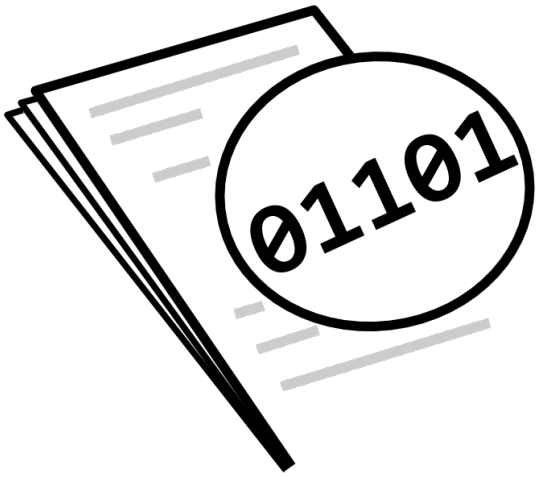
Pyodec Documentation

Release 0.0

Joe Young

July 12, 2014

Contents



Pyodec is a toolkit and a collection package for building and sharing codes to decode data files which are difficult or impossible to decode with other conventional automated file decoding processes. The simplest example is the decoding of human-readable ASCII text, but other files such as binary, compressed or even binary files with well-defined decoders can be decoded with pyodec.

All it requires is someone to write such a decoder.

Contents:

CHAPTER 1

Pyodec Introduction

Pyodec is intended for standardizing and sharing tools for decoding data files which cannot be efficiently read through automated methdods.

CHAPTER 2

Decoder Protocols & Rules

This outlines how data should be returned. This document is really all there is to pyodec. It is split into 3 sections, input, output, and library rules.

Though Pyodec is really a loosely connected set of independently written codes, it is important for the data model API to be compatible between different decoders. The goal of these rules will be to allow a single piece of code to take a diverse set of files (knowing their decoders), and both decode them and look at the data with the same code.

Basic python rules for internal consistency, variable and method naming, etc. should be followed.

This is python, so there are no private methods. Rewrite/reallocate any internal method as you please, but when doing so please try to follow these usage guidelines. If you find a better way to do the central decode method, then share it!

2.1 General

A decoder object is a class, which supports methods defined in the sections below relating to variables and a primary decoding function. It is essential that any decoder class supports decoding in the methods described, inputs as described, and handles variables in a very similar manner.

2.2 Input & client interfacing

Decoders should accept file path string, file object, and (where possible) string inputs to the decoder method as the first argument.

At a minimum, a decoder class should contain a decode method which can be run conforming to these input and interface protocols.

Decoders should produce a generator object, (usually using yield), and should allow the keyword argument generator control whether a generator or list is returned. generator should default to True.

2.2.1 Required `.decode()` keyword arguments

Argument	Type	Default	Purpose
`limit`	int	1000	Number of discrete data results to accumulate before yielding
`generator`	bool	False	Produce a generator based on limit, or a single set of values representing the entire dataset.

Warning Not all decoders at this time implement this protocol, and simply return generators at all times. These input requirements will evolve and expand over time (such as a requirement to handle certain keyword arguments properly)

2.3 Data output and returning

The possible range of outputs is far more diverse than the possible inputs, however there are a few goals that can be shared.

Discrete observations (rows) are collected in a list

Not the other way around. This is slightly less efficient with memory, but it is conceptually much simpler, and allows the second rule to actually work. .. highlight:: python

```
yield [(12,[4,4,4]), (12,[5,5,5]), ... ]
```

not

```
yield [[[12,12,...],[[4,4,4],[5,5,5],...]]
```

Note: This allows the returned data to be considered "records" as far as numpy is concerned, and can be easily converted into a very powerful recarray type simply with:

```
data = np.rec.fromrecords(data, dtype=decoder.get_dtype())
```

Iterative and procedural output are the same structure

Simply, you interact with a returned list of values in the same manner, whether you receive them from a yield or a return.

2.4 Variable and data descriptions

This is the least-defined rule of the library. Many data files are self-describing in some manner, and it is essential to extract this metadata from files.

The current procedure for a decoder class object to reveal the descriptions of the variables is through three methods with the following functionality.

Method Function .getvars() Return a list of dictionaries containing name, dtype and shape info, whose indices correspond to the index of the returned dataset (e.g. column names and descriptions). .get_fixed_vars() Return similar info to .getvars(), but includes the actual data elements as well, since fixed vars are not yielded with the rest of the data by default. .get_dtype() Return a valud Numpy-recarray dtype description, such that you could say import pyodec.files.myDecoder as dec import numpy as np for data in dec.decode(src):

> # convert data into a super awesome numpy recarray data = np.rec.fromrecords(data, dtype=decoder.get_dtype())

By default, the return values of these methods are defined by the VariableList and FixedVariableList class objects, but as always, the functionality can be overwritten in a decoder class when necessary.

As noted, these variable/metadata requirements are not set in stone, and will likely change through development with other users. Obviously backwards compatibility will become an issue pretty quickly, however.

> top

Naming conventions In accordance with python convention, classes will be in CamelCase, and everything else will be in lowercase with underscores. However, there is some inconsistency regarding naming of decoders, and decoder modules.

The current naming convention for the files that contain the decoders is to use lowercase and underscores where necessary.

This is up for debate.

CHAPTER 3

Main Library

This documents the library of shared codess

3.1 Pyodec Head Methods

Two methods are available at the root of the Pyodec package. Only one of them is acutally functional. Pyodec root functionality

`pyodec.`**`decode`**(*source*, *decoder*, **args*, ***kwargs*)
import and execute a file or string decoder on a certain class

`pyodec.`**`detect`**(*source*)
Currently non-functional

run every decoder we have on some amount of the source file, and return every decoder identifier which successfully read data from the chunk.

`pyodec.`**`download`**(*decoder*)
Currently non-functional

Download a decoder and install it on the local `Pyodec` installation

3.2 Core Classes

The classes defined in the `pyodec.core` namespace. These are used by developers to create new decoder objects, compliant with the Pyodec decoder rule set.

CHAPTER 4

Guide to devloping a Pyodec decoder

4.1 Building with Pyodec

4.1.1 Decoding ASCII files

4.2 Deploying and Sharing

Available decoders

CHAPTER 5

Indices and tables

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

p