EasyPDS4writer User Manual

DRAFT-A

Author: Diego Fraga Agudo

Date: 10th of September 2019

Table of Contents

1 INTRODUCTION

EasyPDS4writer is a Python 3 package to write PDS4 compliant products including both the data file and its label file. For more information about PDS4 standards see https://pds.nasa.gov/datastandards/about/ .

Currently the package is a proof of concept or prototype with limited functionality. Before using it in production please contact its author. Only products containing fixed width ASCII tables (Table_Character object) are supported so far.

2 PREREQUISITES

2.1 Dependencies

This package needs the following common python modules/packages:

- numpy

- ElementTree XML API (xml.etree)

It is pending to create a requirements.txt file.

2.2 Python version

This software has been developed and tested on Python 3.7 therefore Python 3.7 or higher is recommended. However it is believed to work on Python V3.4 or higher.

2.3 OS

The package is Operative System independent.

Development and main testing has been done on Windows whereas some limited testing has been done on Linux/MAC.

3 INSTALLATION AND TESTING

3.1 Installation

Download the package and run the provided setup.py in the same way you would run any other Python script in your environment.

3.2 Checking installation

From your systems command line type:

```
from easypds4writer.product_observational import ProductObservational
```

If it does not return an error then the package is correctly installed and can be found by Python

3.3 Running an example

The package comes with an example in the test directory. Run it as you would run in your system any other python script and it should generate two products in the test/output directory.

4 EXAMPLE OF USE

```
"""
This is an example of how to use the library to write a PDS4 compliant product
Consisting on two fixed width ASCII tables (using the PDS4 object Table_Character)
"""

from pathlib import Path
import os

from easypds4writer.product_observational import ProductObservational

test_path = os.getcwd() # The test path is the path where this file is located.

# Create a PDS4 product object and set a label template for it. The label template must
not contain the
# File_Area_Observational which is generated automatically and transparently by the
tool including all necessary tags
# and its values. If an empty template is used the products will still be readable by
readPDS python package but will
# not be fully PDS4 compliant.
pds4_product= ProductObservational(Path(test_path +
"/example_templates/example_template.xml"))

# Now we will configure the tool to write a specific product type.

# Tell the tool that the product type has one fixed ASCII table called table_test
table_character= pds4_product.declare_table_character("table_test")

# Define (declare) the fields in the table. For each field it must be specified its
format, name, units and a
# description. currently it is also necessary to provide the data type but the
intention is to determine this
# automatically from the format except for data types of string type where it is not
possible
# (Correct handling of UITF8 is pending!! Eg. ÉΦ is not written properly).
table_character.declare_field("%-23s",
                              "ASCII_Date_Time_YMD",
                              "PARTICLE DETECTION IN UTC",
                              "N/A",
                              "TIME OF GIADA DETECTION IN UTC")
table_character.declare_field("%6s",
                              "UTF8_String",
                              "DETECTING SUB-SYSTEM",
                              "N/A",
                              "THE SUB-SYSTEM OR SUBSYSTEMS WHICH DETECTED THE
PARTICLE")
table_character.declare_field("%8.3f",
                              "ASCII_Real",
                              "PARTICLE SPEED BY GDS+IS",
                              "METER/SECOND",
                              "PARTICLE SPEED MEASURED BY GDS+IS")


# Define second table in the product
table_character2= pds4_product.declare_table_character("table_test2")

table_character2.declare_field("%-6.6s",
                               "UTF8_String",
                               "DETECTING SUB-SYSTEM",
                               "N/A",
                               "other")

table_character2.declare_field("%5d",
                               "ASCII_Real",
                               "PARTICLE SPEED BY GDS+IS",
                               "METER/SECOND",
```

```
                                    "PARTICLE SPEED MEASURED BY GDS+IS")

# End of product type definition.

# Now create the first instance of the product we want to write...
pds4_product.new_product(Path(test_path + "/outputs/phys20160421t000000m_v1_0.tab"))

# Now fill in the first table with data. The inputs you past will be validated against
the definition given
# in declare_field and exceptions will be raised if do not match.
# Typically this will be in a loop with one iteration per record (row) in the table.

record = ("2016-04-21T03:31:34.958", "GDS-IS", 1.35)
table_character.add_record(record)

record = ("2016-04-21T08:39:07.969", "GDS-IS", 10.83)
table_character.add_record(record)

record = ("2016-05-22T08:39:07.969", "GDS-I", 123.56)
table_character.add_record(record)

record = ("2017-05-22T08:00:00.000", "GDS-I", 123.5678)
table_character.add_record(record)

record = ("2017-05-22T08:00:00.000", "GDS-I", 123.67)
table_character.add_record(record)

# Fill in the second table

record = ("GDS-IS", 5)
table_character2.add_record(record)

record = ("GDS-IS", 12345)
table_character2.add_record(record)

# All metadata outside the File_Area of the label cannot be filled in automatically.
You must provide a template for
# this part (done when creating the ProductObservational object) and now replace the
place holders by the desired values.
pds4_product.set_metadata("$year", "2017")
pds4_product.set_metadata("$mission_name", "imaginary mission")

# Close_product method writes the label file (the data file was written already in each
add_record call), closes data
# files etc and leaves the pds4_product ready to call new_product.
pds4_product.close_product()

# We can reuse the product type definition and write a more products by doing again the
same...
pds4_product.new_product(Path(test_path + "/outputs/phys20160421t000022m_v1_0.tab"))

# Now fill in the two tables with data
record = ("2018-04-21T03:31:34.958", "GDS-IS", 2.34)
table_character.add_record(record)

record = ("2018-04-21T08:39:07.969", "GDS-IS", 11)
table_character.add_record(record)

record = ("2018-05-22T08:39:07.969", "GDS-I", 27.5)
table_character.add_record(record)

record = ("2018-05-22T08:00:00.000", "GDS-I", 156.36)
table_character.add_record(record)

record = ("2018-05-22T08:00:00.000", "GDS-I", 2.0)
```

```
table_character.add_record(record)

record = ("GDS-AA", 6)
table_character2.add_record(record)

record = ("GDS-AA", 67890)
table_character2.add_record(record)

# Replace variables place holders by the desired values
pds4_product.set_metadata("$year", "2018")
pds4_product.set_metadata("$mission_name", "imaginary mission")

# Close the product
pds4_product.close_product()

# You can reuse pds4_product object to write as many products as you want byt you have
to create a new object to write
# products of a different type.

print ("You should have two products in the output directory that in this example is "
+ str(Path(test_path + "/outputs/")))
```

5 FIELD_FORMAT

Method table_character.declare_field will ask you for a field_format that is explained here.

The filed_format specifies the format of the fields including the width of the field, if it is a sting or a number, its precision etc. The width of the field (or column width) is the maximum size in characters that a value can occupy.

The best way to understand the field_formt is through examples. See the following table.

Field_format	Meaning	Examples of result
%-25s	Left justified string of up to 25 characters.	This 25 characters string Shorter string
%3d	Integer number with up to 3 digits	123 12
%4.1f	Real number with 1 decimal and up to 4 characters in total counting the dot .	47.2 8.1
%8.2e	Real number in scientific notation with 2 decimals and 8 characters long counting everything.	1.23e+04 5.97e+02

The tool will return an error if the user attempts to write a value that is not compatible with the field_format. This ensures that what is written in the file actually complies with the data format declared in the label eliminating a common source of errors in the products.

The following full description of field_format is extracted from the Planetary Data System Standards Reference.

The formation rule for a <field_format> value is:

%[+|-]width[.precision]specifier

where square brackets indicate an optional component, “%”is the percent sign (which must precede every field format value), and:

[+|-] denotes either a "+" or "-", but never both. The "-" may be used for string fields, to indicate that the string is (or should be) left-justified in the field. This is actually the preferred way to present most string values in character tables, so the <field_format> value for fields with a data type of ASCII_String will nearly always begin with a "-". Similarly, the "-" denotes left justification for any of the date/time type fields. The "-" prefix is forbidden for all numeric fields (integers, floating point numbers, and numbers using scientific notation). The "+" may be used with numeric fields to indicate that an explicit sign is included in the field for input, and should be displayed on output. In PDS4 labels, the "+" is forbidden for string fields.

width is the potential total width of the field — i.e., the width of the widest value occurring in the field. width is an integer indicating the maximum number of characters needed for the complete representation of the largest (in terms of is display bytes, not necessarily magnitude) value occurring, or potentially occurring, in the field. This should include bytes for signs, decimal points, and exponents. In the case of string values, it is the maximum width from the first non-blank character to the last non-blank character. It does not include bytes for field delimiters or double quotes (“) around character strings, which are not considered part of the field. In character tables, it must be the same as <field_length> for scalar fields.
Width is separated from precision by a decimal point (“.”). If there is no precision specified, the decimal point must be omitted.
Precision is the number of digits following the decimal point for real numbers (but is otherwise ignored).

precision is used in three different ways:

1. For real numbers, it indicates the number of digits to the right of the decimal point.
2. For integers, it indicates that the integer will be zero-padded on the left out to the full field width. For example, the value "2" in "%3.3d" format is "002".
3. If precision is included for a string format, it must be equal to width.

specifier:

d indicates a decimal integer
o indicates an unsigned octal integer
x indicates an unsigned hexadecimal number
f indicates a floating point number in the format [-]ddd.ddd, where the actual number of digits before and after the decimal point is determined by the receding width and precision values (note that width includes the decimal point and any sign).
e,E indicates a floating point number in the format [-]d.ddde+/-dd or [-]d.dddE+/-dd respectively where "+/-" stands for exactly one character (either "+" or "-"), there is always exactly one digit to the left of the decimal point, and the number of digits to

the right of the decimal point is determined by the preceding precision value (note that the width includes all digits, signs, and the decimal point).

s indicates a string value. Note that strings should generally be left-justified in fixed width character tables and on output from a binary table, so most <field_format> values ending in "s" should begin with "-".

EasyPDS4Writer characteristic
In scientific notation EasyPDS4Writer does not allow not having any decimal. It is unclear if this extreme case is allowed in PDS4 or not but it is forbidden in EasyPDS4Writer for better compatibility. For example, use %9.1E with one decimal instead of %9.0E without decimals.

6 DATA_TYPE

Method table_character.declare_field will ask you for the data_type. The intention is that future versions of the tool will determine automatically the data_type from the field_format and by doing so it is guarantee that both are consistent. This is not possible for formats of string type where the user will still need to enter the data_type.

To see the accepted values for data_type consult the data_type in the PDS4 dictionary at https://pds.nasa.gov/tools/dd-search/

7 AUTHOR

This tool is being developed by Diego Fraga Agudo. It is not an official ESA software.

Contact email: dfraga@sciops.esa.int

The tool is being developed in a best efforts basis only.

8 WANT TO PROVIDE FEEDBACK OR CONTRIBUTE?

You are very welcome, please contact the author.

The author is interested on help with testing on different platforms and Python versions. Bugs reports, development priorities and ideas are welcome. If you want to contribute with code, please contact the author to discuss about it.

Also if you are interested on the tool as a potential user you are welcome to contact the author.