

---

# **unicore.distribute Documentation**

***Release 1.0***

**Praekelt Foundation**

May 16, 2016



<b>1</b>	<b>Installation</b>	<b>3</b>
<b>2</b>	<b>Configuration</b>	<b>5</b>
2.1	Indexing . . . . .	5
2.2	Proxying . . . . .	5
<b>3</b>	<b>Running</b>	<b>7</b>
<b>4</b>	<b>Webhooks</b>	<b>9</b>
<b>5</b>	<b>Polling</b>	<b>11</b>
<b>6</b>	<b>Querying</b>	<b>13</b>
<b>7</b>	<b>URL structure</b>	<b>17</b>



unicore.distribute is a collection of APIs and tools for dealing with Universal Core content repositories.



---

# Installation

---

The recommended way to install this for development is to install it in a [virtualenv](#) but it's not necessary.

```
pip install uniconcore.distribute
```





---

## Configuration

---

Put the following in a file called `development.ini`

```
[app:main]
use = egg:unicore.distribute
repo.storage_path = repos/

[server:main]
use = egg:waitress#main
host = 0.0.0.0
port = 6543
```

### 2.1 Indexing

`unicore.distribute` can automatically index data on Elasticsearch. To enable this, add these options to the `[app:main]` section:

```
es.host = http://localhost:9200
es.indexing_enabled = true
```

### 2.2 Proxying

Use `unicore.distribute` as an Elasticsearch proxy by adding these options to the `[app:main]` section:

```
proxy.enabled = True
proxy.path = esapi
proxy.upstream = http://localhost:9200
```

For most use cases `es.host` and `proxy.upstream` should point to the same Elasticsearch service.



---

## Running

---

Clone a Universal Core content repository and run the server:

```
$ git https://github.com/smn/unicore-sample-content \
    repos/unicore-sample-content
$ pserve development.ini
$ curl http://localhost:6543/repos.json
```

It is also possible to clone a repository directly from the API:

```
$ curl -XPOST -H 'Content-Type: application/json' \
    -d '{"repo_url": "https://example.com/repo.git"}' \
    http://localhost:6543/repos.json
```

The repo will only be indexed if cloned via the API (and indexing is enabled). **Note that the repo name and index prefix are the same.** So in the two examples above the index prefixes are “unicore-sample-content” and “repo” respectively.

To use a different repo name, specify `repo_name`:

```
$ curl -XPOST -H 'Content-Type: application/json' \
    -d '{"repo_url": "https://example.com/repo.git", \
        "repo_name": "repo-foo"}' \
    http://localhost:6543/repos.json
```



---

## Webhooks

---

The application can notify you when it is notified of changes made to the upstream repository:

Make sure the lines in `development.ini` relating to `unicore.webhooks` are uncommented and then initialize the database:

```
$ alembic upgrade head
```

Now your database is configured and you can store Webhooks:

```
$ curl -XPOST \
  -H 'Content-Type: application/json' \
  -d '{"event_type": "repo.push", "url": "http://requestb.in/vystj5vy", "active": true}' \
  http://localhost:6543/hooks
{
  "uuid": "09b901ccc5094f1a89f8bd03165fe3d6",
  "owner": null,
  "url": "http://requestb.in/vystj5vy",
  "event_type": "repo.push",
  "active": true
}
```

---

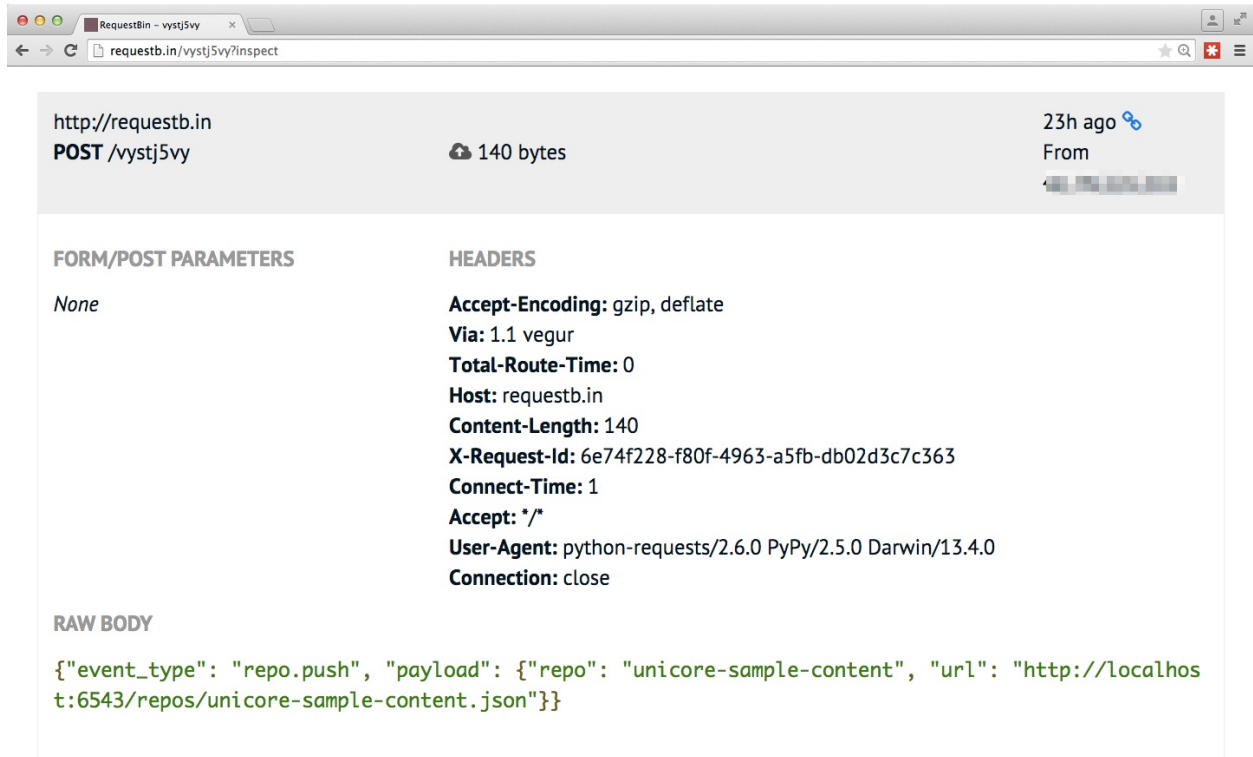
**Note:** Currently the only `event_type` supported is `repo.push`

---

Now if we notify the API of changes being made upstream (say via GitHub's webhooks) we will now relay that all webhooks registered:

```
$ curl -XPOST http://localhost:6543/repos/unicore-sample-content.json
```

Here is the request made to the registered URL with the JSON payload:



The screenshot shows a web browser window with the address bar displaying `requestb.in/vystj5vy?inspect`. The main content area shows details for a POST request to `http://requestb.in/vystj5vy`. The request size is 140 bytes, and it was received 23 hours ago. The page is divided into three sections: FORM/POST PARAMETERS, HEADERS, and RAW BODY.

**FORM/POST PARAMETERS**

None

**HEADERS**

**Accept-Encoding:** gzip, deflate  
**Via:** 1.1 vegur  
**Total-Route-Time:** 0  
**Host:** requestb.in  
**Content-Length:** 140  
**X-Request-Id:** 6e74f228-f80f-4963-a5fb-db02d3c7c363  
**Connect-Time:** 1  
**Accept:** \*/\*  
**User-Agent:** python-requests/2.6.0 PyPy/2.5.0 Darwin/13.4.0  
**Connection:** close

**RAW BODY**

```
{"event_type": "repo.push", "payload": {"repo": "unicore-sample-content", "url": "http://localhost:6543/repos/unicore-sample-content.json"}}
```

---

## Polling

---

Unicore.distribute ships with a command line program:

```
$ unicore.distribute --help
usage: unicore.distribute [-h] {poll-repositories} ...

unicore.distribute command line tools.

positional arguments:
  {poll-repositories}  Commands
  poll-repositories    poll repositories

optional arguments:
  -h, --help            show this help message and exit
```

The only feature currently available is one which can be used to poll repositories at a regular interval to see if new content has arrived. If that is the case then an event is fired and the registered webhook URLs are called:

```
$ unicore.distribute poll-repositories --help
usage: unicore.distribute poll-repositories [-h] [-d REPO_DIR] [-i INI_FILE]
                                             [-u BASE_URL]

optional arguments:
  -h, --help            show this help message and exit
  -d REPO_DIR, --repo-dir REPO_DIR
                        The directory with repositories.
  -i INI_FILE, --ini-file INI_FILE
                        The project's ini file.
  -u BASE_URL, --base-url BASE_URL
                        This server's public URL (for webhooks)
```

Hook up the poll-repositories sub-command to cron for regular polling:

```
*/15 * * * * unicore.distribute poll-repositories -d /var/praezelt/repos/ -i development.ini -u http
```





---

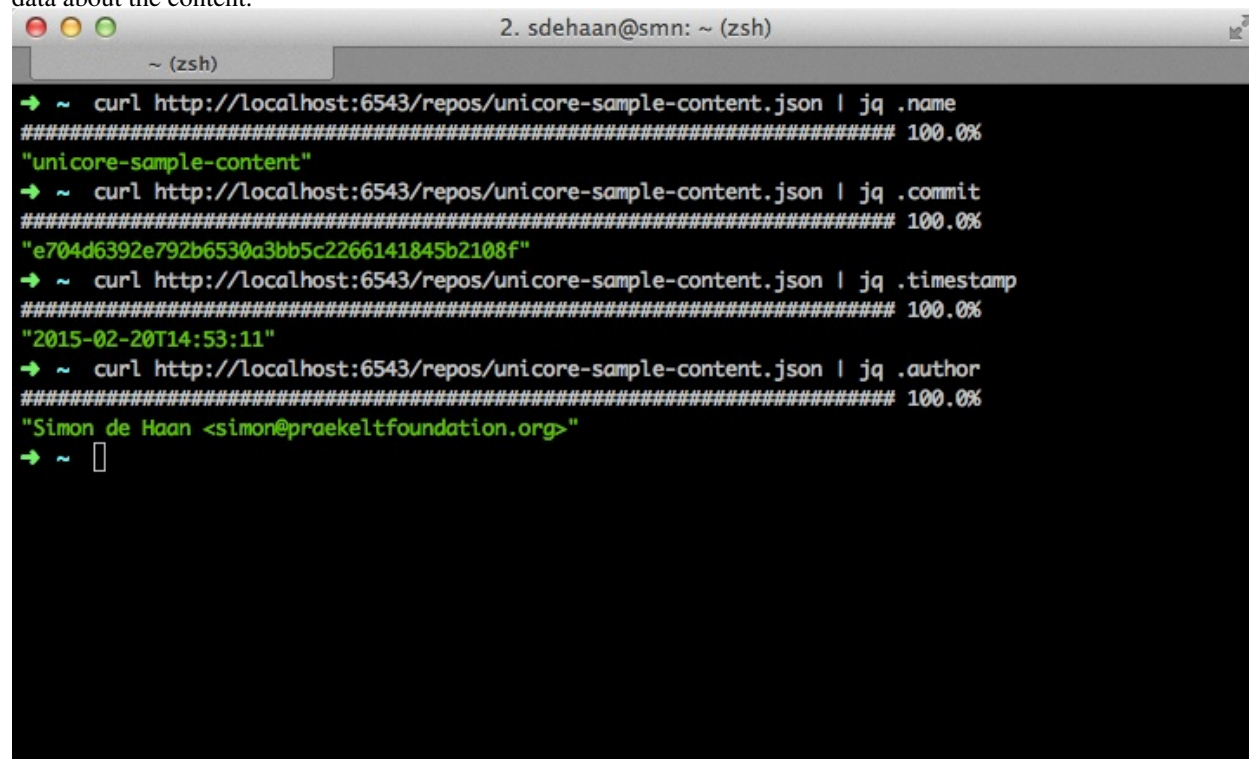
## Querying

---

The individual repositories are all exposed via the `repos.json` base path. Let's step through the process of cloning a repository and then querying the data via the web interface:

```
$ curl -XPOST -H 'Content-Type: application/json' \
  -d '{"repo_url": "https://github.com/smn/unicore-sample-content.git"}' \
  http://localhost:6543/repos.json
```

Now `repos/unicore-sample-content.json` accessible via the API and exposes the schema and some meta-data about the content.

A terminal window titled "2. sdehaan@smn: ~ (zsh)" shows a series of curl commands and their outputs. The first command queries the .name field, returning "unicore-sample-content". The second queries the .commit field, returning a long hexadecimal string. The third queries the .timestamp field, returning "2015-02-20T14:53:11". The fourth queries the .author field, returning "Simon de Haan <simon@praekeltfoundation.org>". The prompt is a tilde character followed by a cursor.

```
2. sdehaan@smn: ~ (zsh)
~ (zsh)
→ ~ curl http://localhost:6543/repos/unicore-sample-content.json | jq .name
##### 100.0%
"unicore-sample-content"
→ ~ curl http://localhost:6543/repos/unicore-sample-content.json | jq .commit
##### 100.0%
"e704d6392e792b6530a3bb5c2266141845b2108f"
→ ~ curl http://localhost:6543/repos/unicore-sample-content.json | jq .timestamp
##### 100.0%
"2015-02-20T14:53:11"
→ ~ curl http://localhost:6543/repos/unicore-sample-content.json | jq .author
##### 100.0%
"Simon de Haan <simon@praekeltfoundation.org>"
→ ~
```

The `schema` key in the repository object has an Avro schema representing the content. This allows one to automatically generate model definitions to work with the data.

```

2. sdehaan@smn: ~ (zsh)
~ (zsh)
→ ~ curl http://localhost:6543/repos/unicore-sample-content.json | jq .schemas
##### 100.0%
{
  "unicore.content.models.Page": {
    "fields": [
      {
        "aliases": [],
        "default": {
          "package_version": "1.0.1",
          "package": "elastic-git",
          "language_version": "2.7.8",
          "language_version_string": "2.7.8 (10f1b29a2bd21f837090286174a9ca030b8680b2, Feb 05 2015, 1
7:48:23)\n[PyPy 2.5.0 with GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)]",
          "language": "python"
        },
        "doc": "Model Version Identifier",
        "type": {
          "fields": [
            {
              "type": [
                "null",
                "string"
              ]
            }
          ]
        }
      }
    ]
  }
}

```

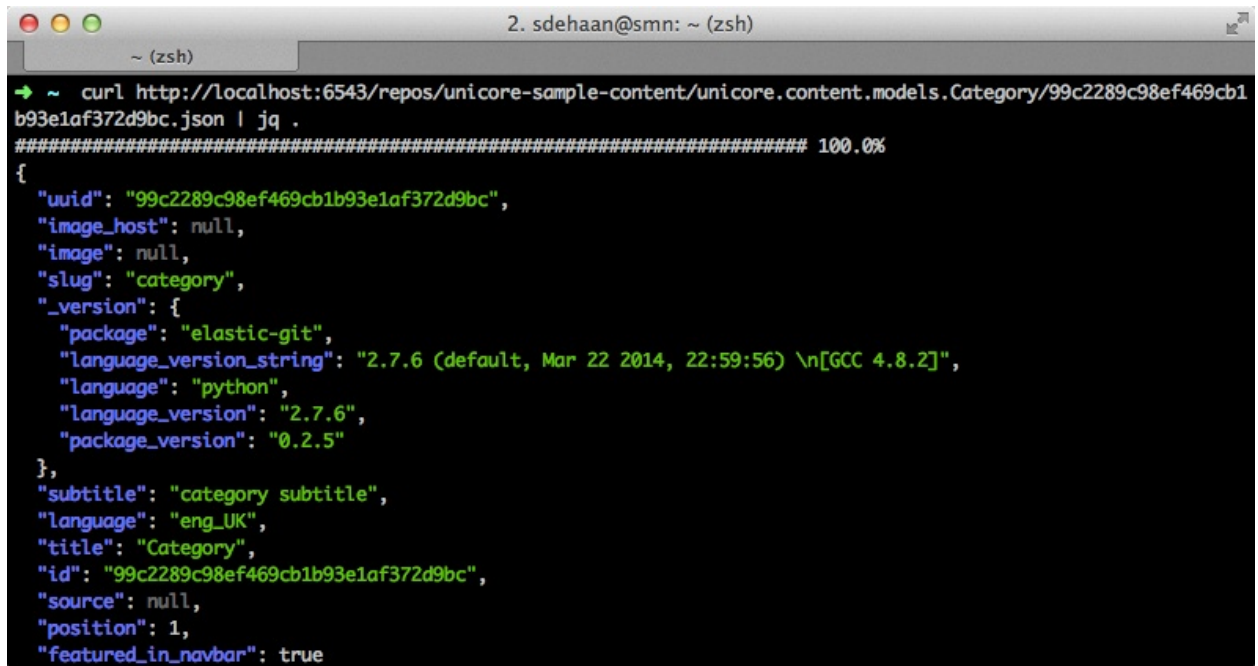
Now that we have a list of all object types in the content repository we can get listings of these models:

```

2. sdehaan@smn: ~ (zsh)
~ (zsh)
→ ~ curl http://localhost:6543/repos/unicore-sample-content/unicore.content.models.Category.json | jq .
##### 100.0%
[
  {
    "uuid": "99c2289c98ef469cb1b93e1af372d9bc",
    "image_host": null,
    "image": null,
    "slug": "category",
    "_version": {
      "package": "elastic-git",
      "language_version_string": "2.7.6 (default, Mar 22 2014, 22:59:56) \n[GCC 4.8.2]",
      "language": "python",
      "language_version": "2.7.6",
      "package_version": "0.2.5"
    },
    "subtitle": "category subtitle",
    "language": "eng_UK",
    "title": "Category",
    "id": "99c2289c98ef469cb1b93e1af372d9bc",
    "source": null,
    "position": 1,
    "featured_in_navbar": true
  }
]

```

Or we can get an individual object by requesting it by its UUID:



A terminal window titled "2. sdehaan@smn: ~ (zsh)" with a sub-tab labeled "~ (zsh)". The terminal shows a command to fetch a JSON file from a local host and pipe it through jq. The output is a JSON object representing a category model.

```
→ ~ curl http://localhost:6543/repos/unicore-sample-content/unicore.content.models.Category/99c2289c98ef469cb1b93e1af372d9bc.json | jq .
##### 100.0%
{
  "uuid": "99c2289c98ef469cb1b93e1af372d9bc",
  "image_host": null,
  "image": null,
  "slug": "category",
  "_version": {
    "package": "elastic-git",
    "language_version_string": "2.7.6 (default, Mar 22 2014, 22:59:56) \n[GCC 4.8.2]",
    "language": "python",
    "language_version": "2.7.6",
    "package_version": "0.2.5"
  },
  "subtitle": "category subtitle",
  "language": "eng_UK",
  "title": "Category",
  "id": "99c2289c98ef469cb1b93e1af372d9bc",
  "source": null,
  "position": 1,
  "featured_in_navbar": true
}
```



---

### URL structure

---

The following URLs are created:

```
http://localhost:6543/repos.json [GET, POST]
http://localhost:6543/repos/<repo-name>.json [GET]
http://localhost:6543/repos/<repo-name>/<content-type>.json [GET]
http://localhost:6543/repos/<repo-name>/<content-type>/<uuid>.json [GET, PUT, DELETE]
```

---

**Note:** The PUT and DELETE methods only operate on the local repository, they are not pushed up to the upstream repository that was cloned.

---