. Documentation

Release

Matt Kenny

June 12, 2014

DataPlunger is a prototype ETL processing toolchain.

The goal is to create a modular package for the purpose of extracting data from multiple backing stores, performing n-number of transformational processing steps on those records, with the final output being loaded into a new format.

A workflow, or processing pipeline, is defined via a JSON configuration file containing the following information:

- Connection information to source data for processing.
- Processing steps to be applied to individual records extracted from source.

Source code for this project can be found at: https://github.com/mattmakesmaps/DataPlunger

Install Instructions:

```
# Create virtualenv
$ mkvirtualenv dp_dev_test
(dp_dev_test)$ cd /path/to/DataPlunger
# Install in development mode (sym-link to site-packages)
(dp_dev_test)$ python setup.py develop
```

CHAPTER 1

Configuration

Processing pipelines are described using a JSON configuration file.

1.1 Configuration File

A processing workflow is outlined using a configuration file. This JSON-encoded file contains the following elements:

1.1.1 Example Configuration File

An example code block is as follows:

```
{
    "type": "ConfigCollection",
    "configs": [
        {
            "name": "PeopleAndGradesConfig",
            "readers": {
                "Grades": {
                    "type": "ReaderCSV",
                    "path": "/Users/matt/Projects/dataplunger/sample_data/grades.csv",
                    "delimeter": ",",
                    "encoding": "UTF-8"
                },
                "People": {
                    "type": "ReaderCSV",
                    "path": "/Users/matt/Projects/dataplunger/sample_data/people.csv",
                    "delimeter": ",",
                    "encoding": "UTF-8"
                }
            },
            "layers": [
                {
                    "name" : "PeopleAndGradesLayer",
                    "processing_steps": [
                        {"ProcessorGetData": {"reader": "People"}},
                        {"ProcessorCombineData_ValueHash": {"reader": "Grades", "keys": ["name"]}},
                        {"ProcessorSortRecords": {"sort_key": "name"}},
                        {"ProcessorCSVWriter": {"path":"/Users/matt/Projects/dataplunger/sample_outpu
                            "fields": ["name","subject","grade","gender","age"]}}
                    ]
                }
            ]
        }
    ]
}
```

1.1.2 Parameters

Config Collection

`type` - Defaults to `ConfigCollection`. *In the future this may be expanded to also include a value of* `Config`.

`configs` - An array of individual config objects. *In the future, if* `type` *param has a value of* `Config` *, this parameter would not be necessary.*

Config Object

`name` - String. The name of the configuration. See `PeopleAndGradesConfig` in example.

`readers` - Object. Keys represent names of specific reader instances, values are objects containing configuration information for that specific reader instance. See `readers` in example.

`layers` - Array. Members are objects that represent an individual layer. A layer object contains a `name` parameter, as well as a `processing_steps` array. See `layers` in example above.

Readers

The `readers` object is composed of objects whose name represents an individual reader, and whose value is an object containing configuration information. The example above contains two separate named readers, `Grades` and `People`.

Each reader requires at minimum a populated `type` attribute. This attribute refers to the name of a Reader class found in `readers.py`. These are all subclasses of ReaderBaseClass. In the example above, both readers `Grades` and `People` have a type value of `ReaderCSV`, but point to different data sources (as seen in the `path` attribute). See doc strings within readers.py for additional reader specific configuration parameters.

Layers

`layers` map a series of user-defined processing steps to be performed against records output by one or more instances of a Reader class. The `layers` array in the example above contains a single layer element. Individual layers within the `layers` array are processed in the order in which they are defined. In the example, the layer element has a `name` of `PeopleAndGradesLayer`.

`processing_steps` - An array containing references to objects that represent instances sub-classed from ProcessorBaseClass. These processing steps are implemented on a per-record level. Each record output from a given Reader object is run through each Processor in the array, in the order defined by the array. See `dataplunger.processors.rst` for available processors and required configuration parameters.

CHAPTER 2

Main Modules

The DataPlunger package is broken down into three main modules, *dataplunger.core*, *dataplunger.processors*, and *dataplunger.readers*.

Core contains configuration and control code.

2.1 dataplunger.core

Processors perform actions on a collection records.

2.2 dataplunger.processors

Readers are responsible for creating a connection to a backing datasource, and returning an iterable that yields a single record of data from that datasource.

2.3 dataplunger.readers

CHAPTER 3

Test Coverage

Unit tests currently cover the `processors` and `readers` modules.

3.1 dataplunger.tests

3.1.1 dataplunger.tests.tests_processors module

3.1.2 dataplunger.tests.tests_readers module

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*