Table of Contents

Synopsis

This project will create an open source, Python-based tool capable of reading the vast majority of data (including tables and images) and metadata formats currently archived by the Planetary Data System (PDS), as well as some formats common in planetary science research workflows but not typically archived in the PDS (e.g. ISIS cube, JP2, GeoTiff). Python is currently the most popular programming and data exploration language in research, including planetary science and astronomy, but its support for planetary data remains poor. The Planetary Data reader (*pdr*) will offer users a consistent, easily-understandable interface that works the same way for all data formats. For instance, a simple command like `read(filename)` will return a Python data object that can immediately be used in research workflows or easily converted to other formats.

This project solves a well-known pain point for planetary data users: simply figuring out how to access data. This is a particular problem for data archived under the Version 3 standards of the Planetary Data System ("PDS3"), which often have inconsistent or bespoke formats. The PDS is in the process of migrating archives to the stricter PDS4 standard, but the timeline for this migration is uncertain, and it is unlikely to be completed before 2025. The *pdr* tool developed under this project provides a stopgap solution and also supplements the migration effort in two important ways. First, reading PDS3 data formats is one limit on a swift and efficient data migration, so this tool will streamline that effort. Second, because this tool will treat PDS3 and PDS4 data identically from the end-user perspective, workflows developed with either PDS3 or PDS4 data will stay the same before and after the migration.

Description of design and functionality

Basic functionality and design concepts

+ The user will not be required to specify the file type or format. The software will infer this from the data. However, there will be an optional parameter to specify file type / format to easily deal with situations where the inference might fail.
+ In cases where data and metadata are detached from each other (e.g. detached headers in PDS3, detached XML labels in PDS4), the user will only be required to specify the location of one of these files; the software will attempt to infer the location of the other. This is possible because the detached data and metadata usually have the same name with different extensions and reside in the same directory. For tables with detached format definition files, the software will also attempt to automatically locate those. In both cases, there will be a fallback option to manually specify data and metadata.
+ The tool will return simple Python objects containing relevant data and metadata. The returned objects will support standard methods of inspection to assess structure and content. Observational data will be returned as a basic Python object type (list, dict, string, etc.) *or* as *numpy* array objects *or* as Pandas dataframes.
+ Memory management. The size of individual observational data files has become very large, and we expect that trend to continue. The *pdr* will incorporate some standard features for memory management when working with large files. This will include *lazy loading* and *memory mapping*. Users will be able to examine the structure and metadata of a file without loading the entire file into memory. They will also be able to easily

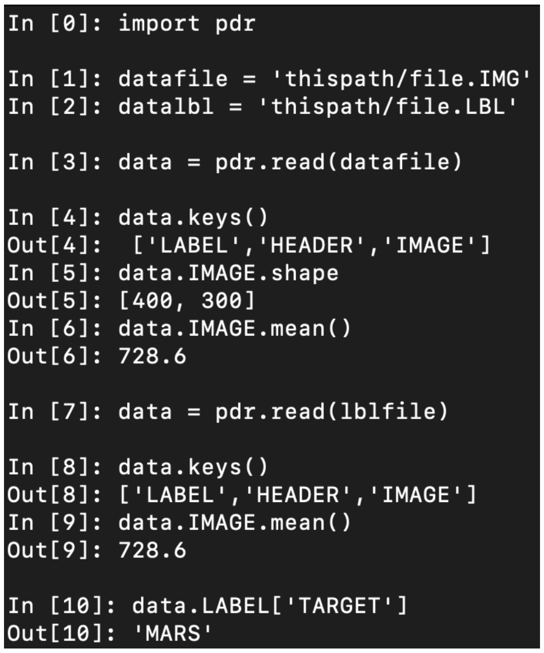
choose to load subsets of data (like image subframes) into memory rather than the entire file. These features are enabled by the structured format of PDS3 data files and are also supported by the standard libraries for working with FITS data.

+ Data writing (intentionally not implemented). The software will not have its own functionality for writing Python data objects to other formats (e.g. CSV or FITS). The requirement that Python data objects be of standard types means that it will be easy to leverage existing and very mature libraries for those capabilities, like the *fits* package in *astropy* or the CSV functionalities of *Pandas*, a core library for data analysis in Python.
+ GUI (intentionally not implemented). The tool created by this project will interface entirely through command line prompts in a Python interpreted environment. It will not include a graphical user interface (GUI). The development of GUI tools is a natural follow-on to this project. In fact, this project is a necessary prerequisite to GUI development; **the reason that Python-based GUI tools do not yet exist is because they require *pdr*.**

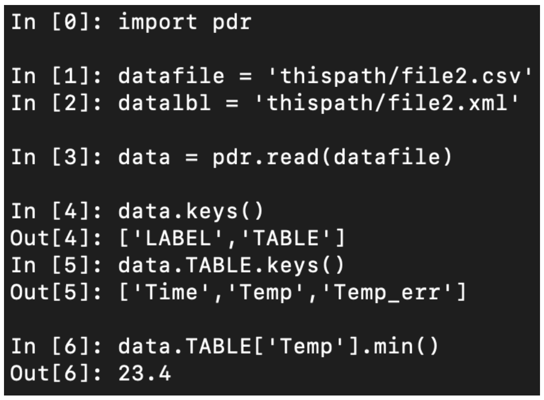
Example workflow

The *pdr* tool will be designed to have almost no learning curve for users with basic Python proficiency. Users will simply import the module and then use a "read()" function to read local data files. Users will be able to specify *either* the observational data file or any detached metadata label (e.g. a LBL or XML file), and the software will infer the name and location of the other. The software will also automatically attempt to locate any other detached metadata format files (e.g. FMT) that may be required. The software will then return an in-memory object that contains all the observational data and metadata. These objects will have standard Python data types (including *numpy* arrays and *pandas* DataFrames) and so will immediately be usable in the huge Python data analysis ecosystem.

The figures to the right illustrate example workflows for reading data and deriving simple information about those data. These are mock-up workflows but very closely reproduce the behavior of actual prototype software that this team has already developed. The top figure demonstrates an example of reading image data archived under PDS3 standards. The user simply specifies the datafile and is given a data object that contains the image data (as "IMAGE"), the label data (as "LABEL"), and, in this case, also an image header



```
In [0]: import pdr

In [1]: datafile = 'thispath/file.IMG'
In [2]: datalbl = 'thispath/file.LBL'

In [3]: data = pdr.read(datafile)

In [4]: data.keys()
Out[4]:  ['LABEL','HEADER','IMAGE']
In [5]: data.IMAGE.shape
Out[5]: [400, 300]
In [6]: data.IMAGE.mean()
Out[6]: 728.6

In [7]: data = pdr.read(lblfile)

In [8]: data.keys()
Out[8]: ['LABEL','HEADER','IMAGE']
In [9]: data.IMAGE.mean()
Out[9]: 728.6

In [10]: data.LABEL['TARGET']
Out[10]: 'MARS'
```



```
In [0]: import pdr

In [1]: datafile = 'thispath/file2.csv'
In [2]: datalbl = 'thispath/file2.xml'

In [3]: data = pdr.read(datafile)

In [4]: data.keys()
Out[4]: ['LABEL','TABLE']
In [5]: data.TABLE.keys()
Out[5]: ['Time','Temp','Temp_err']

In [6]: data.TABLE['Temp'].min()
Out[6]: 23.4
```

data structure (as "HEADER"). The user can then immediately perform analysis on the data or metadata, e.g. computing the mean of the image pixels or querying specific key values in the label. The workflow is exactly the same whether the user specifies the label file or the image file, because their file paths and names are the same (except for the extension), which is standard in PDS archives. The bottom figure demonstrates a different example -- table data archived under the PDS4 standard.

<u>Categories of known data issues</u>

+ *Custom or unusual compression schemes*. The Mastcam and Clementine uncalibrated (Level 0) engineering data records (EDR) are both stored in bespoke compression formats that cannot be parsed without the use of software that both instrument teams included in their archives. In both of these cases, the software is implemented in C which must be compiled in order to work properly, and we consider both to be at high risk of becoming unusable or difficult to use due to changes in common computer operating environments, a process called "bit rot." In any case, the use of a custom compression algorithm that is only documented in C source makes the data difficult to use in research workflows. We will port / re-implement the provided compression algorithms as Python libraries, transparently used by *pdr*. When we are able to successfully run the original software---for example, Mastcam and Clementine EDR data---we will verify the accuracy of our port with regression testing. Otherwise, we will judge success from inspection.
+ *Malformed data or metadata*. Much of the metadata stored in PDS is structured according to the rules of the Parameter Value Language (PVL). However, considerably more often than one might wish, it is not structured *strictly* according to those rules. Ross Beyer et al. have written a Python tool called *pvl* which will parse these structures, but only if they are structured correctly to begin with. We know that there will be a lot of PVL-esque metadata that will only be readable if we either create exception handling in *pdr* or submit changes in exception handling to *pvl*. Some of these situations are known and documented, but probably many more will be discovered. We also expect that there will be cases where the data themselves are systematically malformed owing to some quirk of the data production process that was not caught before archiving; again, we will have to handle these as case-by-case exceptions.
+ *Corrupted data or metadata*. Although it is the core mission of the PDS to preserve the integrity of data over time, and they do an outstanding job of that, there is still certainly some corrupted data in the archive that no one knows about. Almost all of this data was probably corrupted before it was ever submitted for archiving, and the corruption was simply not detected in the archival process. If and when we encounter data corruption, we will make a reasonable but limited effort to work around the issue, but otherwise just log that the data is unreadable and alert the appropriate PDS node. Data recovery *per se* is not in the scope of this project; neither is quality assurance of PDS archives.
+ *Compound file types*. Many files archived in the PDS3 "image" format, especially for older missions, contain multiple data structures and types within the same file. An example would be a file that contains both an image-like object and a table-like object. The files sometimes contain half a dozen different data types. Our prototype version of

pdr already implements a successful solution to this problem: we parse the file metadata for pointers that define the location and type of data objects, and then read those objects out individually. Each data type is included on the data object returned by the tool. We are not aware of any other software tool that works for these files. We note that this is a case in which even the *PDS Transform Tool* fails, according to its online documentation.[1]

+ *Remotely located table data definition files*. Some table-style data archived under PDS3 have their formats defined in metadata files that are not colocated with the data themselves. For such data, the colocated metadata (e.g. the header file) will simply reference a file name (e.g. "TABLE.FMT") but not a location. These format files are usually located somewhere else in the archive directory structure, but the location is, of course, not standard. *Pdr* will remove the need for users to find or acquire these files to use the data. We will make that possible by generating a table of the locations of these reference files on a per-dataset basis. When a user attempts to open a table that references an external format file, *pdr* will locate the correct format file and acquire it (via internet connection) for use. There will be a way for users to manually specify this file if desired. This issue is solved in PDS4, which requires that tables be completely defined in the colocated XML header. Data are no longer being archived under PDS3 standards, so our solution for PDS3 data will never need to be updated.

Impact, demand, time-criticality, and uniqueness

<u>Summary of the state of the art, including prior art</u>

There have been a lot of efforts to create general-purpose planetary science data readers.

+ PlanetaryImage[2] was a prototype PDS image reader developed by Austin Godber as part of the original PlanetaryPy Project. He has since made a career change and relinquished rights to the PlanetaryImage and PlanetaryPy "namespaces" and code. The PlanetaryImage code served as a basis for our own prototype development of *pdr*, but it was never realized as a general solution and should be considered deprecated.
+ The PDS Transform Tool is in development by the PDS Engineering node to help translate data from PDS3 to PDS4. It is Java-based, which makes it difficult to use in Python workflows. It is also still in development and has major missing functionality (such as reading bespoke compressed data files or PDS3 image files that contain multiple data types).
+ *pds4_tools*[3] is a Python library created by Lev Nagdimunov on behalf of the Small Bodies Node to read PDS4 data of all types. It works really well! It essentially solves the problem of reading data already archived under PDS4. We will fully incorporate this tool, but wrap the input and output functionality to match our idiomatic framework.
+ *plio*[4] is "[a] planetary surface data input/output library written in Python" by the USGS. It is updated infrequently and has minimal documentation, with no list of supported file types that we could find. It appears to support Chemcam, TES, and M3.

[1] https://nasa-pds.github.io/transform/
[2] https://github.com/planetarypy/planetaryimage
[3] http://sbndev.astro.umd.edu/wiki/Python_PDS4_Tools
[4] https://github.com/USGS-Astrogeology/plio

+ pysis[5] is a Python wrapper for ISIS3 that supports the reading of "cube" (fmr. "qube" or "qub") files and headers, as well as other (many) functions of ISIS3. We have had success reading cube files and labels as structured binary arrays and PVL data respectively, so wrapping this wrapper will not be necessary.
+ The Geospatial Data Abstraction Library (GDAL)[6] is perhaps the most widely used piece of software in geography. It is “a translator library for raster and vector geospatial data formats." Although it increasingly supports planetary data types and conventions, its primary user base is terrestrial and its functionality is mostly directed towards that purpose. The planetary data that it supports tends to be higher-level derived and mapping data products. For completeness, we plan to wrap the functionality of GDAL for these filetypes to fit our input / output idioms.
+ NASAView[7] is a GUI-based data display tool that is created and maintained by the PDS. It is not clear from available documentation what data it does or does not support, but it is clear that it is intended as a quick-look tool and not as a component of an exploratory data analysis / programming workflow.
+ VICAR[8] (Video Image Communication And Retrieval) is an image processing software suite developed and maintained by JPL in support of past and current mission data processing. It is remarkable software and may be the oldest continuously-maintained image processing software in the world. JPL has made the source code available. It could be used to read the VICAR format of data and metadata. However, we have had success reading these classes of data in pure Python and do not anticipate needing to wrap or use this software.
+ *rasterio*[9] is a Python library for reading and writing raster-based geospatial data formats (like images). We have found that it has a remarkably high success rate parsing image data archived under PDS3, including the IMG, ISIS3, and VICAR formats. We will wrap it to match the idioms of *pdr*. This library already wraps GDAL, so we will probably only need to include *rasterio*, between the two.
+ *sbpy*[10] is an Astropy affiliated package for small-body planetary astronomy. Its development was partially supported by a prior PDART grant. It is a data processing and analysis package that does not include observational data I/O as a major component.

Other Extant Tools

This is just a basic summary of the Python libraries that we know exist, are relatively well-constructed and maintained, and are currently in use in planetary research workflows. Many, many developers have engaged in parallel or overlapping efforts in this domain, and a complete list of all planetary science Python libraries is impossible to compile. The fact that this list is not exhaustive does not change the general points that we are very aware of the current

[5] https://github.com/wtolson/Pysis
[6] https://gdal.org/
[7] https://pds.nasa.gov/tools/about/pds3-tools/nasa-view.shtml
[8] https://github.com/nasa/VICAR
[9] https://rasterio.readthedocs.io/en/latest/intro.html
[10] https://github.com/NASA-Planetary-Science/sbpy

landscape of Python tools for reading PDS data, and the large variety of tools is itself a barrier to use while also not covering all reasonable use cases of planetary data. The tools that we listed do not overlap very much in capabilities or scope, so the specific tool or combination of tools needed to read any specific data can be completely opaque to users who are not already deeply familiar with the landscape of tools and data formats. To the extent that the tools described do have overlapping functionality, this fact provides just a hint of the tremendous duplication of effort that results from the lack of a gold-standard solution. Further, each of these tools has its own idioms and interfaces, so for each tool a user wishes to adopt, they must relearn. We think that reading planetary science data into the *lingua franca* of modern research (Python) should really be as conceptually easy as double-clicking a file. **It should just work.**

We are also aware of tools that were developed for a time but have apparently been abandoned. One example that often comes up immediately after someone types something like "pds data read python github" into a search bar is the *PyPDS* tool.[11] Co-I Aye was a contributor to this project, but it was last updated in January 2014 and we consider it abandoned. One of its critical dependencies is the Python Image Library (PIL),[12] which has been abandoned, fully deprecated with Python 2.x, and replaced in all modern Python distributions with the successor project *Pillow*. The *PyPDS* software does not carry an explicit license, making it not truly "open source," because software is by default not licensed for derivative works or redistribution; we also know from personal communication with the author (R. Balfanz) that he is not interested in adding a license. It is also not actually a general solution to working with PDS3 data, despite what the limited documentation might lead someone to believe.

There are dozens of examples like this. GitHub repositories are often created with README files that describe more ambitious capabilities than are ever actually implemented. Projects often fall short and are abandoned by their developers. This is entirely normal and acceptable for volunteer, hobby, and peripheral work projects, whose developers have no responsibility to deliver any product, including documentation. However, it means that the only way to understand what features they really implement is to read the source code or, better, actually use the tool.

We also know of a lot of stopgap tools that individual researchers have written to solve immediate project needs, e.g. software or scripts that read a particular subset of data from a particular instrument in service to a narrowly focused research task. Even if this software were available to us, which it largely is not, we think that the effort of gluing it all together would exceed the effort of just starting over from scratch, and the result would be worse. We think that nobody has written a general-case tool because, as we outline in this proposal, the problem is too big to accomplish through uncoordinated half-measures or volunteer labor. It requires a focused effort with a coherent vision over thousands of work-hours. Much as *astropy* remained a relatively immature tool until it gained a kind of institutional support (in that case, by being deemed "mission critical" to the James Webb Space Telescope) [E. Tollerud, *priv. comm.*], **nothing like the Planetary Data Reader will arise organically from volunteer effort** in the community. This work does not substantially overlap with efforts that the PDS is currently

[11] https://github.com/RyanBalfanz/PyPDS
[12] https://en.wikipedia.org/wiki/Python_Imaging_Library

undertaking, and the attached letter of commitment from the PDS ENG node director signals our intent to continuously work with PDS to avoid duplicated effort.

Summary of results from the Python in Planetary Science Survey

The proposal team, along with other community members, conducted a survey in 2019 about Python use and software needs in the planetary science community.[13] Though respondents were primarily self-selected in response to a solicitation in PEN, preliminary results (48 responses) support our intuitions about community software needs. **Python is the most commonly used scientific language among our respondents.** About two-thirds regularly use Python in their scientific work, and even some respondents who do not regularly use Python make sporadic use of some Python packages, particularly numpy, pandas, and scipy. Runner-up languages were IDL with about a third of respondents and Matlab with about a quarter of respondents. Numpy and scipy use are nearly ubiquitous among regular Python users, and pandas use is extremely common (about 80% of users).

These results support our contentions that software package development efforts that seek to serve the maximum number of stakeholders in the planetary science community should be targeted at Python, and that APIs that interface readily with numpy, scipy, and pandas idioms and data structures are likely to have the easiest learning curve for potential users.

The survey results also suggest that our proposed development effort would address common planetary science software pain points. Over half of our respondents reported that PDS data ingestion or metadata handling served as major time sinks. Another 10 percent noted that ISIS integration was also a burden. Over half of our respondents also reported spending a great deal of time on tooling for data exploration. *pdr* would directly address this category of issue by providing tools for making data ingestion easier.

In addition, **about half of Python users reported that their Python experiences suffered from poor integration with data sources or had difficulty handling planetary science data formats.** Tellingly, one respondent who identified as a college educator simply listed "opening up new data" as a major problem in teaching Python to student geoscientists. Several non-Python users also listed issues of this type (such as reading and writing CDF files), presumably as reasons they avoid Python. Finally, roughly half of our Python-using respondents – and some of our non-Python-using respondents – reported problems with high learning curves or generally poor user experience. Respondents often linked these issues to packaging and software distribution (“dependency hell”, “feels like beating your head against a wall”), library interoperability, and simple software discovery. The *Planetary Data Reader* directly addresses these issues.

Work Plan

This effort will build on proof-of-concept work already undertaken by this team.[14] Data are currently archived in PDS under both the older PDS3 standard and the current, stricter PDS4 standard. There are plans for migration of the entire archive from PDS3 to PDS4, but the

[13] https://github.com/MillionConcepts/software-survey-2019
[14] https://github.com/MillionConcepts/pdr

timeline is uncertain and keeps slipping. The difficulty of this task is partly due to the incredible volume of the PDS holdings, but is made much worse by issues with data archived under the relatively less strict standard of PDS3. Some of these issues are known and documented, but previously-unknown ones often present unpleasant surprises when working with legacy archives.

The majority of data currently archived under PDS3 (including "DAT" table and "IMG" image files) use a bitwise structured format sometimes referred to as the “PDS3 format.” It is somewhat like the FITS file format in basic structure, although more loosely defined. As with the FITS format, PDS3 image and table files must include a "header" array of structured ASCII information that defines the format and content of the file. Unlike FITS, the header can either be "attached" (i.e. incorporated into the file itself) or "detached" (residing elsewhere, sometimes but not always within the same directory). Also unlike FITS, the structure and definitions of the metadata in PDS3 were not strictly standardized, or at least the standardization was not strictly enforced. It is this last point that, we find, leads to most of the problems reading data.

The most pervasive issues in PDS3 data are (1) inconsistent mapping between specified data types and byte-size / order, (2) complex files containing multiple or mixed data types, and (3) file format definitions that are not stored in standard locations. We have prototyped solutions to these three issues, and the work plan will involve fully developing and testing those solutions. Other issues arise less frequently (e.g. one-off “bad” files, unique formats) and will be dealt with case-by-case.

As described previously, several Python tools exist for reading subsets of planetary data. These will be reused when appropriate, with input and output syntax / behavior wrapped to provide consistent experience across data types, according to the general design principles of the project. Note that it would not be appropriate to conduct this project in the opposite direction by contributing to one of these existing projects. Of the projects that exist and to which we can contribute code, the only comparable one is *PlanetaryImage*, abandoned by the core developer (A. Godber, *pers. comm.*), and its key components have already been refactored into more capable, prototype versions of *pdr*.

A thorough testing suite will also be developed that, in addition to more targeted approaches, will scrape PDS data holdings in a systematic way to verify expected behavior and identify issues requiring additional development. All software developed as part of this project will be released under a permissive license and made publicly available on the PI’s institutional webspace and the NASA PSD GitHub. It will be made available for easy installation through *pip* and *conda* and also submitted as an affiliate package to the PlanetaryPy project.

Colocation of the investigators is not required for this project. All project team members will work remotely at their respective home institutions or offices within the continental US. The PI will remain in constant contact with the rest of the team through email, chat software, and regular videoconferencing. The PI has worked almost entirely remotely from collaborators for nearly a decade while successfully managing complex software projects of comparable scale. A Gantt chart is provided with information about the distribution of work across time and tasks.

Task 1: PDS data scraper and test suite implementation

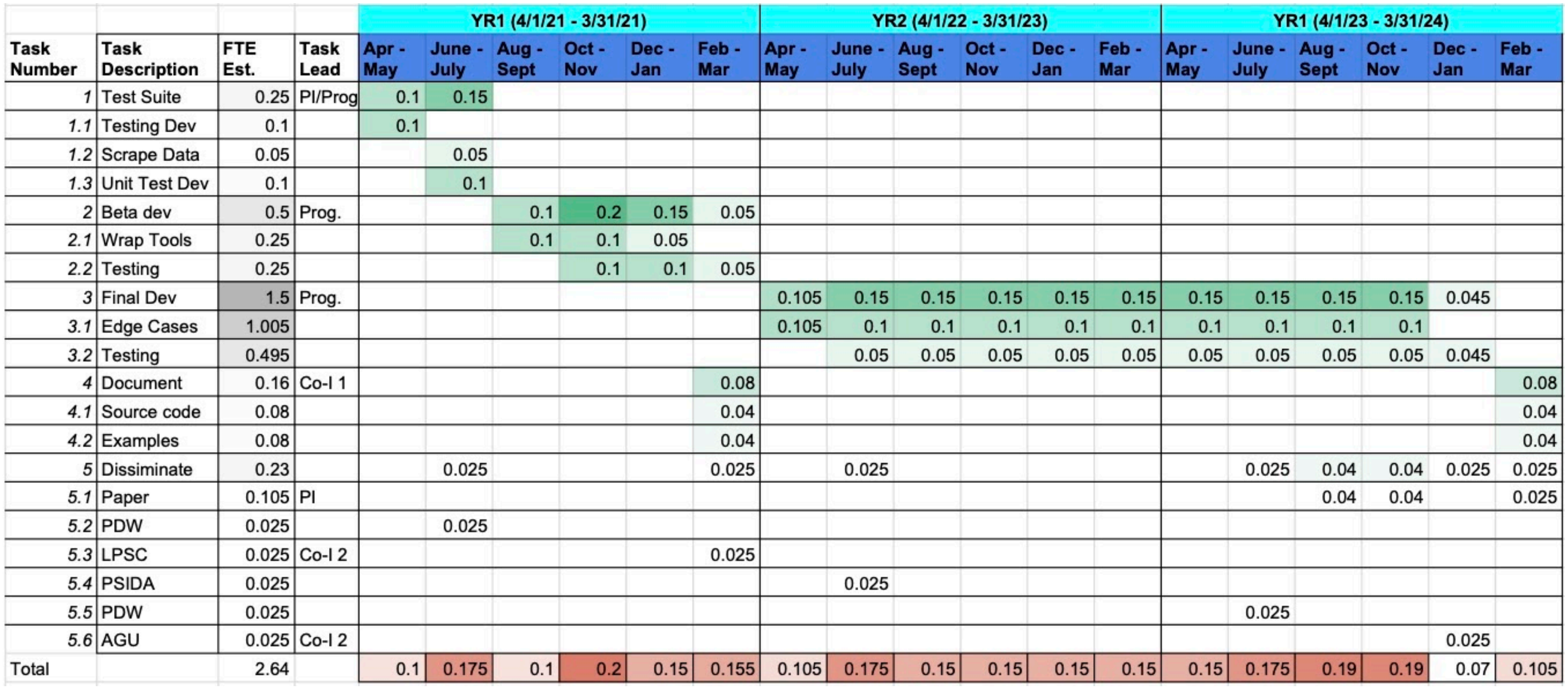
We will systematically crawl or spider the entire PDS data holdings using standard software tools like Scrapy or wget. (Each node of the PDS hosts its data in a separate publicly accessible directory tree, with a slightly different structure between nodes, so we will technically crawl each node individually.) We will record basic information about each file in an extensible database file, including the full web path to the data and the file size. This database of files---which even the PDS does not yet have an equivalent of, as far as we know---will be a remarkably powerful tool for the rest of the project. Critical functions it enables include the ability to estimate the rough distribution and counts of different file types (based on file extensions) across the entire PDS. This will be helpful to understand how to focus our early efforts early for maximum return, and also anticipate where we might have to expend a tremendous amount of effort later to manage strange or rarer file types. (A "PDS Registry" is currently under development by the PDS ENG node to fill this need, but we do not expect it to be completed in time for this project, and even when complete, may not include data archived under PDS3.)

A related script will also download data and run one or more tests on it against development versions of the *pdr*. The compilation of the database file lets us construct systematic sampling tests. For example, as an initial test of the support for the hundreds of thousands MER Pancam images, we can use the information already contained in the directory structure and file names to randomly draw a subsample composed of one of each filetype from each sol-range directory across the whole mission. The output of such tests (including pass / fail as well as error text) will also be stored in the database.

As development proceeds, we will open up testing to larger subsections of the data. The record of "failed" tests will be propagated as issues for software development. Some of those, as appropriate, will also be instantiated as regression or unit tests to be run as part of a continuous integration environment on Github and included with the packaged / shipped version of *pdr*. We will build out a suite of tests that covers the full range of data formats in the PDS, across all missions, instruments, and archives, as well as any particular edge cases. We think that a thorough verification of functionality requires that we test a handful of files of every product type in every archive. Exactly how many files this might be or what volume of data it comprises will not be knowable to us until we complete the reference database. For planning and estimation purposes, however, we must estimate data volume because of the way that cloud compute resources are billed. We believe we will need to directly test functionality on ~10% of data in the PDS, or ~190 Tb.

The spider and test scripts will be deployed via an Amazon Elastic Compute Cloud (EC2) instance, which are virtual machines with resource allocations (e.g. network speed, physical and virtual memory) that can be easily adjusted. This is better than buying dedicated hardware (for approximately the same total cost), because we will enjoy high network speeds and uptimes and can scale resources temporarily to deal with massive files (like HiRISE images) while scaling down to more modest resource use for small files (like Viking data). It also allows all of the members of our team to access and manage this shared resource remotely, and reduces labor and project risk related to system administration and hardware maintenance.

The scripts will be mostly automated, although we expect to routinely modify this system during development. The EC2 instance will run almost continuously until about halfway through



				YR1 (4/1/21 - 3/31/21)						YR2 (4/1/22 - 3/31/23)						YR1 (4/1/23 - 3/31/24)					
Task Number	Task Description	FTE Est.	Task Lead	Apr - May	June - July	Aug - Sept	Oct - Nov	Dec - Jan	Feb - Mar	Apr - May	June - July	Aug - Sept	Oct - Nov	Dec - Jan	Feb - Mar	Apr - May	June - July	Aug - Sept	Oct - Nov	Dec - Jan	Feb - Mar
1	Test Suite	0.25	PI/Prog	0.1	0.15																
1.1	Testing Dev	0.1		0.1																	
1.2	Scrape Data	0.05			0.05																
1.3	Unit Test Dev	0.1			0.1																
2	Beta dev	0.5	Prog.			0.1	0.2	0.15	0.05												
2.1	Wrap Tools	0.25				0.1	0.1	0.05													
2.2	Testing	0.25					0.1	0.1	0.05												
3	Final Dev	1.5	Prog.							0.105	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.045	
3.1	Edge Cases	1.005								0.105	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1		
3.2	Testing	0.495									0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.045	
4	Document	0.16	Co-I 1						0.08												0.08
4.1	Source code	0.08							0.04												0.04
4.2	Examples	0.08							0.04												0.04
5	Dissiminate	0.23			0.025				0.025		0.025						0.025	0.04	0.04	0.025	0.025
5.1	Paper	0.105	PI															0.04	0.04		0.025
5.2	PDW	0.025			0.025																
5.3	LPSC	0.025	Co-I 2						0.025												
5.4	PSIDA	0.025									0.025										
5.5	PDW	0.025															0.025				
5.6	AGU	0.025	Co-I 2																	0.025	
Total		2.64		0.1	0.175	0.1	0.2	0.15	0.155	0.105	0.175	0.15	0.15	0.15	0.15	0.15	0.175	0.19	0.19	0.07	0.105

the final year, and we have budgeted with that assumption. The database file will reside in Amazon S3 storage, which can accommodate arbitrarily large files at marginal cost.

We have allocated 0.25 FTE to the design, development, and testing of these tools in the first four months of the project. The effort will be largely split between the PI and the Programmer. The actual *operation* and ongoing modification / use / maintenance of these tools is not included in this estimate, but incorporated into the development Tasks 2 and 3. The cost drivers for AWS are the resource requirements and total runtime of the EC2 instance and the total data volume that will need to be downloaded from the PDS. (AWS bills for data transfer to or from their servers but not within.) We have budgeted for 2.5 computer-years of a midrange EC2 instance and 190 Tb of data transfer.

<u>Task 2: Initial tool implementation up to beta release</u>

We will release a high-functioning preliminary version of the tool by the end of the first project year. Based on our prior development and testing in this domain, we estimate that this "beta" release will support the *read()* functionality for about 60% of the data that is currently archived in the PDS. All data released under PDS4 will work because these data are all supported by the *pds4_tools* library, which we will wrap. A majority of image data stored as uncompressed binary arrays will work (e.g. correctly structured PDS3

IMG format, FITS, ISIS Cube, Vicar, ENVI array) because they are readable by *rasterio*, which we will wrap. Unusual compressed data file formats will not yet be supported. File types supported by GDAL, including GeoTIFF and JP2, will be supported. We expect that most metadata stored as PVL will be parsable by the already-developed *pvl* tool; that tool may fail when the metadata is malformed according to the strict rules of the PVL language. Table data stored as structured ASCII will probably all be supported, because it will be parsable with the versatile CSV and FWF tools built in to *Pandas*. Table data stored as structured binary (which usually carry the DAT extension in PDS) may or may not be supported at this stage depending on where and how the metadata that stores the structure definitions for those files is stored.

This task will include strictly defining the input / output idioms of the software, wrapping pre-existing functionality from the prototype software and other Python libraries, and adding support for "low difficulty / high impact" data as found in the first phase of testing (per Task 1). We have allocated 0.5 FTE to this effort over 8 months. The task lead will be the Programmer.

Task 3: Full feature and implementation and edge case handling

Following the beta software release, we will begin iterative testing with further software development towards the goal of the *pdr* supporting ~100% of data archived in the PDS. Planned workflow is as follows: (1) Tests for major subsets of data will be designed, initiated, and prioritized based on the work in Task 1. The tests will determine whether the *current* developmental version of *pdr* can successfully read data, and what the failure modes are when it cannot. (2) The output of tests will be propagated as software tickets (i.e. GitHub issues) which will initiate further software development to address those cases. (3) Those cases will be propagated to in-package regression / unit tests which will verify, through a continuous integration service, that software changes do not cause previously-supported data to fail. (4) The cycle will be repeated until the team agrees that complete support for PDS data has been achieved.

We have allocated 1.5 FTE to this effort, spread out over about 1.5 years. The task leader will be the Programmer with substantial assistance from all other team members.

Task 4: Documentation and release of tools and source code

Although we will document the software source code while in development, we also want to provide high quality documentation for end users of the software. The documentation will cover the use of the software with a variety of file types. It will include Jupyter Notebook-based examples of using the software for common scientific tasks like reading and displaying data, modifying data types, simple calibration or standard mathematical operations, and data conversion. We will assume that our target audience for the documentation includes *experienced computer users who are novice Python users,* such as researchers with longtime experience with Matlab or IDL who might be first-time Python users because *pdr* provides such a seamless mechanism for reading and working with planetary data. The task leader will be Co-I St. Clair. We have allocated 0.16 FTE to this task, split evenly to correspond with the release of beta software at the end of the first year and the final release at the end of the project. Co-I Aye will take responsibility for packaging the software for distribution via *pip* and *conda*.

Task 5: Other dissemination of results

Conference presentations and workshops: One of either the PI, Co-I St. Clair, or *Programmer*---TBD based on availability---will present at the anticipated Planetary Data Workshops (PDW) in Flagstaff AZ in summer 2021 and 2023 and the Planetary Science Informatics and Data Analytics (PSIDA) conference at location TBD in summer 2022. They are the two main conference series on the topics of technical / computational, research support, and data management tasks related to planetary science. A large number of software developers---the writers of pipelines and tools---who rarely attend other planetary science conferences *do* regularly attend PDW and PSIDA. This is therefore an excellent place where we will present upon and otherwise advertise the tool to the user bases most likely to be early adopters. Co-I Aye will attend both the Lunar and Planetary Science Conference (LPSC) in 2022 and the American Geophysical Union (AGU) conference in 2023 to present on the *pdr* and also teach workshops or tutorials on its use. Aye has substantial prior experience teaching the use of Python in planetary science research practice. Each conference is ~1 week in duration, so 0.025 FTE has been allocated to each, which includes time for presentation preparation.

Peer-reviewed publication: About midway through the third year of the project, the PI will lead a peer-reviewed article for *Astronomy & Computing* that describes the project as a whole, including tool design, development approach, and lessons learned. *A&C* is an open-access journal that includes descriptions of software tools such as *pdr* in its scope. We have allocated 0.08 FTE for article preparation and an additional 0.025 FTE for revisions.

Astrophysical Source Code Library (ASCL): We will submit the *pdr* software to the ASCL, a registry of software related to astronomical research, to obtain a trackable DOI separate from the *A&C* publication. The requirement for inclusion of software in the ASCL is that it has been described or used in support of a peer-reviewed publication, which the submission to *A&C* will satisfy. We have previously confirmed with Alice Allen, the editor of ASCL, that "planetary science" is included in their scope. [*priv. comm.*] The work effort required is *de minimis.*

OpenPlanetary Tutorials: The documentation and examples will be made available to the OpenPlanetary community for inclusion in "data cafes" that they host from time to time at international planetary science workshops. A video tutorial for *pdr* will be created and posted to the OpenPlanetary YouTube Channel. The PI and Co-I Aye are founding members of the Boards of Directors of OpenPlanetary, respectively. The effort required is minimal and folded into the "documentation" tasks.

PlanetaryPy Affiliate Package: We will apply to have *pdr* included as an "affiliate package" in the PlanetaryPy project to create a unified framework of Python-based planetary software tools for planetary science research. (PlanetaryPy is inspired by the successful *Astropy* project.) The *pdr* software will be developed from the beginning to adhere to the standards of the PlanetaryPy project. The project PI and Co-I Aye are members of the PlanetaryPy technical committee, and we anticipate that *pdr* will be well within scope and published guidelines[15] and will be accepted. The work effort required for application as an affiliate package is *de minimis*.

Project Scope Estimation

This is an unavoidably large project with many possible unforeseen (and unforeseeable) issues, and therefore unusually difficult to scope as an engineering task. We benefit from substantial prior experience with data archived in the PDS, but this will be the first time that anyone (as far as we know) has undertaken a project that completely supports *all* ~1.8 petabytes of extant PDS data. We know that data archived under PDS3 had many problems, many of which have yet to be discovered. We know it not only from simple inspection but also from a recent survey talk on the history of PDS3/4 presented by Anne Raugh (of the Small Bodies Node) at the first OpenPlanetary Virtual Conference (OPvCon).

The required level of effort for this project is clearly a function of the number of unique "data product types" archived in the PDS, However, that number is not available. Nobody has cataloged the data in that way, as far as we know---and we have asked many people. The number of separate *instruments* from which observational data has been archived, however, is a reasonable proxy. This can actually be calculated with some accuracy because "INSTRUMENT_NAME" or some equivalent is a standard metadata label in every PDS data set we have ever seen. (Of course, there are PDS-archived data sets that are *not* the output of specific instruments, but these were relatively rare until recently, when moves towards scientific reproducibility have encouraged more scientists to archive the output of their R&A projects, rather than only the data output from missions. More recently archived R&A output should be subject to the strict requirements of PDS4 and therefore pose almost no issue as far as accessibility.) Per information provided to us by ENG Node staff following a request for a count of unique instruments archived in PDS, we estimate that the number is ~600. We then also estimate that the "difficulty" of reading the data follows a Pareto distribution. This is a common heuristic in software project management, often called the “80/20 Rule:” 20% of the project will take 80% of the effort. Following this heuristic: we estimate that 80% of data types will be relatively easy to provide read support for (~2 hours), 16% of data types will require a moderate amount of additional effort (~2 days or 16 hours), and the remaining 4% of data types will require a relatively large amount of effort (~2 weeks or 80 hours). The Pareto distribution is a good heuristic in many situations where precise information is not available, but it also comports with our impression of the data and associated difficulties based on our prior experience working with planetary science data generally and prototyping the Planetary Data Reader specifically. The result of this analysis is that ~2 FTE is required.

The expectation of NASA for this solicitation is that projects will be *successfully conducted with allocated resources*. We are fully qualified, given sufficient time, to address any data reading issues we might encounter. Our primary concern is having too little allocated FTE to handle unforeseen difficult cases. Our Pareto breakdown of 2/16/80 hours incorporates these eventualities without "padding." We have requested the level of FTE support that we think is necessary and sufficient to guarantee project success. However, if the allocated FTE for this

[15] https://github.com/planetarypy/TC/blob/master/Procedures/

project proves inadequate, it will be triaged out of the 4% of most troublesome data. This is to say that the overwhelming majority of all data archived in the PDS will be supported by the final release of *pdr* even in what weconsider the worst-case scenario for project risk.

<u>Team</u>

Chase Million (PI) is the founder and CEO of Million Concepts, a research and consulting company with particular expertise in astronomical and remote sensing imaging data. He is an experienced software developer and software project manager with almost 20 years of experience in research and research support. He was previously the lead developer for the direct imaging calibration pipeline of the GALEX mission, and he is the designer and developer of the gPhoton project which made 130Tb of GALEX imaging data available on-demand from the Mikulski Archive for Space Telescopes. He has extensive prior experience with PDS data and standards. He developed software for the ground calibration and analysis of MER Pancam data which is still in use for Mastcam and (soon) Mastcam-Z. He is a founding member of the Board of Directors of OpenPlanetary, an international non-profit for the promotion of open source software and workflows in planetary science research. As the project PI, he will be responsible for all aspects of the project. He will be responsible for ensuring that research goals are met with scientific integrity and within time and budget constraints, and for the proper allocation of funds. He will manage team members and tasks as described in the proposal.

Michael St. Clair (Co-I) is a general technologist. He recently led an effort to correct various errors in the Apollo 15 and 17 Heat Flow Experiment data archives and migrate these archives to a single PDS4 bundle (urn:nasa:pds:a15_17_hfe_concatenated, held by GEO), and is currently funded (with Million) for other efforts to process and deliver Arecibo, Chang'E, and VLA data to the PDS. He is also an experienced analyst, technical writer, and Python programmer. He holds undergraduate degrees in theater and mathematics from The Case Western Reserve University and a Ph.D. from Stanford in Theater and Performance Studies, and was formerly an instructor on design at Stanford's d.school and UC Berkeley. He will participate in all aspects of this project, as outlined elsewhere in the proposal, and take particular responsibility for the software documentation tasks.

Michael Aye (Co-I) has 18 years of experience developing in Python and several years of experience creating shared packages on public services like PyPI and Github. He has previously produced Python-based tools for the Cassini Imaging Science Subsystem [Aye, 2016], the MAVEN IUVS instrument, MRO HiRISE, and LRO Diviner, and UVIS. He is a founding member of the Board of Directors of OpenPlanetary. He frequently teaches and lectures on the use of Python in planetary research. He will remain abreast of all aspects of the project, and contribute to programming and design of the overall tool. He will take primary responsibility for the dissemination of results, by leading training sessions, at AGU and LPSC.

Programmer: A half-time professional programmer or software developer will assist throughout the project, performing a large fraction of the programming tasks. The nature of the work does not require that the programmer be very familiar with planetary research or data at the outset, which will allow us to draw on a larger pool of talent outside of academia. This person will be an employee / staff of Million Concepts. We will start a search for this person as soon as the project selection is made, with the expectation that they will be available on the first day of the project.

Data Management Plan

This project team is committed to open and reproducible research. It is our intention that all of our research data products and results can be fully reproduced from publicly available source code and documentation with minimal effort. Any data or source code underlying figures, tables, or other results in publications or archived work products will be provided or included as supplementary material or documentation.

The primary work product of this project will be a Python-based software tool that reads planetary observation data as contained in the PDS archives. We anticipate that the software will have a total uncompressed volume of <2 Mb. Secondary work products will include documentation and tests in a variety of standard machine- and human-readable formats, with a total volume of ~10Mb. The software will be developed to a high standard of quality, well-documented for both end users and later developers, and designed with goals of future-proofing in mind (e.g. limited and mature dependencies). It will include a thorough testing suite. The documentation will include information on how to use the software, with examples corresponding to common use cases, as well as information about the operating environment in which the code was developed and tested, including OS and other dependencies. We will package and make the tool available for easy installation through the two main Python package repositories: *pip* and *conda*. Members of this project team will attend several conferences during the period of performance to describe this tool and instruct community members on its use.

Software and source code developed as part of this project will be publicly released under a permissive open-source license, and we will strongly favor the use of open-source tools and software dependencies. Source code, testing scripts, and documentation will be hosted on the PI's institutional webpage or version control repository (e.g. Github) and updated on an ongoing basis throughout the period of performance. Source code, testing scripts, and documentation will also be "archived" at NASA's PSD Github site before the end of the period of performance. The PI will also retain virtual machine images with examples of running software, with all dependencies, in non-public institutional repositories. The software will be submitted as a *core package* to the PlanetaryPy project, which designation will make it a priority for ongoing community maintenance. The software will be registered with the *Astrophysics Source Code Library*, and thereby assigned a unique permanent identifier. Its design and functionality will be described in a peer-reviewed journal article.