Multimission Labels in PDS4, Part 3: Templates R. G. Deen[1], C. M. De Cesare[1], J. H. Padams[1], S. S. Algermissen[1], N. T. Toole[1], S. R. Levoe[1], J. S. Hughes[1], P. M. Ramirez[1], [1]Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Dr, Pasadena, CA 91109, Bob.Deen@jpl.nasa.gov

Introduction: This is the third of a set of three abstracts [1,2] describing the authors' experience with creating multimission dictionaries, labels, and tools for PDS4. See part 1 [1] for a full introduction.

This abstract discusses the use of Velocity templates and the PDS Generate tool in ways that support multimission reuse.

Multimission Velocity Template ("Generate" tool): While proper label design is important, it doesn't matter if we cannot actually create a PDS4 label. It is certainly possible to generate XML straight out of the telemetry processor or other data creation tool. However, this is untenable for large labels. Directly writing out XML leads to very brittle code that is hard to maintain. An intermediary tool is needed that can construct the proper XML from the available metadata – which could be a data structure in telemetry, a database, a JSON file, a keyword=value list, or any of a number of formats.

One answer to this need is the Velocity engine. This neatly solves the problem of creating XML from disparate input data sources, and it turns out that with some care, Velocity templates can be written in a multimission, reusable manner.

The PDS Engineering Node has created a tool called "generate" [3], which wraps around the Velocity engine and customizes it for use with PDS4 labels.

Velocity Basics: Velocity is a Java-based template engine, created by the Apache project [4]. In a nutshell, a Velocity template looks a lot like XML, with dynamic references embedded. These dynamic references can pull information from other data sources. For example:

```
<img:Downsampling_Parameters>
   <img:downsampling_venue>
      $!label.INSTRUMENT_STATE_PARMS.
              PIXEL_DOWNSPAMPLE_OPTION
   </img:downsampling_venue>
   <img:downsampling_method>
      $!label.INSTRUMENT_STATE_PARMS.
              DOWNSAMPLE_METHOD
   </img:downsampling_method>
   <img:Pixel_Averaging_Dimensions>
      <img:height_pixels unit="pixel">
         $!label.INSTRUMENT_STATE_PARMS.
                 PIXEL_AVERAGING_HEIGHT
      </img:height_pixels>
      <img:width_pixels unit="pixel">
         $!label.INSTRUMENT_STATE_PARMS.
                 PIXEL_AVERAGING_WIDTH
      </img:width_pixels>
   </img:Pixel_Averaging_Dimensions>
</img:Downsampling_Parameters>
```

The "$!label" pulls information from a data structure. The "generate" tool reads the VICAR or PDS3 labels and puts them in the "label" data structure based on their property/group and keyword name. Thus, the first instance finds the value of the `PIXEL_DOWNSAMPLE_OPTION` keyword in the `INSTRUMENT_STATE_PARMS` group in the label.

Currently, the "generate" tool will read VICAR and PDS3 labels as data sources, as well as JSON files. Other data sources can be easily added as needed.

The above results in XML that, for InSight, looks like:

```
<img:Downsampling_Parameters>
   <img:Pixel_Averaging_Dimensions>
      <img:height_pixels unit="pixel">
         1
      </img:height_pixels>
      <img:width_pixels unit="pixel">
         1
      </img:width_pixels>
   </img:Pixel_Averaging_Dimensions>
</img:Downsampling_Parameters>
```

A very important feature of "generate" is that if there is no data within a "paragraph" of XML, that "paragraph" will be suppressed in the output. Thus in the example above, `DOWNSAMPLE_METHOD` and `PIXEL_DOWNSAMPLE_OPTION` do not exist in InSight labels. Therefore, the `<downsampling_venue>` and `<downsampling_method>` elements are suppressed. If this same template were applied to MSL data, those elements would be present, because the corresponding keywords exist in MSL labels. If the `PIXEL_AVERAGING_HEIGHT` and `PIXEL_AVERAGING_WIDTH` were also missing, the entire `<Downsampling_Parameters>` element would be suppressed, and so on. It turns out this feature is key for making multimission templates, as described in the next section.

Velocity is a complete language, including conditionals, loops, subroutines ("macros"), and variables. It is possible to do quite complex operations in Velocity. For example, the following snippet will loop through a parallel array of temperature names/values in separate labels, and generate an XML list of them. It also calls a macro ("`#getUnitAttr`", not shown here) to take the unit in a separate keyword and convert it from PDS3 to PDS4 format, including the "`unit=`" XML attribute, only if the unit keyword

exists. Lines starting with "#" are Velocity commands.

```
#set($length=$label.INSTRUMENT_STATE_PARMS.
        INSTRUMENT_TEMPERATURE_NAME.size()
#foreach ( $i in [1..$length] )
   <img:Instrument_Device_Temperature_Index>
      <img:device_name>
         $label.INSTRUMENT_STATE_PARMS.
               INSTRUMENT_TEMPERATURE_NAME.
               get($foreach.index)
      </img:device_name>
      <img:temperature_value
         #getUnitAttr(
               $!label.INSTRUMENT_STATE_PARMS
               'INSTRUMENT_TEMPERATURE'
               $foreach.index)>
         $label.INSTRUMENT_STATE_PARMS.
               INSTRUMENT_TEMPERATURE.
               get($foreach.index)
      </img:temperature_value>
   </img:Instrument_Device_Temperature_Index>
#end
```

Multimission Velocity Templates: Because of the effort made to create and use multimission dictionaries, the PDS4 labels are very similar for all the cameras on the recent landed Mars missions (MER, MSL, InSight, Mars 2020, even Phoenix). The VICAR/PDS3 labels are also very similar, due to the multimission VICAR software that creates these products. This means they are the perfect candidate for a multimission Velocity template.

The multimission Velocity template first determines which mission the data belongs to, by looking at certain VICAR/PDS3 labels. It then uses the "#parse" command (like an include file) to include the mission-specific Velocity module for that mission.

The mission-specific modules implement a defined set of macros – basically, a multimission API – that define things that vary per mission, such as the mission name, URI prefix, which dictionary versions are used, how to parse the filename for information, etc. Currently there are 20 constants and 14 macros that have to be defined for each mission (this number will likely increase somewhat over time).

These macros and constants allow the rest of the Velocity template to work in a multimission manner, customizing items as needed. As of this writing, the multimission template is about 3500 lines, with only about 300 in each mission-specific module.

Once the macros for the first mission (InSight) were written, it took only about 3-4 hours to create the mission-specific macro file for MSAM, a PDART task using MSL data [5]. The resulting label was correct and valid (although not complete, as additional items needed to be added to the multimission template). Still, that is an impressive time savings over a traditional, non-multimission approach.

One of the most important factors in enabling the multimission template is the feature of "generate" (mentioned above) where "paragraphs" of XML are deleted if they have no contents. This means the template can include code used for one mission but not others, without hurting the missions that don't use it. For example, the entire MSSS "mini-header" section (described in part 1 of this series [1]) is simply coded into the multimission template. For missions or instruments that don't have a mini-header, the section is ignored. Similarly, the template has paragraphs specifying video parameters, Z-stack parameters, and filters, which are simply ignored for still, non-focusing, or monochrome cameras.

This allows the template to grow over time to accommodate different features of new missions, without compromising support for the old missions. We fully expect all of the MSL and Mars 2020 labels to be added in to the multimission template, with a regression test on InSight data showing no differences.

The value of this approach is demonstrated if you contemplate migrating other missions to PDS4, such as MSL or MER, that are currently archived with PDS3. Using a traditional approach, it could take months of work to update mission-specific Velocity macros for these missions. Using this approach, it is expected to take a day or two.

Conclusion: Contrary to what some may believe, multimission PDS4 labels are well within reach. With a little attention up front, it is entirely possible to write dictionaries, generate labels, and document them in ways that are highly reusable across multiple missions.

In addition to saving development time, these techniques facilitate data mining, as the same concepts are used and re-used on different missions. Parsers need be written only once, metadata schemas can be reused, and people familiar with one mission will instantly recognize data from another.

It is the authors' belief that these approaches unleash the full power of PDS4 – power that until now has been hidden in complexity. Commonality and reuse across missions is what will make PDS4 successful, now and into the future.

Acknowledgements: This research is being performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with NASA.

References:

[1] Deen, R.G. et al (2019), 4th PDW, Abstract #7050
[2] Deen, R.G. et al (2019), 4th PDW, Abstract #7051
[3] https://pds.nasa.gov/tools/about/generate/
[4] https://velocity.apache.org
[5] Deen, R.G. et al (2018), 49th LPSC, Abstract #2332