Labelocity Users Guide
Multimission PDS4 Labels

R. Deen
K. Crombie
N. Toole
M. McAuley
S. Mayer
J. Engelke
Jet Propulsion Laboratory, California Institute of Technology

June 26, 2023

Version 1.0

Introduction

Creating PDS4 labels can be a daunting task – significant detail is needed to create a rich label with usable metadata. When hundreds of thousands of labels are required for a large project, such as Mars 2020, the task is magnified. A different challenge occurs in a smaller task like a PDART where labels are required within the context of a limited budget.

What is needed is a flexible, customizable, automated multi-mission system for label production. In *Labelocity,* we believe we have created just such a system.

Labelocity consists of a large set of reusable Velocity macros [velocity], driver scripts, and a wrapper around the PDS tool `mi-label` [mi_label] that creates labels from a variety of data sources. It can read metadata from ODL (PDS3) products and VICAR images, and supports extra metadata insertion via JSON. Other types of products (e.g. FITS) can be indirectly supported via an ODL-conversion step. The JSON file provides a more flexible input option as its metadata could come from virtually anywhere (for example extracting from a database).

The system so far has been used to create most of the image, browse, mesh, calibration, and documentation labels for the InSight [insight_bundle] and Mars 2020 [m20_bundle] missions; PDS3->4 transitions for MSL and Phoenix (in progress); three PDARTs with data from MSL [Deen2023], Voyager [Wenkert2022], and NEAR-MSI [Golish2023]; and non-image data from Mars 2020 and MSL.

This document provides an overview of *Labelocity*, describes how to use it, and why you might want to.

Obtaining Labelocity

The *Labelocity* package is fully open source. It can be obtained on its own here:

https://github.com/NASA-AMMOS/labelocity

Visit that site for installation instructions. It is an evolving system that is being actively extended to support new missions and types of data products while retaining compatibility with existing ones.

Labelocity is also included as a component of the VICAR/VISOR open source delivery here:

https://github.com/NASA-AMMOS/VICAR

which might be more convenient if working with VICAR.

Using Labelocity

Labelocity can be used in one of three primary ways.

First, you can reuse one of the “main” macros (named `*.vm`) and wrapper scripts, with mission-specific details captured in per-mission macro files. This gives you the primary structure of a label for free, and you just fill in or modify the details.

Second, you can write your own custom “main” macro making use of the extensive macro library (named `*.vmac`).

Third, you can simply use *Labelocity* as a source of inspiration, or examples of how to write complex Velocity macros. We hope if you go this route you will eventually realize the benefits of the first two methods.

Within *Labelocity*, there are a set of “main” macros that guide the construction of the label. The most common “mains” are: `mipl_cameras.vm` (which supports most imaging products) and `mipl_non_imaging.vm` (which supports a variety of table and binary products, for example M20 Supercam LIBS products [libs_pds]). The name “mipl” here denotes that the *Labelocity* package was developed by the Multimission Image Processing Laboratory at JPL; it does not mean the package can only be used with MIPL images. Other “main” macros exist which support other types of products including documents, ancillary and browse products.

Wrapper Scripts

There are a number of wrapper scripts included as part of *Labelocity*. These are convenience wrappers that call the core `jConvertIIO` program (which itself wraps `mi-label`) with appropriate parameters and “main” macros. In some cases, such as the mesh wrapper, the script scrapes the data products to determine needed metadata and put it in JSON format. In others, such as the documentation label script, it provides the mission name for use by the mission factory.

Labelocity relies on `jConvertIIO`, which requires some form of an input label (embedded or detached). For cases where no such label product exists, the wrapper scripts can refer to a dummy image to pass into the tool. Along with this dummy image, extra information can be supplied via a separate JSON file. An appropriate dummy image is included as part of the package and is used by the scripts.

Note that the scripts expect the environment variable $V2JBIN to point to the script directory, and $V2TEMPLATES to point at the Velocity template directory.

One other script in the $V2JBIN directory, `diff_and_version_dir.csh`, is described in a later section.

LabelImage

This is the primary wrapper script for images and is in some ways the simplest. It works with VICAR and ODL images directly, extracting label information from them automatically.

LabelNonImage

This is the primary wrapper for non-image data products (delimited-tables, binary). Originally written to support Mars2020, the non-imaging templates and wrappers are being refactored to become multimission (including a non-imaging factory template), with an expanded set of macro support for various table structures.

LabelBrowse

Creates a label for the browse image. Surprisingly, it doesn't actually need the browse image itself to create the label, just the filename, from which most of the metadata is determined (it does use a dummy image to invoke `jConvertIIO`). It supports key-value pairs (on the command line in the form `-prop key value` which go into a JSON file for use by the templates. This is handy when the image filename does not conform to known patterns in the templates. For example, the browse image for the PLACES base map is created by:

```
$V2JBIN/LabelBrowse $output_dir/orbital_map.png -prop
STANDARD_MISSION_FILENAME False -prop PDS4_BUNDLE msl_places -prop
PDS4_COLLECTION data_maps -prop VERSION_NUMBER 6 -prop MISSION_NAME
m2020 -prop BROWSE_COLLECTION browse_maps -prop BROWSE_VERSION 6
```

Doing it this way requires no knowledge of filename conventions or data specifics on the part of the template and is usable by any mission.

label_doc_file.csh

This is a simple script that is surprisingly useful to build labels for documentation files. All necessary information is provided on the command line. That gets built into a JSON file and passed to *Labelocity*. Note that the doc file itself is not needed, just the filename.

label_ancillary_file.csh

Similar to `label_doc_file.csh` but creates labels for ancillary files used by the VICAR/Mars suite, such as list files and tiepoint files.

LabelObj

This script is used to label terrain meshes in OBJ format [obj]. It is invoked using the `.obj` file but assumes a material file (`.mtl`) and image skin (`.vic/VIC/img/IMG`) exist with the same base name in the same directory. It uses the `build_mesh_extras.py` script to extract metadata from the `.mtl` file of the form

```
#lbl NLF_0645_0724205951M777RAS_N0310856NCAM02645_0A0195J03.lbl
#obj NLF_0645_0724205951M777RAS_N0310856NCAM02645_0A0195J03.obj
#xyz NLF_0645_0724205951_777XYM_N0310856NCAM02645_0A0195J02.IMG
#mtl NLF_0645_0724205951M777RAS_N0310856NCAM02645_0A0195J03.mtl

#INSTRUMENT_ID NAVCAM_LEFT
#START_TIME 2022-12-13T12:28:13.367
#STOP_TIME 2022-12-13T12:28:14.574
#ROVER_MOTION_COUNTER_NAME ( site , drive , pose , arm , sha , drill , rsm , hga ,
bitcar , seal )
#ROVER_MOTION_COUNTER ( 31 , 856 , 6 , 0 , 0 , 0 , 18 , 0 , 0 , 0 )

#REFERENCE_COORD_SYSTEM_NAME SITE_FRAME
#REFERENCE_COORD_SYSTEM_INDEX_NAME ( site )
#REFERENCE_COORD_SYSTEM_INDEX ( 31 )
```

and convert it to a JSON file for input into *Labelocity*. Note that this part could be replaced by any method to obtain the metadata and convert it to JSON. It can also be extended to allow more types of metadata values.

The mesh "main" template (`mesh_obj_label.vm`) also looks for a file with the same name but a .lbl extension. This should be an XML label fragment that describes the tables that make up the mesh. The VICAR program `marsmesh` generates this. It looks like the following:

```
<Table_Delimited>
    <name>OBJ Material Reference Table</name>
    <offset unit="byte">0</offset>
    <parsing_standard_id>PDS DSV 1</parsing_standard_id>
    <description>Material file for the OBJ file</description>
    <records>1</records>
    <record_delimiter>Carriage-Return Line-Feed</record_delimiter>
    <field_delimiter>Horizontal Tab</field_delimiter>
    <Record_Delimited>
        <fields>1</fields>
        <groups>0</groups>
        <Field_Delimited>
            <name>mtllib</name>
            <field_number>1</field_number>
            <data_type>ASCII_String</data_type>
            <description>Description of .mtl file.</description>
        </Field_Delimited>
    </Record_Delimited>
</Table_Delimited>
<Table_Delimited>
    <name>OBJ Material Name Table</name>
     ...
</Table_Delimited>
<Table_Delimited>
    <name>OBJ Vertices Table</name>
     ...
    </Table_Delimited>
 ...
```

make_meshlab.csh

This is a script that makes a meshlab project file given a list of .obj files. It makes both the project file itself and the label for it. It may be of limited utility but is a good/simple example of using JSON to feed metadata to a template.

label_cal_file.csh

Similar to `label_doc_file.csh`, but for use with calibration files. Supports both image and text files. It contains a mission-specific section to determine the location of the SIS, which is pointed to by the cal label.

create_cal_labels.csh

May not be useful on its own but is a great example of scripting label creation. This script goes through the calibration files used by the VICAR/Mars software and builds labels for every cal file. It's a good example of multimission design as well, in that it looks for data that any of the missions use and simply does nothing if that data is not available. So if we're labeling InSight data, it picks up a whole host of InSight-specific calibration files, but ignores them for other missions.

diff_and_version_dir.csh

Script to check one bundle against a prior one and bump the version number of anything that has change. See the section on versioning, below.

The jConvertIIO Program

The `jConvertIIO` program is the core of *Labelocity*. It is a general-purpose image conversion tool that can convert almost any image format to another (see [vicar_guide, section 2.7.1]). In certain circumstances (notably VICAR <-> PDS3/ODL) it will preserve metadata as well. It has been extended to generate PDS4 labels by wrapping the PDS `mi-label` tool. The `mi-label` tool does most of the work calling the Velocity engine itself, but `jConvertIIO` provides several additional capabilities, such as JSON input, input label conditioning, supporting multiple levels of label groups, and direct support for VICAR and PDS3/ODL images.

How Labelocity Works

The concept of multimission PDS4 labels is discussed in [Deen2019]. Mechanically, *Labelocity* works by having one mission-specific macro file per mission that implements a standard "API". This is in turn used by a mission factory template that determines which mission-specific macro file to load automatically. By using this factory system, many of the details are encapsulated, which makes multimission templates practical.

An enabling concept for multimission templates is that we “collapse out” any sections that are not used. For example there is a section in the reusable templates specifically supporting the MSSS mini-header, but if no input labels exist for the mini-header, that section does not appear in the output label. The same thing applies to mission-specific data dictionaries. This allows you to add new sections with impunity, knowing they will not affect other missions unless the enabling metadata is provided as input.

Supporting New Missions or Products

If you are a data provider for an already-supported mission, *Labelocity* may work for you right out of the box. However, most people will need to modify *Labelocity* to support new missions, or new types of data products for an existing mission.

You have a choice on how to modify the library. We obviously recommend implementing compatible extensions, so your changes can be committed back for others to use. This is more work, as you have to sometimes create additional mission-specific macros in each of the supported missions’ templates and think about how changes might affect existing missions. But the reward of doing so is great, as your work will help others (including, likely, yourself) in the future. But you can also just fork the macros and edit them as you need, which might be appropriate for a simple/quick data set.

In the case of adding compatible extensions, new missions can be added by copying an existing mission implementation and providing your own values and logic to fields and macros starting with “msn_” prefix. Then just update the mission factory to determine which mission the data is for, so it can invoke the appropriate mission implementation.

Mission implementations declare their own set of Local Data Dictionaries and respective versions to use when generating the label prologue, including which Information Model (PDS4 version) is in use.

Macro Library Functionality

As mentioned, the macro library contains a set of main macro files (*.vm) and supporting macro files (*.vmac). The supporting macro files partition re-usable macros and field definitions based on categories: e.g. core field definitions in `vcore.vmac`, image support in `image.vmac`, mission surface in `surface.vmac`, common macros in `general.vmac`, utilities in `utils.vmac`. A set of multimission non-imaging support is available in `non-imaging-utils.vmac`.

While the supporting macro files exist to support the main macros, they can also be utilized as pieces for custom macro files. For example, if your product’s ODL label contains a set of articulation-state groups (*_ARTICULATION_STATE), one could include the *generateArticulationDeviceParams()* macro (from `surface.vmac`) to generate all of the

associated PDS4 `<geom:Articulation_Device_Parameters>` labels, including content based on the nested group labels.

More generally, there is a large set of commonly usable macros to support concepts such as nil-checking (either prevent PDS4 label from being generated, or add a PDS4 nil-entry), handling situations where a child label is either a single node or a list of nodes, string manipulation and tests, value-bounds checking, etc.

Format file conversion for tables

Most non-image data sets are in the form of tables, either delimited-text or binary. In the PDS3 world, when a particular table definition is re-used amongst a set of products, the table definition can be described individually in a FMT file, which contains a segment of an ODL (aka PDS3) label. These FMT files are then referenced in product labels with a structure pointer (`^STRUCTURE`) entry. These FMT files have all the information necessary to create PDS4 label descriptions of the table.

The `jConvertIIO` application does not support resolving these structure pointers, so a pre-processing step is required to substitute those pointers with the contents of the FMT files. *Labelocity* provides such a tool, `label_expand_struct.py`, that accepts a label and a directory of FMT files, and then repeatedly expands the structure pointers until no further expansion can be performed. This provides `jConvertIIO` with a self-contained representation of the whole label.

FMT files are obviously useful in a PDS3 -> PDS4 conversion scenario, but we have found them easier to write than PDS4 label descriptions, so we sometimes use this mechanism even when creating pure PDS4 labels.

Automation of Label Creation

It is not enough to have a system like *Labelocity* that will create labels for individual products. You must be able to create labels for the entire set of data products in your bundle. This can be accomplished in a number of ways. To some extent this is outside the scope of *Labelocity*, but we will provide some use cases as examples.

The simplest method is a script, such as a shell script or Python script. An example of this is the create_cal_labels.csh script mentioned above. A much more extensive but similar in concept script is used to create the PLACES delivery [places_bundle]. See below for another example script.

Label creation can be done as part of the pipeline that makes the products themselves. Just call *Labelocity* as each product is created. While in principle this should work fine, in practice it doesn't work that well, as you typically have to iterate many times on label creation until

everything validates, and you generally don't want to waste the time re-creating the actual products each time.

JPL has developed a system called APPS [Radulescu2015] which, in various incarnations, will take huge bundles of data and run the labels for all of them in a distributed fashion. This is used for InSight and Mars 2020, as well as the MSAM PDART. A system such as this can be a hybrid; for example on Mars 2020 the calibration and document collections are built with scripts but the data products themselves use APPS.

Example driver script

The following example script creates the labels for the Mars 2020 camera documents. Having such a script makes it much easier to update things, vs. hand-editing of labels every time.

```
#!/bin/csh
#
# Label the doc files for M2020 cameras
#
# Usage:
#   m20_label_docs.csh bundlename
#

if ($# != 1) then
   head -7 $0
   exit
Endif

set bundle = $1

set in = `pwd`/indoc/
set out = `pwd`/../bundles/$bundle/document_camera/

mkdir -p $out
cd $out

#### SIS ####
set file = Mars2020_Camera_SIS.pdf
cp $in/$file $out/
$V2JBIN/label_doc_file.csh M2020 $file $bundle document_camera "This document is a software interface
specification describing the format of the Mars 2020 Camera Data Products" "N. Ruoff, R. Deen, O.
Pariser" "PDF/A" "PDF version" 2 "Mars 2020 Project Software Interface Specification (SIS): Camera Data
Products"

#### BUNDLE SIS ####
set file = Mars2020_Camera_Bundle_SIS.pdf
cp $in/$file $out/
$V2JBIN/label_doc_file.csh M2020 $file $bundle document_camera "This document is a software interface
specification describing the format of the PDS Archive Bundle for the Mars 2020 Cameras" "R. Deen, K.
Crombie" "PDF/A" "PDF version" 2 "Mars 2020 Project Software Interface Specification (SIS): PDS Camera
Archive Bundle Structure"

#### README ####
### set file = release_notes.txt
### cp $in/$file $out/
### $V2JBIN/label_doc_file.csh M2020 $file $bundle document_camera "Mars 2020 Camera Release Notes" "R.
Deen, M. McAuley", "7-Bit ASCII Text" "Mars 2020 Camera Release Notes, ASCII Text Edition" 1

#### LABEL TABLE ####
set file = Mars2020_Camera_SIS_Labels_sort_pds.pdf
cp $in/$file $out/
```

```
$V2JBIN/label_doc_file.csh M2020 $file $bundle document_camera "This document is part of Appendix B to
the Mars 2020 Camera SIS: a table containing PDS4 label attributes and classes, their XPaths, VICAR
equivalents, and definitions" "N. Ruoff, R. Deen, O. Pariser, N. Arena" "PDF/A" "PDF version sorted by
PDS" 2 "Mars 2020 Cameras PDS4 Attribute Definitions"

set file = Mars2020_Camera_SIS_Labels_sort_pds.html
cp $in/$file $out/
$V2JBIN/label_doc_file.csh M2020 $file $bundle document_camera "This document is part of Appendix B to
the Mars 2020 Camera SIS: a table containing PDS4 label attributes and classes, their XPaths, VICAR
equivalents, and definitions" "N. Ruoff, R. Deen, O. Pariser, N. Arena" "HTML" "HTML version sorted by
PDS" 2 "Mars 2020 Cameras PDS4 Attribute Definitions"

set file = Mars2020_Camera_SIS_Labels_sort_vicar.pdf
cp $in/$file $out/
$V2JBIN/label_doc_file.csh M2020 $file $bundle document_camera "This document is part of Appendix B to
the Mars 2020 Camera SIS: a table containing PDS4 label attributes and classes, their XPaths, VICAR
equivalents, and definitions" "N. Ruoff, R. Deen, O. Pariser, N. Arena" "PDF/A" "PDF version sorted by
VICAR" 2 "Mars 2020 Cameras PDS4 Attribute Definitions"

set file = Mars2020_Camera_SIS_Labels_sort_vicar.html
cp $in/$file $out/
$V2JBIN/label_doc_file.csh M2020 $file $bundle document_camera "This document is part of Appendix B to
the Mars 2020 Camera SIS: a table containing PDS4 label attributes and classes, their XPaths, VICAR
equivalents, and definitions" "N. Ruoff, R. Deen, O. Pariser, N. Arena" "HTML" "HTML version sorted by
VICAR" 2 "Mars 2020 Cameras PDS4 Attribute Definitions"

#### MANUAL LABEL TABLE ####
set file = Mars2020_Camera_SIS_Operations_Label.pdf
cp $in/$file $out/
$V2JBIN/label_doc_file.csh M2020 $file $bundle document_camera "This document is part of Appendix B to
the Mars 2020 Camera SIS: a table containing the operations keywords and their telemetry data sources"
"N. Ruoff, R. Deen, O. Pariser" "PDF/A" "PDF version" 1 "Mars 2020 Cameras Operations Keyword
Definitions"

#### PIPELINE DIAGRAM ####
set file = Mars2020_Camera_SIS_Pipeline_Flow.pdf
cp $in/$file $out/
$V2JBIN/label_doc_file.csh M2020 $file $bundle document_camera "This document is part of Appendix D to
the Mars 2020 Camera SIS: a diagram containing the complete pipeline data flow" "G. Hollins" "PDF/A"
"PDF version" 1 "Mars 2020 Cameras Pipeline Data Flow"

#### WATSON/ACI TABLE ####
set file = Mars2020_SHERLOC_ACI_WATSON_Image_Information_Supplement.pdf
cp $in/$file $out/
$V2JBIN/label_doc_file.csh M2020 $file $bundle document_camera "This document provides additional
information regarding SHERLOC ACI and WATSON images.  It is described in Appendix E of the Mars 2020
Camera SIS" "K. Edgett" "PDF/A" "PDF version" 3 "SHERLOC ACI/WATSON Image Information Supplement"

#### SPECIAL PROCESSING FLAG TABLE ####
set file = Mars2020_Special_Processing_Flags.pdf
cp $in/$file $out/
$V2JBIN/label_doc_file.csh M2020 $file $bundle document_camera "This document is part of Appendix G to
the Mars 2020 Camera SIS: a list of Special Processing Flags used in the data set" "H. Abarca, S.
Abercombie, F. Ayoub, R. Deen, S. Fernandez, S. Oij, O. Pariser" "PDF/A" "PDF version" 2 "Special
Processing Flag Table"

#### SCLK USAGE TABLE ####
set file = Mars2020_SCLK_usage.pdf
cp $in/$file $out/
$V2JBIN/label_doc_file.csh M2020 $file $bundle document_camera "This document is part of Appendix F to
the Mars 2020 Camera SIS: a list of SCLKs and how they are used" "R. Deen, O. Pariser, H. Lee, A.
Sanboli" "PDF/A" "PDF version" 2 "SCLK Usage Table for Mars 2020"

#### CAMERA CHARACTERISTICS TABLES ####
set file = Mars2020_Camera_Basics.pdf
cp $in/$file $out/
```

```
$V2JBIN/label_doc_file.csh M2020 $file $bundle document_camera "This document is part of Appendix H to
the Mars 2020 Camera SIS: a summary of the basic characteristics of each camera" "J. Van Beek, R. Deen,
J. Maki" "PDF/A" "PDF version" 1 "Camera Basics Table for Mars 2020"

set file = Mars2020_Camera_Characteristics.pdf
cp $in/$file $out/
$V2JBIN/label_doc_file.csh M2020 $file $bundle document_camera "This document is part of Appendix H to
the Mars 2020 Camera SIS: a detailed set of characteristics for each camera" "J. Van Beek, R. Deen, J.
Maki" "PDF/A" "PDF version" 1 "Camera Characteristics Table for Mars 2020"
```

Versioning of Data Products and Documents

Versioning is a necessary evil. Data products and especially documents get updated over time. When they change, a new version must be created, as PDS does not support deleting or overwriting already-delivered products.

Tracking versions as they change can be tedious. For that reason, we include a utility script called `diff_and_version_dir.csh` in the $V2JBIN scripts directory, which will take an old and new directory, find files that are common, diff them, and if they've changed, update the label of the new product to old+1. This way you can save the bundle you sent to PDS last time, generate a new bundle, and let the script look for changes and bump version numbers for you. A simple modification to the script could eliminate files that have not been updated, but our policy on Mars 2020 deliveries (of the document and cal collection where this is used) has been to redeliver everything; the receiving PDS Node then just ignores duplicate products. Consult with your PDS node on their delivery preference.

When Manual Label Creation Makes Sense

There are instances where manual label creation makes sense. Generally, there are only a few bundle and collection labels needed to complete an archive. These labels have specific formats that are not widely variable. In these cases, using an existing template (or example from another PDS bundle) and inserting your bundle or collection specific values into the standard attributes will yield results quickly. It is not worth trying to automate production of these labels, as hand-editing is faster.

Another instance where manual creation makes sense is for one-off products, such as individual products for a PDART where you are only delivering it once and there's only one of the products. Such a label may not be similar enough to anything else to make automation worthwhile.

However, things like document labels, while simple, have enough commonality that scripting their production is worthwhile (especially since we already provide such a script).

Misconceptions about Multimission Labels

The goal of multimission label templates [Deen2019] is to represent concepts in a common way. Missions may represent the same concepts in a number of ways. However, translating those

concepts to a common representation using multimission label templates assists end users in searching for and identifying useful metadata. There are arguments that certain concepts may be too mission-specific to map appropriately into multimission labels. In certain cases, it may be appropriate to keep the mission specificity, but in many cases it will be more useful to end-users to use the multimission methodology. Even if the definition does not fit exactly (which can be handled by mission-specific documentation, e.g. part 2 of [Deen2019]), there are similar properties that can be exploited. For example, the translation of spacecraft clock to actual time may be mission specific, but all spacecraft clocks share the property of being monotonically increasing in time, so data can be sorted in time order using them.

In the case of PDS3 -> PDS4 migrations, the multimission templates may save data providers time and effort by illuminating common mappings from PDS3 ODL keywords to PDS4 XML attributes. The PDS3 -> PDS4 mappings should hold up from mission to mission. If there are questions on the mappings, care should be taken to understand why the mappings do not fit.

An additional benefit of using multimission templates is that the common concepts can facilitate data mining and machine learning activities. The common concepts and common metadata will allow algorithms to easily interpret metadata from one project to the next (see for example [Bond2023]).

Multimission label templates incorporate and make use of the PDS4 Information Model that is implemented through the Core Model and a set of Local Data Dictionaries (LDDs) [pds_im]. The LDDs are used to specify discipline and mission specific metadata concepts. Multimisson label templates need to be responsive to dictionary rules, enumerations, definitions and development. From a multimission perspective, overly prescribed LDDs can make label development difficult for secondary data providers, who might want to add or modify metadata to represent the value they added to the primary data set. PDS has addressed this concern by allowing modification requests for metadata definitions, enumerations, nil-ability, etc. The LDD update process and negotiation may take significant time, but it should be the goal of LDD stewards and secondary data providers to find a middle ground between specificity and permissiveness. In summary, LDDs should be designed with flexibility and concept reuse in mind.

Experiences in Developing the Package

This project started with a set of mission-specific main templates, which were first reviewed for commonality and then reorganized into a set of reusable macro files, mission API macros, and a mission-factory. As missions were added, new mission-based extension points in the main templates became necessary, and as a consequence the mission API continues to evolve and grow.

Once a critical set of mission implementations were reached, the general structure of the macros became more stable. Guiding principles were established and redundancies were deconstructed into smaller supporting macros.

At this point, supporting a new mission is generally approached by an initial focus on the specifics of the <Mission_Area>, followed by an examination of the more general sections of the label or new additions or special-handling.

We strive to maintain compatibility across missions, which places extra burden on ensuring special cases are appropriately supported without adding unnecessary complexity to general cases. Non-trivial template updates should also include regression testing across mission data sets searching for unintended deltas. While we generally recommend adding your own extensions to the framework with the same level of compatibility, we acknowledge that for some users, a fork of the library would be easier, especially for one-off projects such as PDARTs. Missions supporting multiple deliveries really should use the framework so they can be upgraded over time.

As a general caveat, the limitation of the Velocity template language where macros can only write or return text is a challenge that requires some seemingly-cryptic design decisions for supporting functions that return values. However, after some basic usage of the library, many of the conventions become more apparent.

Conclusion/Summary

If you are starting a mission (or PDART etc) from scratch, you should take a look at *Labelocity*. Defining labels based on multimission commonality helps users better understand your labels because many of them have an experience base to build from. And you have a starting place in the often bewildering world of PDS4 labels.

If you have an existing mission for which you need to generate PDS4 labels, *Labelocity* may still be helpful. Its use with Voyager data - very different from the Mars missions, which share a lot in common - shows that it can be done. At the very least, *Labelocity* provides a very rich set of examples from which you can build your own templates if necessary. However, we look forward to incorporating your updates into the package for the benefit of future users.

Please feel free to contact the authors with questions regarding *Labelocity*.

References

Bond2023	S.A. Bond, S. Lu, J. McAuley, “Machine Learning Dictionary for the ML Community”, 6th Planetary Data Workshop, Flagstaff, AZ, June 2023. Abstract #7029.
Deen2019	R.G. Deen, C.M. De Cesare, J.H. Padams, S.S. Algermissen, N.T. Toole, S.R. Levoe, J.S. Hughes, P.M. Ramirez, “Multimission Labels in PDS4”, set of posters and talks at 4th Planetary Data Workshop, Flagstaff, AZ, June 2019. Abstracts #7050, #7051, and #7052. https://www.hou.usra.edu/meetings/planetdata2019/pdf/7050.pdf https://www.hou.usra.edu/meetings/planetdata2019/pdf/7051.pdf https://www.hou.usra.edu/meetings/planetdata2019/pdf/7052.pdf
Deen2023	R.G. Deen, A. Tinio, N. Ruoff, H.E. Abarca, N. Toole, M. McAuley, M.K. Crombie, “Mastcam Stereo Analysis and Mosaics (MSAM) Release”, 6th Planetary Data Workshop, Flagstaff, AZ, June 2023. Abstract #7070.
Golish2023	D.R. Golish, D.N. DellaGiustina, K.J. Becker, C.A. Bennett, M. Robinson, M.K. Crombie, “Blur remediation in NEAR MSI images”, Icarus, 2023, 115536, ISSN 0019-1035, https://doi.org/10.1016/j.icarus.2023.115536.
insight_bundle	https://pds-imaging.jpl.nasa.gov/portal/insight_mission.html
labelocity	https://github.com/NASA-AMMOS/labelocity
libs_pds	https://pds-geosciences.wustl.edu/missions/mars2020/supercam.htm
mi_label	https://nasa-pds.github.io/mi-label/
m20_bundle	https://pds-imaging.jpl.nasa.gov/volumes/mars2020.html
obj	https://en.wikipedia.org/wiki/Wavefront_.obj_file
pds_im	https://pds.nasa.gov/datastandards/documents/im/
places_bundle	https://pds-geosciences.wustl.edu/missions/mars2020/places.htm
Radulescu2015	C. Radulescu, S.R. Levoe, S.S. Algermissen, E.D. Rye, S.H. Hardman, J.S. Hughes, M.D. Cayanan, E.M Sayfi, “AMMOS-PDS Pipeline Service (APPS) - A Data Archive Pipeline for Mission Operations”, 2nd Planetary Data Workshop, Flagstaff, AZ, June 2015. Abstract #7050 https://www.hou.usra.edu/meetings/planetdata2015/pdf/7050.pdf
velocity	https://velocity.apache.org
vicar_guide	https://github.com/NASA-AMMOS/VICAR/blob/master/vos/docsource/vicar/VICAR_guide_4.0.pdf
vicar_open	https://github.com/NASA-AMMOS/VICAR/
Wenkert2022	D. Wenkert, R.G. Deen, W.L. Bunch, “Uranus and Neptune Albedos May Need Revision”, AGU Fall Meeting, Chicago, IL, December 2022. Bibcode: 2022AGUFM.P32E1865W

	https://ui.adsabs.harvard.edu/abs/2022AGUFM.P32E1865W/abstract

Acknowledgements

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004).