Pyodec Documentation

Release 0.0

Joe Young

July 11, 2014

Contents:

CHAPTER

ONE

PYODEC INTRODUCTION

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

CHAPTER

TWO

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

CHAPTER

THREE

PYODEC CORE CLASSES

class `pyodec.core.`**`FileDecoder`**(*vars=False*, *inherit=False*, *fixed_vars=False*)
The inheritable class for a decoder of files.

`decode`(*filepath*, *generator=False*, *limit=1000*, ***kwargs*)
run the contained decode_proc as either a generator or a procedural decoder. If run as a generator, generator=True.

`decode_chunks`(*filepath*, *limit*, *begin=None*, *end=None*)
A "precompiled" generator-based decoder, to allow you to skip having to write the standard lines.

`decode_lines`(*filepath*, *limit*)
A precompiled generator-based decoder allowing line-decoding without having to write the standard modules - if the default options are all that are needed.

`decode_proc`(*filepath*, *limit*, ***kwargs*)
this should be a standardized function - defined by the decoder which takes a file path, and opens it, and calls read_lines or read_chunks and then returns the data those two functions produce.

Alternatively, you can not decode it, and use the default. But, it really won't work for most applications. Sorry.

`on_chunk`(*chunk*)
return a tuple from an observation – defined by the specific decoder. return False if the ob should be skipped

`on_line`(*line*)
return a tuple whose indices correspond to those of varlist. return False if the ob should be skipped

`read_chunks`(*yieldcount*, *gfhandle*, *begin=False*, *end=False*)
generator form of chunk reading

`read_lines`(*yieldcount*, *gfhandle*)
Read the file, and yield the # of obs as a generator

`yield_update`(*update*)
A reading process can throw updates if it wishes. The default self._throw_updates must be set to true.

class `pyodec.core.`**`FixedVariableList`**
Similar to a variable list, but much simpler, with fewer functions

class `pyodec.core.`**`MessageDecoder`**(*vars=False*, *inherit=False*, *fixed_vars=False*)
Just a wrapper for the decoder class, because message decoders can (and should) contain a varlist just as the main decoders

`decode`(*message*)
the decode method should be refactored, and used to decode a string message

class `pyodec.core.`**`VariableList`**

the requrements of the varaible list are somewhat strict, it must provide information regarding the names of the variables, their ranges, data conversions and units.

`addvar`(*name*, *longname*, *dtype*, *shape*, *unit*, *index=None*, *scale=1*, *offset=0*, *mn=0*, *mx=1*)

Add a variable to the variable list.

`dtype`()

This utility will produce the numpy recarray dtype entry for the pytable which will hold the data contained within.

This description could be used to create a recarray of the returned data.

To insert into pytables as a description, create the array with np.array([],dtype=decoder.tables_desc())

`get_index`(*varname*)

return the index of the variable with the name ‘varname’

`tables_desc`()

DEPRECATED: alias for self.dtype()

CHAPTER

FOUR

INDICES AND TABLES

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

PYTHON MODULE INDEX