# Comics

*Release 4.1.0*

**Stein Magnus Jodal and contributors**

**Apr 10, 2021**

# SETUP

Comics is a webcomics aggregator. Out of the box it can crawl and archive about two hundred comics every day. The comics are made available through an easy to use web interface were users can build personalized collections of their favorite comics, and then read them on the site or using a feed reader.

Adding a new comic to your installation requires only the addition of a single Python file with some metadata and a few lines of code. To make crawler development easy, Comics comes with both documentation and powerful APIs for crawling web sites and feeds.

CHAPTER

ONE

# PROJECT RESOURCES

- Source code
- Releases
- Issue tracker
- Contributors

## 1.1 Installation

First of all, Comics is just a Django application. Thus, if there are details not outlined in Comics' own docs, you'll probably find the answer in Django's docs. For example, database settings are mentioned on this page, but no details are given, as we're just talking about Django's database settings. Django got better docs for their database settings than we could ever write, so please refer to Django's docs.

### 1.1.1 Get release

You can get hold of Comics in two ways:

- Download the lastest release from https://github.com/jodal/comics/releases.
- Clone the Git repository. You can do so by running:

```
git clone https://github.com/jodal/comics.git
```

  You'll then have the latest development snapshot in the `main` branch:

```
cd comics/
```

  If you want to run a specific release, they are available as tags in the Git repository. If you checkout a tag name, you'll have exactly the same as you find in the release archives:

```
git tag -l
git checkout v4.0.0
```

### 1.1.2 Software requirements

First of all, you need Python 3.7 or newer.

It is recommended to create a virtualenv to isolate the dependencies from other applications on the same system:

```
cd comics/
python3 -m venv .venv
```

Every time you want to use the virtualenv, it must be activated:

```
source .venv/bin/activate
```

If you make use of a virtualenv for a real deployment, you'll also need to make sure that the app server and the cronjob activate the virtualenv.

#### Minimum dependencies

The absolute minimum requirements for getting Comics up and running can be installed with:

```
python3 -m pip install .
```

#### Optional dependencies for real deployments

To deploy Comics, you need a WSGI server. There are several options, but we tend to use Gunicorn. To install it, run:

```
python -m pip install ".[server]"
```

By default, Comics is configured to use an SQLite database. While SQLite is good enough for local development, we recommend PostgreSQL when running Comics long-term. To install the extra dependencies required to use PostgreSQL as the database, run:

```
python3 -m pip install ".[pgsql]"
```

Comics does not require a cache, but responses are significantly faster with a cache available. To install the dependencies required for to use memcached as a cache, run:

```
python3 -m pip install ".[cache]"
```

The Comics API is able to respond using JSON, XML, or several other formats. To install the dependencies required to provide all possible response formats, run:

```
python3 -m pip install ".[api]
```

#### Development dependencies

If you're setting up Comics for development, you should install Poetry, and in the Comics git repository, run:

```
poetry install
```

This installs both the minimum dependencies as described above and all extra dependencies required for development.

## 1.2 Bootstrapping

Once you've installed Comics, you need to create the database and create the initial users and comics.

To get Comics to a state useful for testing of new crawlers and personal usage, the following steps are all that is needed.

### 1.2.1 Create database

A file-based SQLite database will be used, unless you have configured another database, like PostgreSQL.

To create the database and database schema, open a terminal, go to top level directory in your checkout of the Comics repo, where you'll find the file `manage.py`, and run:

```
python manage.py migrate
```

### 1.2.2 Create first user

When `migrate` has finished, create a superuser by running:

```
python manage.py createsuperuser
```

### 1.2.3 Add comics

Then we need to seed the database with information on what comics to crawl. E.g. to add the *XKCD* comic from `comics/comics/xkcd.py`, run:

```
python manage.py comics_addcomics -c xkcd
```

Optionally, you can add all available active comics to the database:

```
python manage.py comics_addcomics -c all
```

### 1.2.4 Get comic releases

Next, we need to get hold of some comic releases, so we will crawl the web for them. This will get today's releases for all added comics:

```
python manage.py comics_getreleases
```

To get the release for a specific added comics, you can filter with `--comic` or `-c`:

```
python manage.py comics_getreleases -c xkcd
```

To get releases for a range of days, you can specify a date range with `--from` or `-f` and `--to` or `-t`. Both defaults to today, so you can leave the end of the range out:

```
python manage.py comics_getreleases -f 2011-11-11
```

### 1.2.5 Development web server

Finally, to be able to browse the comic releases we have aggregated, start the Django development web server by running:

```
python manage.py runserver
```

If you now point your web browser at http://localhost:8000/ you will be able to browse all available comics. If you created a superuser above, you can log in at http://localhost:8000/admin/ to do simple administration tasks, like removing comics or releases.

### 1.2.6 More options

All of the `manage.py` commands got more options available. Add the `--help` argument to any of the commands to get a full listing of the available options.

## 1.3 Deployment

The following example documents *one way* to deploy Comics. As Comics is a standard Django project with an additional batch job for crawling, it may be deployed in just about any way a Django project may be deployed. Please refer to Django's deployment documentation for further details.

In the following examples we assume that we are deploying Comics at http://comics.example.com/, using Nginx, Gunicorn, and PostgreSQL. The Django application and batch job is both running as the user `comics-user`. The static media files, like comic images, are served from http://comics.example.com/static/.

### 1.3.1 Database

Comics should theoretically work with any database supported by Django. Though, development is mostly done on SQLite and PostgreSQL. For production use, PostgreSQL is the recommended choice.

---

**Note:** If you are going to use SQLite in a deployment with Nginx and so on, you need to ensure that the user the web server will be running as has write access to the *directory* the SQLite database file is located in.

---

### 1.3.2 Example `.env`

In the following examples, we assume the Comics source code is unpacked at `/srv/comics.example.com/app/comics`.

To change settings, you should not change the settings files shipped with Comics, but instead override the settings using environment variables, or by creating a file named `/srv/comics.example.com/app/comics/.env`. You must at least set `DJANGO_SECRET_KEY` and database settings, unless you use SQLite.

A full set of environment variables for a production deployment may look like this:

```
DJANGO_SECRET_KEY=replace-this-with-a-long-random-value
DJANGO_ADMIN=comics@example.com
DJANGO_DEFAULT_FROM_EMAIL=comics@example.com

DJANGO_MEDIA_ROOT=/srv/comics.example.com/htdocs/static/media/
```

(continues on next page)

(continued from previous page)

```
DJANGO_MEDIA_URL=https://comics.example.com/static/media/
DJANGO_STATIC_ROOT=/srv/comics.example.com/htdocs/static/
DJANGO_STATIC_URL=https://comics.example.com/static/

DATABASE_URL=postgres://comics:topsecret-password@localhost:5432/comics

CACHE_URL=memcache://127.0.0.1:11211

COMICS_LOG_FILENAME=/srv/comics.example.com/app/log/comics.log
COMICS_SITE_TITLE=comics.example.com
COMICS_INVITE_MODE=true
```

Of course, you should change most, if not all, of these settings to fit your own installation.

If your are not running a `memcached` server, remove `CACHE_URL` variable from your environment. Comics does not require a cache, but responses are significantly faster with a cache available.

### 1.3.3 Example Gunicorn setup

Comics is a WSGI app and can be run with any WSGI server, for example Gunicorn. Gunicorn is a Python program, so you can simply install it in Comics' own virtualenv:

```
source /srv/comics.example.com/app/venv/bin/activate
python -m pip install gunicorn
```

Then you need to start Gunicorn, for example with a systemd service:

```
[Unit]
Description=gunicorn-comics
After=network.target

[Install]
WantedBy=multi-user.target

[Service]
User=comics-user
Group=comics-user
Restart=always

ExecStart=/srv/comics.example.com/app/venv/bin/gunicorn --bind=127.0.0.1:8000 --
↪workers=9 --access-logfile=/srv/comics.example.com/htlogs/gunicorn-access.log --
↪error-logfile=/srv/comics.example.com/htlogs/gunicorn-error.log comics.wsgi
ExecReload=/bin/kill -s HUP $MAINPID
ExecStop=/bin/kill -s TERM $MAINPID

WorkingDirectory=/srv/comics.example.com/app/comics
Environment=PYTHONPATH='/srv/comics.example.com/app/comics'

PrivateTmp=true
```

### 1.3.4 Example Nginx vhost

The web server Nginx can be used in front of Gunicorn to terminate HTTPS connections and effectively serve static files.

The following is an example of a complete Nginx vhost:

```
server {
    server_name comics.example.com;
    listen 443 ssl http2;
    listen [::]:443 ssl http2;

    access_log /srv/comics.example.com/htlogs/nginx-access.log;
    error_log /srv/comics.example.com/htlogs/nginx-error.log error;

    ssl_certificate /etc/letsencrypt/live/comics.example.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/comics.example.com/privkey.pem;

    location /static {
        root /srv/comics.example.com/htdocs;
        expires max;

        location ~* \/fonts\/ {
            add_header Access-Control-Allow-Origin *;
        }
    }

    location / {
        proxy_pass_header Server;
        proxy_set_header Host $http_host;
        proxy_redirect off;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Scheme $scheme;
        proxy_connect_timeout 10;
        proxy_read_timeout 30;
        proxy_pass http://localhost:8000/;
    }
}
```

For details, please refer to the documentation of the Nginx project.

### 1.3.5 Collecting static files

When you're not running in development mode, you'll need to collect the static files from all apps into the `STATIC_ROOT`. To do this, run:

```
python manage.py collectstatic
```

You have to rerun this command every time you deploy changes to graphics, CSS and JavaScript. For more details, see the Django documentation on staticfiles.

### 1.3.6 Example cronjob

To get new comics releases, you should run `comics_getreleases` regularly. In addition, you should run `clearsessions` to clear expired user sessions. One way is to use `cron` e.g. by placing the following in `/etc/cron.d/comics`:

```
MAILTO=comics@example.com
PYTHONPATH=/srv/comics.example.com/app/comics
1 * * * * comics-user python /srv/comics.example.com/app/comics/manage.py comics_
↪getreleases -v0
1 3 * * * comics-user python /srv/comics.example.com/app/comics/manage.py␣
↪clearsessions -v0
```

If you have installed Comics' dependencies in a virtualenv instead of globally, the cronjob must also activate the virtualenv. This can be done by using the `python` interpreter from the virtualenv:

```
MAILTO=comics@example.com
PYTHONPATH=/srv/comics.example.com/app/comics
1 * * * * comics-user /srv/comics.example.com/app/venv/bin/python /srv/comics.example.
↪com/app/comics/manage.py comics_getreleases -v0
1 3 * * * comics-user /srv/comics.example.com/app/venv/bin/python /srv/comics.example.
↪com/app/comics/manage.py clearsessions -v0
```

By setting `MAILTO` any exceptions raised by the comic crawlers will be sent by mail to the given mail address. `1 * * * *` specifies that the command should be run 1 minute past every hour.

## 1.4 Creating crawlers

For each comic Comics is aggregating, we need to create a crawler. At the time of writing, more than 200 crawlers are available in the `comics/comics/` directory. They serve as a great source for learning how to write new crawlers for Comics.

### 1.4.1 A crawler example

The crawlers are split in two separate pieces. The *`ComicData`* part contains meta data about the comic used for display at the web site. The *`Crawler`* part contains properties needed for crawling and the crawler implementation itself.

```
from comics.aggregator.crawler import CrawlerBase, CrawlerImage
from comics.core.comic_data import ComicDataBase

class ComicData(ComicDataBase):
    name = 'xkcd'
    language = 'en'
    url = 'https://www.xkcd.com/'
    start_date = '2005-05-29'
    rights = 'Randall Munroe, CC BY-NC 2.5'

class Crawler(CrawlerBase):
    history_capable_days = 10
    schedule = 'Mo,We,Fr'
    time_zone = 'US/Eastern'
```

(continues on next page)

(continued from previous page)

```
    def crawl(self, pub_date):
        feed = self.parse_feed('https://www.xkcd.com/rss.xml')
        for entry in feed.for_date(pub_date):
            url = entry.summary.src('img[src*="/comics/"]')
            title = entry.title
            text = entry.summary.alt('img[src*="/comics/"]')
            return CrawlerImage(url, title, text)
```

### 1.4.2 The `ComicData` class fields

**class ComicData**

**name**
*Required.* A string with the name of the comic.

**url**
*Required.* A string with the URL of the comic's web page.

**active**
*Optional.* Wheter or not this comic is still being crawled. Defaults to `True`.

**start_date**
*Optional.* The first date the comic was published at.

**end_date**
*Optional.* The last date the comic was published at if it is discontinued.

**rights**
*Optional.* Name of the author and the comic's license if available.

### 1.4.3 The `Crawler` class fields

**class Crawler**

**history_capable_date**
*Optional.* Date of oldest release available for crawling.

Provide this *or* *`Crawler.history_capable_days`*. If both are present, this one will have precedence.

Example: `'2008-03-08'`.

**history_capable_days**
*Optional.* Number of days a release is available for crawling.

Provide this *or* *`Crawler.history_capable_date`*.

Example: `32`.

**schedule**
*Optional.* On what weekdays the comic is published.

Example: `'Mo,We,Fr'` or `'Mo,Tu,We,Th,Fr,Sa,Su'`.

**time_zone**
*Optional.* In approximately what time zone the comic is published.

Example: `Europe/Oslo` or `US/Eastern`.

See the IANA timezone database for a list of possible values.

**`multiple_releases_per_day`**
*Optional.* Default: `False`. Whether to allow multiple releases per day.

Example: `True` or `False`.

**`has_rerun_releases`**
*Optional.* Default: `False`. Whether the comic reruns old images as new releases.

Example: `True` or `False`.

**`headers`**
*Optional.* Default: `None`. Any HTTP headers to send with any URI request for values.

Useful if you're pulling comics from a site that checks either the `Referer` or `User-Agent`. If you can view the comic using your browser but not when using your loader for identical URLs, try setting the `Referer` to be `http://www.example.com/` or set the `User-Agent` to be `Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)`.

Example: `{'Referer': 'http://www.example.com/', 'Host': 'http://www.example.com/'}`

### 1.4.4 The `Crawler.crawl()` method

The `Crawler.crawl()` is where the real work is going on. To start with an example, let's look at *XKCD*'s `Crawler.crawl()` method:

```
def crawl(self, pub_date):
    feed = self.parse_feed('http://www.xkcd.com/rss.xml')
    for entry in feed.for_date(pub_date):
        url = entry.summary.src('img[src*="/comics/"]')
        title = entry.title
        text = entry.summary.alt('img[src*="/comics/"]')
        return CrawlerImage(url, title, text)
```

#### Arguments and return values

The `Crawler.crawl()` method takes a single argument, `pub_date`, which is a `datetime.date` object for the date the crawler is currently crawling. The goal of the method is to return a `CrawlerImage` object containing at least the URL of the image for `pub_date` and optionally a `title` and `text` accompanying the image. `CrawlerImage`'s signature is:

```
CrawlerImage(url, title=None, text=None)
```

This means that you must always supply an URL, and that you can supply a `text` without a `title`. The following are all valid ways to create a `CrawlerImage`:

```
CrawlerImage(url)
CrawlerImage(url, title)
CrawlerImage(url, title, text)
CrawlerImage(url, text=text)
```

For some crawlers, this is all you need. If the image URL is predictable and based upon the `pub_date` in some way, just create the URL with the help of Python's strftime documentation, and return it wrapped in a `CrawlerImage`:

```
def crawl(self, pub_date):
    url = 'http://www.example.com/comics/%s.png' % (
        pub_date.strftime('%Y-%m-%d'),
    )
    return CrawlerImage(url)
```

Though, for most crawlers, some interaction with RSS or Atom feeds or web pages are needed. For this a *web parser* and a *feed parser* are provided.

#### Returning multiple images for a single comic release

Some comics got releases with multiple images, and thus returning a single `CrawlerImage` will not be enough for you. For situations like these, Comics lets you return a list of `CrawlerImage` objects from `Crawler.crawl()`. The list should be ordered in the same way as the comic is meant to be read, with the first frame as the first element in the list. If the comic release got a `title`, add it to the first `CrawlerImage` object, and let the `title` field stay empty on the rest of the list elements. The same applies for the `text` field, unless each image actually got a different `title` or `text` string.

The following is an example of a `Crawler.crawl()` method which returns multiple images. It adds a `title` to the first list element, and different `text` to all of the elements.

```
def crawl(self, pub_date):
    feed = self.parse_feed('http://feeds.feedburner.com/Pidjin')
    for entry in feed.for_date(pub_date):
        result = []
        for i in range(1, 10):
            url = entry.content0.src('img[src$="000%d.jpg"]' % i)
            text = entry.content0.title('img[src$="000%d.jpg"]' % i)
            if url and text:
                result.append(CrawlerImage(url, text=text))
        if result:
            result[0].title = entry.title
        return result
```

### 1.4.5 `LxmlParser` – Parsing web pages and HTML

The web parser, internally known as *`LxmlParser`*, uses CSS selectors to extract content from HTML. For a primer on CSS selectors, see *Matching HTML elements using CSS selectors*.

The web parser is accessed through the `Crawler.parse_page()` method:

```
def crawl(self, pub_date):
    page_url = 'http://ars.userfriendly.org/cartoons/?id=%s' % (
        pub_date.strftime('%Y%m%d'),)
    page = self.parse_page(page_url)
    url = page.src('img[alt^="Strip for"]')
    return CrawlerImage(url)
```

This is a common pattern for crawlers. Another common patterns is to use a feed to find the web page URL for the given date, then parse that web page to find the image URL.

### `LxmlParser` API

The available methods only require a CSS selector, `selector`, to match tags. In the event that the selector doesn't match any elements, `default` will be returned.

If the `selector` matches multiple elements, one of two things will happen:

- If `allow_multiple` is `False`, a `MultipleElementsReturned` exception is raised.
- If `allow_multiple` is `True`, a list of zero or more elements is returned with all of the elements matching `selector`.

**class** comics.aggregator.lxmlparser.**LxmlParser**

**text**(*selector*[, *default=None*, *allow_multiple=False*])
Returns the text contained by the element matching `selector`.

**src**(*selector*[, *default=None*, *allow_multiple=False*])
Returns the `src` attribute of the element matching `selector`.

The web parser automatically expands relative URLs in the source, like `/comics/2008-04-13.png` to a full URL like `http://www.example.com/2008-04-13.png`, so you do not need to think about that.

**alt**(*selector*[, *default=None*, *allow_multiple=False*])
Returns the `alt` attribute of the element matching `selector`.

**title**(*selector*[, *default=None*, *allow_multiple=False*])
Returns the `title` attribute of the element matching `selector`.

**href**(*selector*[, *default=None*, *allow_multiple=False*])
Returns the `href` attribute of the element matching `selector`.

**value**(*selector*[, *default=None*, *allow_multiple=False*])
Returns the `value` attribute of the element matching `selector`.

**id**(*selector*[, *default=None*, *allow_multiple=False*])
Returns the `id` attribute of the element matching `selector`.

**remove**(*selector*)
Remove the elements matching `selector` from the parsed document.

### Matching HTML elements using CSS selectors

Both web page and feed parsing uses CSS selectors to extract the interesting strings from HTML. CSS selectors are those normally simple strings you use in CSS style sheets to select what elements of your web page the CSS declarations should be applied to.

In the following example `h1 a` is the selector. It matches all `a` elements contained in `h1` elements. The rule to be applied to the matching elements is `color: red;`.

```
h1 a { color: red; }
```

Similarly `class="foo"` and `id="bar"` in HTML may be used in CSS selectors. The following CSS example would color all `h1` headers with the class `foo` red, and all elements with the ID `bar` which is contained in `h1` elements would be colored blue.

```
h1.foo { color: red; }
h1 #bar { color: blue; }
```

In CSS3, the power of CSS selectors have been greatly increased by the addition of matching by the content of elements' attributes. To match all `img` elements with a `src` attribute *starting with* `http://www.example.com/` simply write:

```
img[src^="http://www.example.com/"]
```

Or, to match all `img` elements whose `src` attribute *ends in* `.jpg`:

```
img[src$=".jpg"]
```

Or, `img` elements whose `src` attribute *contains* `/comics/`:

```
img[src*="/comics/"]
```

Or, `img` elements whose `alt` attribute *is* `Today's comic`:

```
img[alt="Today's comic"]
```

For further details on CSS selectors in general, please refer to http://css.maxdesign.com.au/selectutorial/.

### 1.4.6 `FeedParser` – Parsing feeds

The feed parser is initialized with a feed URL passed to `Crawler.parse_feed()`, just like the web parser is initialized with a web page URL:

```
def crawl(pub_date):
    ...
    feed = self.parse_feed('http://www.xkcd.com/rss.xml')
    ...
```

#### `FeedParser` API

The `feed` object provides two methods which both returns feed elements: *`FeedParser.for_date()`* and *`FeedParser.all()`*. Typically, a crawler uses *`FeedParser.for_date()`* and loops over all entries it returns to find the image URL:

```
for entry in feed.for_date(pub_date):
    # parsing comes here
    return CrawlerImage(url)
```

**class** comics.aggregator.feedparser.**FeedParser**

**for_date**(*date*)
Returns all feed elements published at `date`.

**all**()
Returns all feed elements.

#### Feed `Entry` API

The Comics feed parser is really a combination of the popular feedparser library and *`LxmlParser`*. It can do anything *feedparser* can do, and in addition you can use the *`LxmlParser`* methods on feed fields which contains HTML, like *`Entry.summary`* and *`Entry.content0`*.

**class** comics.aggregator.feedparser.**Entry**

**summary**
This is the most frequently used entry field which supports HTML parsing with the *`LxmlParser`* methods.

Example usage:

```
url = entry.summary.src('img')
title = entry.summary.alt('img')
```

**content0**
This is the same as *feedparser*'s `content[0].value` field, but with *`LxmlParser`* methods available. For some crawlers, this is where the interesting stuff is found.

**html** (*string*)
Wrap `string` in a *`LxmlParser`*.

If you need to parse HTML in any other fields than *`summary`* and *`content0`*, you can apply the `html(string)` method on the field, like it is applied on a feed entry's title field here:

```
title = entry.html(entry.title).text('h1')
```

**tags**
List of tags associated with the entry.

### 1.4.7 Testing your new crawler

When the first version of you crawler is complete, it's time to test it.

The file name is important, as it is used as the comic's slug. This means that it must be unique within the Comics installation, and that it is used in the URLs Comics will serve the comic at. For this example, we call the crawler file `foo.py`. The file must be placed in the `comics/comics/` directory, and will be available in Python as `comics.comics.foo`.

#### Loading `ComicData` for your new comic

For Comics to know about your new crawler, you need to load the comic meta data into Comics' database. To do so, we run the `comics_addcomics` command:

```
python manage.py comics_addcomics -c foo
```

If you do any changes to the *`ComicData`* class of any crawler, you must rerun `comics_addcomics` to update the database representation of the comic.

#### Running the crawler

When `comics_addcomics` has created a `comics.core.models.Comic` instance for the new crawler, you may use your new crawler to fetch the comic's release for the current date by running:

```
python manage.py comics_getreleases -c foo
```

If you want to get comics releases for more than the current day, you may specify a date range to crawl, like:

```
python manage.py comics_getreleases -c foo -f 2009-01-01 -t 2009-03-31
```

The date range will automatically be adjusted to the crawlers *history capability*. You may also get comics for a date range without a specific end. In which case, the current date will be used instead:

```
python manage.py comics_getreleases -c foo -f 2009-01-01
```

If your new crawler is not working properly, you may add `-v2` to the command to turn on full debug output:

```
python manage.py comics_getreleases -c foo -v2
```

For a full overview of `comics_getreleases` options, run:

```
python manage.py comics_getreleases --help
```

### 1.4.8 Submitting your new crawler for inclusion in Comics

When your crawler is working properly, you may submit it for inclusion in Comics. You should fork Comics at GitHub, commit your new crawler to your own fork, and send me a *pull request* through GitHub.

All contributions must be granted under the same license as Comics itself.

## 1.5 Web API

Comics comes with a web API that exposes all useful data about the current user, the user's comics subscriptions, comics, comic releases, and comic images. The web API may be used to e.g. create iOS/Android apps or alternative comics browsers, while leaving the comics crawling job to a Comics instance.

Please make any apps using this API generic, so that they can be used with any Comics instance as the backend. In other words, when starting your app, let the end user enter the hostname of the Comics instance, in addition to his secret key or email/password pair.

### 1.5.1 Authentication

The web API is only available for users with an active user account on the Comics instance. The user must authenticate himself using the same secret key as is used to access comic feeds. The key can be found in the account section of the Comics instance.

The secret key can be provided in one of two ways:

- Using a HTTP GET parameter named `key`, i.e. as part of the URL. Example:

  ```
  http://example.com/api/v1/users/?key=76acdcdf16ae4e12becb00d09a9d9456
  ```

- Using the `Authorization` HTTP header. Example:

```
Authorization: Key 76acdcdf16ae4e12becb00d09a9d9456
```

#### Get secret key using email and password

If it's inconvenient for the end user to enter the secret key in your user interface–on mobile phones copy-pasting the API key from the Comics instance's account page is time consuming at best–you may retrieve the secret key on behalf of the end user by following steps:

1. Ask the end user to provide:
   - the Comics instance's base URL (e.g. `example.com`)
   - the email address the end user have registered on the Comics instance
   - the password the end user have registered on the Comics instance
2. Use the provided information to retrieve the *Users resource* from the API. Authenticate using Basic Authentication with the email address as username and the password as password. This does only work for the *Users resource*.
3. In the response from the *Users resource*, you'll find the end user's secret key. You should cache the secret key in your application and use it for all future requests to the API on behalf of this user. The end user's password should be thrown away at this point.

### 1.5.2 Response format

You can specify what response format you prefer in one of two ways:

- Using a HTTP GET parameter named `format`, i.e. as part of the URL. Examples:

```
# Returns JSON
http://example.com/api/v1/?format=json

# Returns JSONP with function name 'callback'
http://example.com/api/v1/?format=jsonp

# Returns JSONP with function name 'foo'
http://example.com/api/v1/?format=jsonp&callback=foo

# Returns JSONP with function name 'foo'
http://example.com/api/v1/?callback=foo

# Returns XML
http://example.com/api/v1/?format=xml

# Returns YAML
http://example.com/api/v1/?format=yaml

# Returns Apple binary plist
http://example.com/api/v1/?format=plist
```

- Using the `Accept` HTTP header. Examples:

```
# Returns JSON
Accept: application/json
```

(continues on next page)

(continued from previous page)

```
# Returns JSONP with function name 'callback'
Accept: text/javascript

# Returns XML
Accept: application/xml

# Returns YAML
Accept: text/yaml

# Returns Apple binary plist
Accept: application/x-plist
```

JSON and JSONP are always supported. Other formats like XML, YAML, and Apple binary plists (bplist) may be available if the Comics instance got the additional dependencies required by the format installed.

If you run a Comics instance yourself, and want support for more response formats, check out Tastypie's serialization docs for details on what you need to install.

### 1.5.3 Pagination

All the resource collections support pagination. The pagination parameters that may be passed as GET arguments are:

- **limit** – max number of returned resources per response. Defaults to 20. Use 0 to remove the limit and request all objects in a single response.
- **offset** – offset into the full collection of resources. Defaults to 0.

The `meta` section of the collection responses include the current pagination parameters, and–if available–links to the previous and next page, and the total count of resources matching the query:

- **next** – link to the next page of the collection, if available
- **previous** – link to the previous page of the collection, if available
- **total_count** – total number of resources in the collection, given the current filters.

### 1.5.4 Resources

#### Root resource

**GET /api/v1/**
Lists all available resources, and URLs for their schemas.

#### Users resource

**GET /api/v1/users/**
List of all authenticated users. Not surprisingly, it always has a single result.

**Example request using secret key**

```
GET /api/v1/users/ HTTP/1.1
Host: example.com
Accept: application/json
Authorization: Key 76acdcdf16ae4e12becb00d09a9d9456
```

**Example request using Basic Authentication**

This is the only resource that also accepts Basic Authentication, using the user's email address and password. Use the secret key from the response for authenticating all future requests to the API.

```
GET /api/v1/users/ HTTP/1.1
Host: example.com
Accept: application/json
Authorization: Basic YWxpY2VAZXhhbXBsZS5jb206c2VjcmV0
```

**Example response**

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{
    "meta": {
        "limit": 20,
        "next": null,
        "offset": 0,
        "previous": null,
        "total_count": 1
    },
    "objects": [
        {
            "date_joined": "2012-04-30T18:39:59+00:00",
            "email": "alice@example.com",
            "last_login": "2012-06-09T23:09:54.312109+00:00",
            "resource_uri": "/api/v1/users/1/",
            "secret_key": "76acdcdf16ae4e12becb00d09a9d9456"
        }
    ]
}
```

**Status Codes**

- 200 OK – no error
- 401 Unauthorized – authentication/authorization failed

## Comics resource

**GET /api/v1/comics/**

Lists all available comics. Supports *Pagination*.

**Example request**

```
GET /api/v1/comics/?slug=xkcd HTTP/1.1
Host: example.com
Accept: application/json
Authorization: Key 76acdcdf16ae4e12becb00d09a9d9456
```

**Example response**

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
```

(continues on next page)

(continued from previous page)


```
{
    "meta": {
        "limit": 20,
        "next": null,
        "offset": 0,
        "previous": null,
        "total_count": 1
    },
    "objects": [
        {
            "active": true,
            "added": "0001-01-01T00:00:00+00:00",
            "end_date": null,
            "id": "18",
            "language": "en",
            "name": "xkcd",
            "resource_uri": "/api/v1/comics/18/",
            "rights": "Randall Munroe, CC BY-NC 2.5",
            "slug": "xkcd",
            "start_date": "2005-05-29",
            "url": "http://www.xkcd.com/"
        }
    ]
}
```


**Query Parameters**

- **subscribed** – only include comics the authorized user is subscribed to if `true`, or unsubscribed to if `false`
- **active** – only include active comics (`true`) or inactive comics (`false`)
- **language** – only include comics with the exact language, e.g. `en`
- **name** – only include comics with matching name. Queries like `name__startswith=Dilbert` and `name__iexact=XkcD` are supported.
- **slug** – only include comics with matching slug. Queries like `slug__contains=kc` and `slug__endswith=db` are supported.

**Status Codes**

- 200 OK – no error
- 400 Bad Request – bad request, e.g. unknown filter used
- 401 Unauthorized – authentication/authorization failed

**GET /api/v1/comics/(int: *comic_id*)/**

Show a specific comic looked up by comic ID.

**Example request**

```
GET /api/v1/comics/18/ HTTP/1.1
Host: example.com
Accept: application/json
Authorization: Key 76acdcdf16ae4e12becb00d09a9d9456
```

**Example response**

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{
    "active": true,
    "added": "0001-01-01T00:00:00+00:00",
    "end_date": null,
    "id": "18",
    "language": "en",
    "name": "xkcd",
    "resource_uri": "/api/v1/comics/18/",
    "rights": "Randall Munroe, CC BY-NC 2.5",
    "slug": "xkcd",
    "start_date": "2005-05-29",
    "url": "http://www.xkcd.com/"
}
```

**Parameters**

- **comic_id** – the comic ID

**Status Codes**

- 200 OK – no error
- 401 Unauthorized – authentication/authorization failed
- 404 Not Found – comic not found

### Releases resource

**GET /api/v1/releases/**

Lists all available releases, last fetched first. Supports *Pagination*.

**Example request**

```
GET /api/v1/releases/?comic__slug=xkcd&limit=2 HTTP/1.1
Host: example.com
Accept: application/json
Authorization: Key 76acdcdf16ae4e12becb00d09a9d9456
```

**Example response**

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{
    "meta": {
        "limit": 2,
        "next": "/api/v1/releases/?limit=2&key=76acdcdf16ae4e12becb00d09a9d9456&
↪format=json&comic__slug=xkcd&offset=2",
        "offset": 0,
        "previous": null,
        "total_count": 1104
    },
    "objects": [
        {
            "comic": "/api/v1/comics/18/",
```

(continues on next page)

(continued from previous page)


```
            "fetched": "2012-10-08T04:03:56.411028+00:00",
            "id": "147708",
            "images": [
                {
                    "checksum":
↪"605d9a6d415676a21ee286fe2b369f58db62c397bfdfa18710b96dcbbcc4df12",
                    "fetched": "2012-10-08T04:03:56.406586+00:00",
                    "file": "https://comics.example.com/static/media/xkcd/6/
↪605d9a6d415676a21ee286fe2b369f58db62c397bfdfa18710b96dcbbcc4df12.png",
                    "height": 365,
                    "id": "151937",
                    "resource_uri": "/api/v1/images/151937/",
                    "text": "Facebook, Apple, and Google all got away with their␣
↪monopolist power grabs because they don't have any 'S's in their names for␣
↪critics to snarkily replace with '$'s.",
                    "title": "Microsoft",
                    "width": 278
                }
            ],
            "pub_date": "2012-10-08",
            "resource_uri": "/api/v1/releases/147708/"
        },
        {
            "comic": "/api/v1/comics/18/",
            "fetched": "2012-10-05T05:03:33.744355+00:00",
            "id": "147172",
            "images": [
                {
                    "checksum":
↪"6d1b67d3dc00d362ddb5b5e8f1c3f174926d2998ca497e8737ff8b74e7100997",
                    "fetched": "2012-10-05T05:03:33.737231+00:00",
                    "file": "https://comics.example.com/static/media/xkcd/6/
↪6d1b67d3dc00d362ddb5b5e8f1c3f174926d2998ca497e8737ff8b74e7100997.png",
                    "height": 254,
                    "id": "151394",
                    "resource_uri": "/api/v1/images/151394/",
                    "text": "According to my mom, my first word was (looking up␣
↪at the sky) 'Wow!'",
                    "title": "My Sky",
                    "width": 713
                }
            ],
            "pub_date": "2012-10-05",
            "resource_uri": "/api/v1/releases/147172/"
        }
    ]
}
```


**Query Parameters**

- **subscribed** – only include releases the authorized user is subscribed to if `true`, or unsubscribed to if `false`
- **comic** – only include releases with matching comic. All filters on the comic resource may be used, e.g. `comic__slug=xkcd`.
- **images** – only include releases with matching image. All filters on the image resource may be used, e.g. `images__height__gt=1000`.

- **pub_date** – only include releases with pub_date matching. Date range queries, like `pub_date__year=2011` or `pub_date__gte=2011-01-01&pub_date__lte=2011-12-31`, are supported.
- **fetched** – only include releases with fetched matching. Date range queries are supported.

**Status Codes**

- 200 OK – no error
- 400 Bad Request – bad request, e.g. unknown filter used
- 401 Unauthorized – authentication/authorization failed

**GET /api/v1/releases/(int: *release_id*)/**

Show a specific release looked up by release ID.

**Example request**

```
GET /api/v1/releases/147708/ HTTP/1.1
Host: example.com
Accept: application/json
Authorization: Key 76acdcdf16ae4e12becb00d09a9d9456
```

**Example response**

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{
    "comic": "/api/v1/comics/18/",
    "fetched": "2012-10-08T04:03:56.411028+00:00",
    "id": "147708",
    "images": [
        {
            "checksum":
↪"605d9a6d415676a21ee286fe2b369f58db62c397bfdfa18710b96dcbbcc4df12",
            "fetched": "2012-10-08T04:03:56.406586+00:00",
            "file": "https://comics.example.com/static/media/xkcd/6/
↪605d9a6d415676a21ee286fe2b369f58db62c397bfdfa18710b96dcbbcc4df12.png",
            "height": 365,
            "id": "151937",
            "resource_uri": "/api/v1/images/151937/",
            "text": "Facebook, Apple, and Google all got away with their␣
↪monopolist power grabs because they don't have any 'S's in their names for␣
↪critics to snarkily replace with '$'s.",
            "title": "Microsoft",
            "width": 278
        }
    ],
    "pub_date": "2012-10-08",
    "resource_uri": "/api/v1/releases/147708/"
}
```

**Parameters**

- **release_id** – the release ID

**Status Codes**

- 200 OK – no error

- 401 Unauthorized – authentication/authorization failed
- 404 Not Found – release not found

### Images resource

You will probably not use the images resource, as the images are available through the `releases` resource as well. The images resource is included to give the images referenced to by releases their own canonical URLs.

**GET /api/v1/images/**
Lists all images. Supports *Pagination*.

**Query Parameters**

- **fetched** – only include images with fetched matching. Date range queries are supported.
- **title** – only include images with matching title. Queries like `title__icontains=cake` are supported.
- **text** – only include images with matching text. Queries like `text__icontains=lies` are supported.
- **height** – only include images with height matching. Integer range queries, like `height__gt=1000` are supported.
- **width** – only include images with width matching. Integer range queries are supported.

**Status Codes**

- 200 OK – no error
- 400 Bad Request – bad request, e.g. unknown filter used
- 401 Unauthorized – authentication/authorization failed

**GET /api/v1/images/(int: *image_id*)/**
Show a specific image looked up by image ID.

**Parameters**

- **image_id** – the image ID

**Status Codes**

- 200 OK – no error
- 401 Unauthorized – authentication/authorization failed
- 404 Not Found – image not found

### Subscriptions resource

**GET /api/v1/subscriptions/**
List all the authenticated user's comic subscriptions. Supports *Pagination*.

**Example request**

```
GET /api/v1/subscriptions/?comic__slug=xkcd HTTP/1.1
Host: example.com
Accept: application/json
Authorization: Key 76acdcdf16ae4e12becb00d09a9d9456
```

**Example response**

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{
    "meta": {
        "limit": 20,
        "next": null,
        "offset": 0,
        "previous": null,
        "total_count": 1
    },
    "objects": [
        {
            "comic": "/api/v1/comics/18/",
            "id": "2",
            "resource_uri": "/api/v1/subscriptions/2/"
        }
    ]
}
```

**Query Parameters**

- **comic** – only include releases with matching comic. All filters on the comic resource may be used, e.g. `comic__slug=xkcd`.

**Status Codes**

- 200 OK – no error
- 401 Unauthorized – authentication/authorization failed

### POST /api/v1/subscriptions/

Subscribe the authenticated user to the given comic.

**Example request**

Note that the request specifies the `Content-Type` since it includes a body with JSON.

```
POST /api/v1/subscriptions/ HTTP/1.1
Host: example.com
Accept: application/json
Authorization: Key 76acdcdf16ae4e12becb00d09a9d9456
Content-Type: application/json

{
    "comic": "/api/v1/comics/18/"
}
```

**Example response**

```
HTTP/1.1 201 CREATED
Content-Type: text/html; charset=utf-8
Location: https://example.com/api/v1/subscriptions/4/
```

**Status Codes**

- 201 Created – no error, object was created, see `Location` header
- 401 Unauthorized – authentication/authorization failed

- 500 Internal Server Error – if the request cannot be processed, e.g. because the subscription already exists

**PATCH /api/v1/subscriptions/**

Do bulk updates of subscriptions: e.g. create and delete multiple subscriptions with a single request.

If any part of the bulk update fails, all changes are rolled back.

**Example request**

```
PATCH /api/v1/subscriptions/ HTTP/1.1
Host: example.com
Accept: application/json
Authorization: Key 76acdcdf16ae4e12becb00d09a9d9456
Content-Type: application/json

{
    "objects": [
        {
            "comic": "/api/v1/comics/19/"
        },
        {
            "comic": "/api/v1/comics/20/"
        }
    ],
    "deleted_objects": [
        "/api/v1/subscriptions/4/",
        "/api/v1/subscriptions/5/"
    ]
}
```

**Example response**

```
HTTP/1.1 202 ACCEPTED
Content-Length: 0
Content-Type: text/html; charset=utf-8
```

**Status Codes**

- 202 Accepted – no error, changes was accepted, use GET to see the changes
- 401 Unauthorized – authentication/authorization failed
- 500 Internal Server Error – if the request cannot be processed, e.g. because a subscription already exists

**GET /api/v1/subscriptions/(int: *subscription_id*)/**

Show one of the authenticated user's comic subscriptions looked up by subscription ID.

**Example request**

```
GET /api/v1/subscriptions/2/ HTTP/1.1
Host: example.com
Accept: application/json
Authorization: Key 76acdcdf16ae4e12becb00d09a9d9456
```

**Example response**

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{
    "comic": "/api/v1/comics/18/",
    "id": "2",
    "resource_uri": "/api/v1/subscriptions/2/"
}
```

**Parameters**

- **subscription_id** – the subscription ID

**Status Codes**

- 200 OK – no error
- 401 Unauthorized – authentication/authorization failed
- 404 Not Found – subscription not found

**DELETE /api/v1/subscriptions/**(**int:** *subscription_id*)**/**

Unsubscribe the authenticated user from the given comic.

**Example request**

```
DELETE /api/v1/subscriptions/17/ HTTP/1.1
Host: example.com
Accept: application/json
Authorization: Key 76acdcdf16ae4e12becb00d09a9d9456
```

**Example response**

```
HTTP/1.1 204 NO CONTENT
Content-Length: 0
Content-Type: text/html; charset=utf-8
```

**Parameters**

- **subscription_id** – the subscription ID

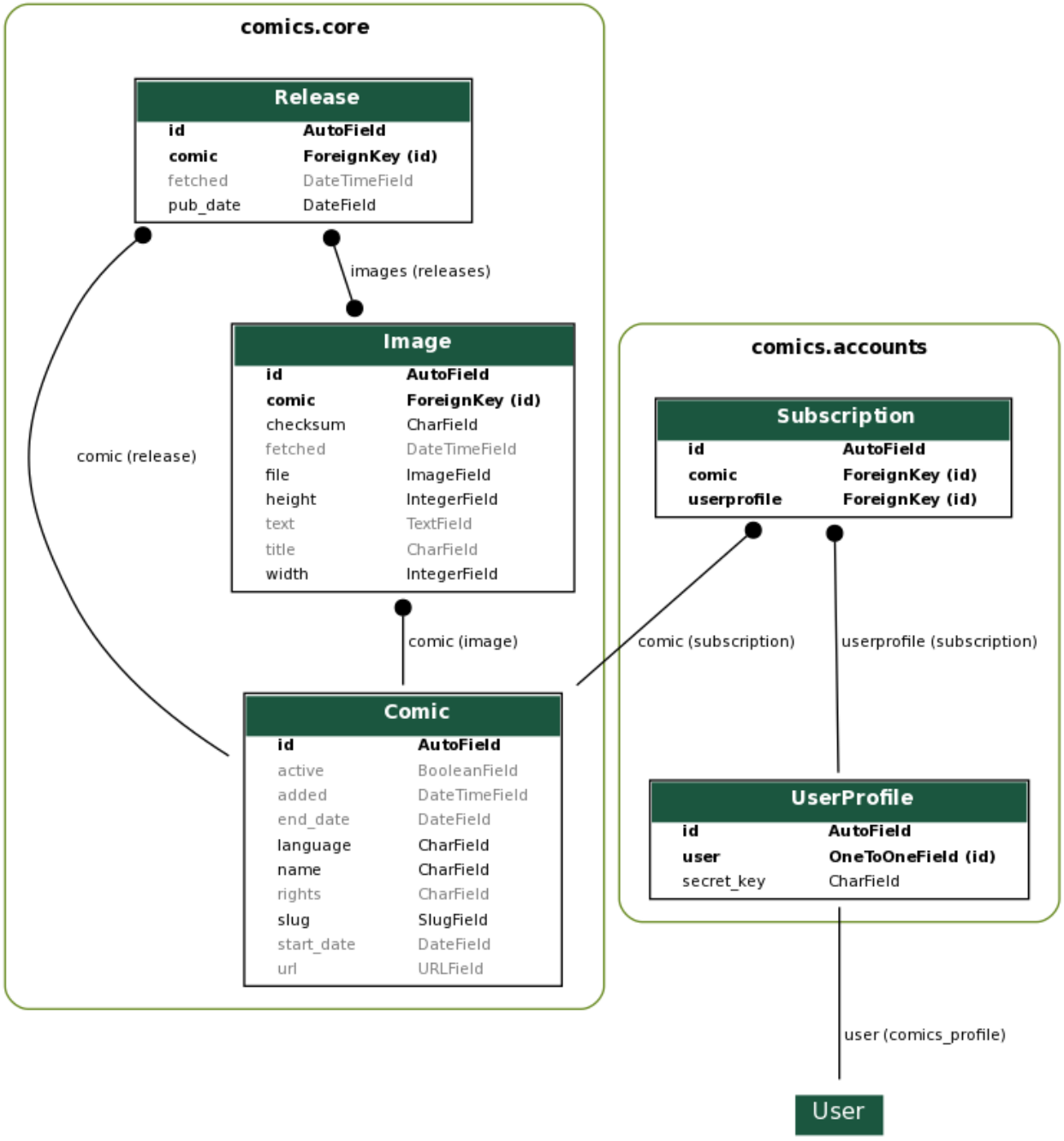
**Status Codes**

- 204 No Content – no error, and no content returned
- 401 Unauthorized – authentication/authorization failed
- 404 Not Found – subscription not found

## 1.6 Data model

Comics' data model is quite simple:

- The `comics.core` app consists of three models; `Comic`, `Release`, and `Image`.
- The `comics.accounts` app adds a `UserProfile` which add comic specific fields to Django's user model, including a mapping from the user to her preferred comics.



### 1.6.1 Database migrations

Changes to the data model are managed using Django's database migrations. If you need to change the models, please provide the needed migrations.

### 1.6.2 Updating diagram

The above data model diagram was generated using the Django app django-extensions and the following command:

```
python manage.py graph_models \
  --output docs/_static/data_model.png \
  --group-models \
  core accounts
```

## 1.7 Development environment

Comics development is coordinated through GitHub.

### 1.7.1 Testing

Comics got some tests, but far from full test coverage. If you write new or improved tests for Comics' functionality it will be greatly appreciated

You can run the tests with pytest:

```
pytest
```

To check test coverage, run with `--cov`:

```
pytest --cov
```

### 1.7.2 Code formatting

All code is autoformatted, and PRs will only be accepted if they are formatted in the same way. To format code, use Black:

```
black .
```

### 1.7.3 Linting

All code should be lint free, and PRs will only be accepted if they pass linting. To check the code for code quality issues, use flake8:

```
flake8
```

### 1.7.4 Run it all

To locally run all the same tests as GitHub Actions runs on each pull request, use tox:

```
tox
```

# PYTHON MODULE INDEX

## C

# HTTP ROUTING TABLE

## /api

# INDEX