

# Brown Dog Extractor How-to Guide (README.md)

## Introduction

This README describes how to develop and test a Brown Dog extractor tool, based on the template project in this repository. It also explains how you can contribute your new tool to the Brown Dog Tools Catalogue. Brown Dog Extractor Tools are used to analyse files and produce data points as a result. For example, one existing extractor uses the OpenCV software to process photographs, finding any human faces that are in a picture. You can find these and other extractor examples in the Brown Dog code repository. This repository contains a template project only. You can clone this repository to your local system and build it right away. However, until you add your custom extraction code the extractor will only report a tag of "Hello, World" for any given file.

## Brown Dog Runtime Environment

[system diagram](#)

The Brown Dog environment includes several services that work together to deliver the Brown Dog API. These include the Clowder web application, which hosts the API, a RabbitMQ message broker and extractor tools running on separate hosts. Brown Dog extractors are message-based network services that process data files in response to RabbitMQ messages. The client-facing Brown Dog API receives data from a client and passes messages to the extractor message topic. Any extractors that are subscribed to the topic will receive the message and decide for themselves if they can process the data, usually by looking at the MIME type. When an extractor starts or finishes working on a file it posts status and results back onto a message queue. All communication between the Brown Dog servers and the extractor tools is handled by messaging, except for ??? ANY EXCEPTIONS ???

## Docker

In general the Brown Dog extractors are developed to run within Docker containers, allowing them to be lightweight yet well described in a Dockerfile and well encapsulated along with their dependencies. This project contains a template Dockerfile that is ready to build and run within the Brown Dog environment. It can be customized to include any dependencies your extractor will need. The project also includes a docker-compose.yml file. This file can be used with the docker-compose command to create a fully functional Brown Dog runtime environment on your local machine. You can use docker-compose to deploy your extractor tool locally during development and testing.

## Step-by-Step Instructions

### Create Project and Runtime Environment

There are some basic software dependencies to install, after which Docker will handle the rest. Docker Compose is run from within a python virtual environment for your project. Docker Compose will start and connect several Docker containers together to create an extractor runtime environment. It will also deploy your own project code into a Docker container that is connected to this runtime environment.

1. Install prerequisite software. The install methods will depend on your operating system:
  - a. [VirtualBox](#) (or an equivalent Docker-compatible virtualization environment)
  - b. [Docker](#)
  - c. Python and PIP
2. Clone this extractor template project from the repository, substituting your extractor project name:

```
git clone https://opensource.ncsa.illinois.edu/bitbucket/scm/bd/bd-templates.git <extractor project name>
```

3. Create and activate a Python virtual environment for their new project:

```
a. cd <extractor project name>/bd-extractor-templates
   virtualenv .
   source bin/activate
   pip install docker-compose
```

NOTE: To activate this Python virtual environment in the future, repeat the "source bin/activate" step.

4. Start up the Extractor runtime environment using Docker Compose:
  - a. ~~TODO: docker-compose.yml for extractor runtime environment~~

```
docker-compose up
./tests.py
```

### Add Sample Files and Create Tests

At this point you have seen the template extractor deployed and working within your local runtime environment. Now it's time to add your sample input files and create the custom code. We recommend that you develop your extractor in a test-driven manner, by first adding input sample files, then modify the test script to validate the extractor results are correct.

1. Select a few representative sample files and add them to the "sample\_files" folder.

2. Edit the tests.py script to add tests for your sample files. You can remove the template example file and tests.
3. Run the new tests:

```
$ ./tests.py
```

4. The tests will fail of course, but you can look at the output logged to the console to see why they failed.

## Develop Extractor Code

Now that we have sample files and failing tests, we can start to write code to make those tests pass. You'll also presumably modify your test code too, as you learn more about your extractor output.

1. Edit extractor.py to add your data processing steps in the commented areas.
2. In particular, make sure you edit the MIME type filter (link to line) so that your extractor will only run on relevant input files.
3. Redeploy the code into the runtime environment by issuing a docker-compose command:

```
$ docker-compose ????
```

4. Run tests.py again:

```
$ ./tests.py
```

5. Repeat 1 - 3 until tests pass!
6. Try adding some more sample files and tests.

## Contribute the Extractor Tool to the Brown Dog Service

## JSON-LD Metadata Requirements

The DTS returns extracted metadata in the form of [JSON-LD](#), with a graph representing the output of each extractor tool. JSON-LD scopes all data to namespaces, which help keep the various tools from using the same keys for results. For example, here is a response that includes just one extractor graph in JSON-LD:

```
[
  {
    "@context": {
      "bd": "http://dts.ncsa.illinois.edu:9000/metadata/#",
      "@vocab": "http://dts.ncsa.illinois.edu:9000/extractors/ncsa.cv.caltech101"
    },
    "bd:created_at": "Mon Mar 07 09:30:14 CST 2016",
    "bd:agent": {
      "@type": "cat:extractor",
      "bd:name": "ncsa.cv.caltech101",
      "@id": "http://dts.ncsa.illinois.edu:9000/api/extractors/ncsa.cv.caltech101"
    },
    "bd:content": {
      "@id": "https://dts-dev.ncsa.illinois.edu/files/938373748293",
      "basic_caltech101_score": [
        "-0.813741"
      ],
      "basic_caltech101_category": [
        "BACKGROUND_Google"
      ]
    }
  }
]
```

If your metadata is a simple key/value map, without more depth or ordered elements, such as arrays, then submitting a plain JSON dictionary will be fine. Your metadata will be processed into JSON-LD on the server-side and tagged with your extractor as the software agent. All values returned by your extractor will fall in a namespace particular to your extractor

## Other Extractor Guidelines

- Handling failure gracefully
- How to use Key/Value pairs
- Different kinds of output data
- JSON-LD
- ??