EPSC Abstracts
Vol. 13, EPSC-DPS2019-1416-1, 2019
EPSC-DPS Joint Meeting 2019
© Author(s) 2019. CC Attribution 4.0 license.

EPSC
EPSC-DPS Joint Meeting 2019
15–20 September 2019 | Geneva, Switzerland

Concept of a PDS4 product writer library and a python demonstrator

Diego Fraga Agudo
ESAC, Camino Bajo del Castillo s/n, Urb. Villafranca del Castillo, 28692 Villanueva de la Cañada, Madrid, Spain
(dfraga@sciops.esa.int)

1. Introduction

We present the concept of a library to write PDS4[1] products, together with a working demonstrator in Python, which could evolve into a reusable PDS4 writer Python package that would be provided to the community as open source.

The proposed tool is a PDS4 product writer meaning that it generates both the data file (digital object) and the PDS4 label file, as opposed to a PDS4 label only writer. The inputs of the tool are the data that are in memory that have to be written in files, in contrast to tools that take as an input already existing data files and construct a label for them. To our knowledge there is not any open source tool like this.

2. Motivation (why this is needed)

The goal is to facilitate the generation of reliable PDS4 compliant products, hiding to the data producer much of the complexity of the physical representation of the data and the metadata (label) and ensuring the consistency between the two. This not only facilitates the generation of the products but also its validation since the data producer relies on an already validated piece of SW that encapsulates a common reusable functionality.

One of the advantages of relying on such a library is the prevention of errors. During the experience the author has of validating Rosetta mission data as an ESA Archive Scientist, it was found that errors that could have been prevented by using a library like this were very common. Rosetta is a PDS3 mission but equivalent errors can happen in PDS4. These errors can be classified into three categories:

- Inconsistencies between the label and the data: For example an incorrect value is written in the offset attribute of the label indicating that the data object begins in a byte number different from the actual byte number in the data file. These type of inconsistencies are extremely common and might sometimes remain unnoticed until someone attempts to use the label to decode the data.
- Inconsistencies within the label: For example a number of table fields (columns in a table) is declared in the fields attribute but a different number of fields is defined with the Field Character objects. These type of errors are common.
- Errors in the data: there is an error while dumping the data in the file and the data are not written as the producer expected. These errors exists but are more rare and are often corrected in early stages.

The consequences of any of these errors can be to make the data unreadable by PDS4 aware readers. Also if data users attempt to write their own reading SW using incorrect information on the label they can encounter serious difficulties.

Of course such a library can have bugs that introduce errors, but it needs to be debugged only once and reused by any data producer.

It can be argued that the use of schemas and schematrons together with validation tools solve all issues but they have some limitations:

***Limitation of XML schemas and schematrons*.**
While the advantages of using XML schemas and schematron are enormous they do not guarantee by themselves the correctness of a product. Validating a label against its schemas and schematron can only tell whether the label is consistent with its schemas and schematrons or not. It does not tell anything about the consistency of the label with its data file.

***Limitation of Validation tools*.**

It is not possible to build a validation tool that detects all errors. Some errors are not detectable by common SW because there is nothing to check against. For example, an image stored as an array could have been written using unsigned integers but the label could incorrectly indicate that the values are signed integers. A validation tool cannot detect this error, the only thing it can do is to trust what is declared in the label. However if the image is displayed using the information from the label the image will be completely corrupted.

Another characteristic of validation tools is that they can detect issues whereas the proposed library aims to prevent them.

A library like the proposed one can also increase the homogeneity between products and promote or even enforce PDS4 recommendations and best practices.

The next section explains a specific implementation of the concept while other implementations, for example in other languages, are also possible.

The Python demonstrator

A demonstrator with limited functionality has been developed in Python 3.

The library generates both the data file (digital object) and its PDS4 label file. It generates completely automatically and transparently to the data producer the File Area part of the label where the data format is defined and provides a mechanism to fill in easily the remaining information in the label, which are specific for each product or mission and cannot be automatized.

Currently the Python implementation is a demonstrator covering only Observational Products consisting of any arbitrary number of fixed width ASCII tables (PDS4 Table Character objects). It can be extended in the future to cover all PDS4 data structures and most PDS4 capabilities. The current plan is to release the demonstrator as open source software before the congress date.

References

[1] JPL (2018). *Planetary Data System Standards Reference.* pds.nasa.gov/datastandards