_category**(*cat*)

aleph.model.validate.**validate**(*data*, *schema*)

## aleph.notify

aleph.notify.**notify_role**(*role*, *subject*, *html*)
Send an email to a user with a given address.

## aleph.oauth

aleph.oauth.**configure_oauth**(*app*)

aleph.oauth.**get_oauth_token**()

aleph.oauth.**handle_facebook_oauth**(*sender*)

aleph.oauth.**handle_google_oauth**(*sender*)

aleph.oauth.**handle_keycloak_oauth**(*sender*)

aleph.oauth.**setup_providers**(*app*)

## aleph.queues

## aleph.schema

### Schema

**class** aleph.schema.**Schema**

#### Imports

- object

#### Summary

Defines the abstract data model.

Schema items define the entities and links available in the model.

**__init__**(*self*, *schemata*, *section*, *name*, *data*)

**__repr__** (*self*)

**extends** (*self*)
Return the inherited schemata.

**get** (*self*, *name*)

**properties** (*self*)
Return properties, those defined locally and in ancestors.

**schemata** (*self*)
Return the full inheritance chain.

**to_dict** (*self*)

**validate** (*self*, *data*)
Validate a dataset against the given schema.

This will also drop keys which are not present as properties.

### SchemaProperty

**class** aleph.schema.**SchemaProperty**

#### Imports

- object

**__init__** (*self*, *schema*, *name*, *data*)

**__repr__** (*self*)

**to_dict** (*self*)

**validate** (*self*, *data*)
Validate that the data should be stored.

Since the types system doesn't really have validation, this currently tries to normalize the value to see if it passes strict parsing.

### SchemaSet

**class** aleph.schema.**SchemaSet**

#### Imports

- object

#### Summary

A collection of schemata.

**__init__** (*self*, *data*)

**__iter__** (*self*)

**__repr__** (*self*)

**get** (*self*, *name*)

**merge_entity_schema** (*self*, *left*, *right*)
Select the most narrow of two schemata.

When indexing data from a dataset, an entity may be declared as a LegalEntity in one query, and as a Person in another. This function will select the most specific of two schemata offered. In the example, that would be Person.

**to_dict** (*self*)

### SchemaValidationException

**class** aleph.schema.**SchemaValidationException**

#### Imports

- exceptions.Exception

#### Summary

Schema validation errors will be caught by the API.

**__init__** (*self*, *errors*)

## aleph.schema.types

aleph.schema.types.**resolve_type** (*name*)
Look up a configerty type by name.

### AddressProperty

**class** aleph.schema.types.**AddressProperty**

#### Imports

- aleph.schema.types.StringProperty

**normalize_value** (*self*, *value*)

### CountryProperty

**class** aleph.schema.types.**CountryProperty**

#### Imports

- aleph.schema.types.StringProperty

**clean** (*self*, *value*, *record*, *config*)

**normalize_value** (*self*, *value*)

### DateProperty

**class** aleph.schema.types.**DateProperty**

#### Imports

- aleph.schema.types.StringProperty

**clean** (*self*, *value*, *record*, *config*)

**normalize_value** (*self*, *value*)

### EmailProperty

**class** aleph.schema.types.**EmailProperty**

#### Imports

- aleph.schema.types.StringProperty

**clean** (*self*, *value*, *record*, *config*)

**normalize_value** (*self*, *value*)

### IdentiferProperty

**class** aleph.schema.types.**IdentiferProperty**

#### Imports

- aleph.schema.types.StringProperty

**normalize_value** (*self*, *value*)

### NameProperty

**class** aleph.schema.types.**NameProperty**

#### Imports

- aleph.schema.types.StringProperty

**fingerprint** (*self*, *values*)

**normalize_value** (*self*, *value*)

### PhoneProperty

**class** aleph.schema.types.**PhoneProperty**

#### Imports

- aleph.schema.types.StringProperty

**clean** (*self*, *value*, *record*, *config*)

### StringProperty

**class** aleph.schema.types.**StringProperty**

#### Imports

- object

**__init__** (*self*)

**clean** (*self*, *value*, *record*, *config*)

**fingerprint** (*self*, *values*)

**normalize** (*self*, *values*)

**normalize_value** (*self*, *value*)

### URLProperty

**class** aleph.schema.types.**URLProperty**

#### Imports

- aleph.schema.types.StringProperty

## aleph.search

## aleph.search.documents

aleph.search.documents.**documents_iter**(*state*)
Iterate over a set of documents based on a query state.

aleph.search.documents.**documents_query**(*state*)
Parse a user query string, compose and execute a query.

aleph.search.documents.**entity_documents**(*entity*, *state*)
Try and find all documents mentioning a particular entity.

aleph.search.documents.**facet_collections**(*state*, *q*, *aggs*)

aleph.search.documents.**facet_entities**(*state*, *aggs*)
Filter entities, facet for collections.

## aleph.search.entities

aleph.search.entities.**entities_query**(*state*)
Parse a user query string, compose and execute a query.

aleph.search.entities.**facet_collections**(*state*, *q*, *aggs*)

aleph.search.entities.**get_dataset_countries**(*dataset_name*)
Create a list of the top 300 countries mentioned in a dataset.

aleph.search.entities.**load_entity**(*entity_id*)
Load a single entity by ID.

aleph.search.entities.**similar_entities**(*entity*, *state*)
Merge suggestions API.

aleph.search.entities.**suggest_entities**(*prefix*, *authz*)
Auto-complete API.

## aleph.search.facet

aleph.search.facet.**parse_facet_result**(*state*, *result*)

### CollectionFacet

**class** aleph.search.facet.**CollectionFacet**

#### Imports

- aleph.search.facet.Facet

**expand**(*self*, *keys*)

**get_data**(*self*)

**get_values**(*self*)

### CountryFacet

**class** aleph.search.facet.**CountryFacet**

#### Imports

- aleph.search.facet.Facet

**expand**(*self*, *keys*)

### DatasetFacet

**class** aleph.search.facet.**DatasetFacet**

#### Imports

- aleph.search.facet.Facet

**expand**(*self*, *keys*)

### EntityFacet

**class** aleph.search.facet.**EntityFacet**

#### Imports

- aleph.search.facet.Facet

**expand**(*self*, *keys*)

**get_data**(*self*)

**get_values**(*self*)

### Facet

**class** aleph.search.facet.**Facet**

#### Imports

- object

**__init__**(*self*, *state*, *name*, *aggs*)

**expand**(*self*, *keys*)

**get_data**(*self*)

**get_values**(*self*)

**to_dict**(*self*)

### LanguageFacet

**class** aleph.search.facet.**LanguageFacet**

#### Imports

- aleph.search.facet.Facet

**expand**(*self*, *keys*)

### SchemaFacet

**class** aleph.search.facet.**SchemaFacet**

#### Imports

- aleph.search.facet.Facet

**expand**(*self*, *keys*)

## aleph.search.fragments

aleph.search.fragments.**aggregate**(*state*, *q*, *aggs*, *facets*)
Generate aggregations, a generalized way to do facetting.

aleph.search.fragments.**authz_filter**(*q*, *authz*)

aleph.search.fragments.**filter_query**(*q*, *filters*)
Apply a list of filters to the given query.

aleph.search.fragments.**match_all**()

aleph.search.fragments.**multi_match**(*text*, *fields*)

aleph.search.fragments.**phrase_match**(*text*, *field*)

aleph.search.fragments.**text_query**(*text*)
Part of a query which finds a piece of text.

aleph.search.fragments.**text_query_string**(*text*)

## aleph.search.leads

aleph.search.leads.**lead_count**(*collection_id*)
Inaccurate, as it does not reflect auth.

aleph.search.leads.**leads_query**(*collection_id*, *state*)

## aleph.search.links

aleph.search.links.**links_query**(*origin*, *state*)
Parse a user query string, compose and execute a query.

## aleph.search.query

### QueryState

**class** aleph.search.query.**QueryState**

#### Imports

- object

#### Summary

Hold state for common query parameters.

**__init__**(*self*, *args*, *authz*)

**collection_id**(*self*)
Return the set of collection IDs to be queried.

**entities**(*self*)

**entity_terms**(*self*)

**facet_size**(*self*)

**filter_items**(*self*)

**filters**(*self*)

**get**(*self*, *name*)

**get_filters**(*self*, *field*)

**getbool**(*self*, *name*)

**getfilter**(*self*, *name*)

**getint**(*self*, *name*)

**getlist**(*self*, *name*)

**has_query**(*self*)

**has_text**(*self*)

**highlight_terms**(*self*)

**items**(*self*)

**limit**(*self*)

**offset**(*self*)

**page**(*self*)

**sort**(*self*)

**text**(*self*)

## aleph.search.records

aleph.search.records.**execute_records_query**(*document_id*, *state*, *query*)
Execute a query against records and return a set of results.

aleph.search.records.**records_query**(*document_id*, *state*)

aleph.search.records.**records_query_internal**(*document_id*, *shoulds*)

aleph.search.records.**records_query_shoulds**(*state*)

aleph.search.records.**scan_entity_mentions**(*entity*)
Find mentions of a given entity in all records.

## aleph.search.util

aleph.search.util.**add_filter**(*q*, *filter_*)
Add the given filter `filter_` to the given query.

aleph.search.util.**clean_highlight**(*hlt*)

aleph.search.util.**execute_basic**(*doc_type*, *query*)
Common part of running a particular query.

aleph.search.util.**next_params**(*args*, *result*)
Get the parameters for making a next link.

aleph.search.util.**scan_iter**(*query*, *type*)
Scan the results of a query. No pagination is applied.

## aleph.signals

## aleph.tests

## aleph.tests.factories

## aleph.tests.factories.models

## aleph.tests.factories.models.collection

### CollectionFactory

**class** aleph.tests.factories.models.collection.**CollectionFactory**

#### Imports

- <UNKNOWN>

**class Meta**

## aleph.tests.factories.models.entity

### EntityFactory

**class** aleph.tests.factories.models.entity.**EntityFactory**

#### Imports

- <UNKNOWN>

**class Meta**

## aleph.tests.factories.models.role

### RoleFactory

**class** aleph.tests.factories.models.role.**RoleFactory**

#### Imports

- <UNKNOWN>

**class Meta**

## aleph.tests.util

### TestCase

**class** aleph.tests.util.**TestCase**

#### Imports

- <UNKNOWN>

**create_app**(*self*)

**create_user**(*self*)

**get_fixture_path**(*self*, *file_name*)

**load_fixtures**(*self*, *file_name*)

**login**(*self*)

**setUp**(*self*)

**tearDown**(*self*)

## aleph.text

aleph.text.**encoded_value**(*text*)

aleph.text.**has_value**(*value*)
Check a given value is not empty.

aleph.text.**index_form**(*texts*)
Turn a set of strings into the appropriate form for indexing.

aleph.text.**match_form**(*text*)
Turn a string into a form appropriate for name matching.

The goal of this function is not to retain a readable version of the given string, but rather to yield a normalised version suitable for comparisons and machine analysis.

aleph.text.**string_value**(*value*)

## aleph.util

aleph.util.**checksum**(*filename*)
Generate a hash for a given file name.

aleph.util.**dict_list**(*data*)
Get an entry as a list from a dict. Provide a fallback key.

aleph.util.**ensure_list**(*obj*)
Make the returned object a list, otherwise wrap as single item.

aleph.util.**is_list**(*obj*)

aleph.util.**load_config_file**(*file_path*)
Load a YAML (or JSON) graph model configuration file.

aleph.util.**make_tempdir**()

aleph.util.**make_tempfile**()

aleph.util.**remove_tempdir**(*dirpath*)

aleph.util.**remove_tempfile**(*filepath*)

aleph.util.**resolve_includes**(*file_path*, *data*)
Handle include statements in the graph configuration file.

This allows the YAML graph configuration to be broken into multiple smaller fragments that are easier to maintain.

aleph.util.**unique_list**(*lst*)
Make a list unique, retaining order of initial appearance.

### SessionTask

**class** aleph.util.**SessionTask**

#### Imports

- <UNKNOWN>

**on_failure**(*self*, *exc*, *task_id*, *args*, *kwargs*, *einfo*)

## aleph.views

aleph.views.**mount_app_blueprints**(*app*)

## aleph.views.alerts_api

aleph.views.alerts_api.**create**()

aleph.views.alerts_api.**delete**(*id*)

aleph.views.alerts_api.**index**()

aleph.views.alerts_api.**view**(*id*)

## aleph.views.base_api

aleph.views.base_api.**angular_templates**()

aleph.views.base_api.**handle_authz_error**(*err*)

aleph.views.base_api.**handle_es_error**(*err*)

aleph.views.base_api.**handle_schema_validation_error**(*err*)

aleph.views.base_api.**handle_validation_error**(*err*)

aleph.views.base_api.**metadata**()

aleph.views.base_api.**statistics**()

aleph.views.base_api.**ui**()

## aleph.views.cache

aleph.views.cache.**cache_response**(*resp*)

aleph.views.cache.**enable_cache**()

aleph.views.cache.**handle_not_modified**(*exc*)

aleph.views.cache.**setup_caching**()

### NotModified

**class** aleph.views.cache.**NotModified**

#### Imports

- exceptions.Exception

### aleph.views.collections_api

aleph.views.collections_api.**create**()

aleph.views.collections_api.**delete**(*id*)

aleph.views.collections_api.**index**()

aleph.views.collections_api.**pending**(*id*)

aleph.views.collections_api.**process**(*id*)

aleph.views.collections_api.**update**(*id*)

aleph.views.collections_api.**view**(*id*)

### aleph.views.crawlers_api

aleph.views.crawlers_api.**index**()

aleph.views.crawlers_api.**queue**()

### aleph.views.datasets_api

aleph.views.datasets_api.**index**()

aleph.views.datasets_api.**view**(*name*)

### aleph.views.documents_api

aleph.views.documents_api.**file**(*document_id*)

aleph.views.documents_api.**index**()

aleph.views.documents_api.**pdf**(*document_id*)

aleph.views.documents_api.**record**(*document_id*, *index*)

aleph.views.documents_api.**records**(*document_id*)

aleph.views.documents_api.**references**(*document_id*)

aleph.views.documents_api.**table**(*document_id*, *table_id*)

aleph.views.documents_api.**update**(*document_id*)

aleph.views.documents_api.**view**(*document_id*)

### aleph.views.entities_api

aleph.views.entities_api.**all**()

aleph.views.entities_api.**create**()

aleph.views.entities_api.**delete**(*id*)

aleph.views.entities_api.**documents**(*id*)

aleph.views.entities_api.**index**()

aleph.views.entities_api.**links**(*id*)

aleph.views.entities_api.**merge**(*id*, *other_id*)

aleph.views.entities_api.**similar**(*id*)

aleph.views.entities_api.**suggest**()

aleph.views.entities_api.**update**(*id*)

aleph.views.entities_api.**view**(*id*)

## aleph.views.exports_api

aleph.views.exports_api.**export**()

aleph.views.exports_api.**get_results**(*state*, *limit*)

## aleph.views.ingest_api

aleph.views.ingest_api.**ingest_upload**(*collection_id*)

## aleph.views.leads_api

aleph.views.leads_api.**index**(*collection_id*)

aleph.views.leads_api.**update**(*collection_id*)

## aleph.views.reconcile_api

aleph.views.reconcile_api.**entity_link**(*id*)

aleph.views.reconcile_api.**get_freebase_types**()

aleph.views.reconcile_api.**reconcile**()
Reconciliation API, emulates Google Refine API.

See: http://code.google.com/p/google-refine/wiki/ReconciliationServiceApi

aleph.views.reconcile_api.**reconcile_index**()

aleph.views.reconcile_api.**reconcile_op**(*query*)
Reconcile operation for a single query.

aleph.views.reconcile_api.**suggest_entity**()
Suggest API, emulates Google Refine API.

aleph.views.reconcile_api.**suggest_property**()

aleph.views.reconcile_api.**suggest_type**()

## aleph.views.roles_api

aleph.views.roles_api.**check_visible**(*role*)
Users should not see group roles which they are not a part of.

aleph.views.roles_api.**create**()

aleph.views.roles_api.**invite_email**()

aleph.views.roles_api.**permissions_index**(*collection*)

aleph.views.roles_api.**permissions_update**(*collection*)

aleph.views.roles_api.**suggest**()

aleph.views.roles_api.**update**(*id*)

aleph.views.roles_api.**view**(*id*)

## aleph.views.search_api

aleph.views.search_api.**query**()

## aleph.views.sessions_api

aleph.views.sessions_api.**callback**(*provider*)

aleph.views.sessions_api.**load_role**()

aleph.views.sessions_api.**login**()

aleph.views.sessions_api.**logout**()

aleph.views.sessions_api.**password_login**()
Provides email and password authentication.

aleph.views.sessions_api.**status**()

## aleph.views.util

aleph.views.util.**extract_next_url**(*req*)
Extracts the URL/path to follow when redirects/unauthorization occurs.

**Parameters** **req** (*object*) – Flask request object to extract from.

**Returns** Path of the next target URL.

**Return type** str

aleph.views.util.**get_document**(*document_id*)

aleph.views.util.**get_entity**(*id*, *action*)

aleph.views.util.**is_safe_url**(*target*)

aleph.views.util.**make_excel**(*result_iter*, *fields*)

## a

## Symbols



## A

## B



## C

## D

## E

## F



## G

## H



## I



## L



## M

## N



## O



## P

## Q



## R

## S

## T



## U



## V



## W