Passthrough: Template-Driven PDS4 Product Generation. A. Ladegaard[1], [1]Department of Physics, Aberystwyth University, Aberystwyth, SY23 3BZ, UK, arl13@aber.ac.uk

Introduction: The Passthrough software library seeks to provide PDS4 data processors with an integrated solution for generating output labels based on declarative product type templates. It serves as a complementary counterpart to the PDS4 Tools [1] read-in library, enabling processors to interact natively with the PDS4 format without the need for intermediary internal product representations or separate file formats.

Background: When the data processing infrastructure used by a mission does not natively support or produce PDS4 compliant products, a distinction is formed between "operations" and "archive" products [2]. To generate the latter, product labels must be transformed from their operations format into PDS4 before delivery to the archive. Dedicated tools exist to tackle this task, such as MILabel [3] (formerly "Generate Tool", as described by [4]), which employs multimission templates written in the popular Velocity [5] language.

On the other hand, when a mission is not tied to existing processing infrastructure or operations formats, there are few compelling reasons for treating archive products as secondary. Since PDS4 labels are implemented in XML, its Document Object Model (DOM) can be used as a platform- and language-neutral interface to the metadata of products read in by a processor or other software tool. This is demonstrated by the PDS4 Tools library, which only lightly wraps a label's underlying tree structure to provide convenient access also to its associated payload data. Thus, it is straightforward for processors to interact with existing PDS4 products on the input side, and doing so does not require processors to carry a separate internal representation (i.e. intermediary data structure).

The task of generating a PDS4 product on the output side, however, requires specific knowledge of the structure of both its label and payload data. Ideally, the source of this knowledge should not be contained within a particular product processor. Rather, a mission or instrument's product types (e.g. raw, calibrated, mosaic, housekeeping etc.) should be externally defined via template files, to enable these to serve as the formal interfaces between processors that might likely be developed by separate teams. As the basis for such a "type template", a skeleton label could be used, for instance as assembled with PLAID [6]. But at a minimum the template should additionally carry information on permissible structural variations. A key principle of type templates that differentiates them from the kinds used by tools such as MILabel, is that a member of the former should describe the key properties of a product type, invariant of whichever processor that happens to generate it.

Using type templates, a processor wishing to generate products of a particular type would require a means by which to instantiate a blank copy - a "partial label" - based on the type's template. Passthrough is intended to provide this functionality, together with a language specification for type templates, and an extension API to facilitate code reuse.

Template Handler: The template handler component of Passthrough takes the form of a Python library intended to be integrated early in a processor's product generation flow. The "Template" handler class pre-processes a template file into a partial label, which is subsequently made available for the processor to populate key attributes via the DOM. Finally, the handler performs any post-processing tasks before exporting the output product to the file system.

This three-step sequence means that the partial label can be made available to the processor from the get-go, and template logic can be associated with either the pre- or post-processing stage. Conventionally, metadata inheritance and remapping from input products takes place during pre-processing, while pruning of unused substructures, population of regulated attributes, and consistency checks are performed as part of post-processing.

Consistency checking constitutes an important feature of Passthrough, whereby a partial label is validated against the allowed structural variations afforded by the product's template. This provides benefits in particular for human-in-the-loop applications (where a user can be prompted to make corrective changes), as well as during processor development and conformance testing.

Passthrough Template Language: The Passthrough template language (PTL) was developed to accommodate the requirements of type templates. It encourages a declarative approach to logic expression, and seeks to aid human readability by letting templates closely reflect the structure of their end product label. Therefore, substructure imputing macros commonly used in e.g. the Velocity language are disallowed.

The language uses XML-attribute annotations coupled with XPath expressions, which enables template logic to leverage the XML DOM directly. Functionality such as metadata inheritance and remapping, conditional logic, and statically specified attribute population, is coupled syntactically to XML-elements (i.e. PDS4 classes and attributes). For example:

```
<img:Exposure pt:fetch="true()" pt:sources="input">
    <img:exposure_duration_count/>
    <img:exposure_duration unit="s" pt:fetch="false()"/>
    <img:exposure_type
     pt:required="//psa:identifier = 'HRC'"/>
</img:Exposure>
```

XML-attributes under the `pt` namespace are referred to as Passthrough "properties". Above, the `fetch` property is declared on the `img:Exposure` class, through the XPath expression "true()". This indicates that descendants of the class should have their values fetched and imputed (inherited) by Passthrough from their counterparts present in another product label, unless overridden. The `sources` property also declared on the class specifies that whichever product the processor has assigned to the moniker "input" should be the source queried for these values. While this rule will be applied to the `img:exposure_duration_count` child attribute, `fetch` is explicitly deactivated on `img:exposure_duration`. The absence of further properties within its context implies that the product processor is expected to populate this attribute after the pre-processing stage. Finally, `required` is declared on the `img:exposure_type` child, contingent on the conditional "//psa:identifier = 'HRC'". This expression compares the value of a specific attribute (in this case, a sub-instrument ID) of the "input" product to the string literal "HRC", to determine whether Passthrough should expect the presence of `img:exposure_type`'s counterpart in the source tree.

Beyond those showcased in this example, PTL defines several other properties. In short, these are:

- `multi`, used to specify the expected cardinality of fetched elements, or to expand repeating elements that have been collapsed in the template for clarity;
- `fill`, which allows attributes to be populated by XPath expressions; and
- `defer`, which delays the evaluation of an XML-element's other properties until the post-processing stage.

It should also be mentioned that `sources` more generally sets the evaluation context(s) of other property expressions.

The fact that the PTL syntax is exclusively made up of XML-attributes works well with PDS4's very sparse use of these, and arguably results in highly readable templates. This also makes future implementations of the template handler in other languages than Python straightforward.

Extension functions: Employing XPath as the expression evaluation engine has saved a significant amount of development time, but it also presents templates with powerful functionality in the form of extension functions. XPath defines a range of built-in functions that can be used as part of expressions, but it also enables PTL, missions, instruments or processors to define their own extensions via Passthrough's API. For instance, the following excerpt uses the `pt:datetime_add` extension function to populate `pds:stop_date_time` during post-processing, based on the partial label's inherited `pds:start_date_time` and processor-populated `img:exposure_duration`:

```
<Time_Coordinates pt:fetch="true()" pt:sources="input">
    <start_date_time/>
    <stop_date_time pt:fetch="false()"
     pt:sources="template" pt:defer="true()"
     pt:fill="pt:datetime_add(//pds:start_date_time,
     //img:exposure_duration)"/>
</Time_Coordinates>
```

Note that a useful feature of extension functions is their access to the DOM representation of their arguments. The `pt:datetime_add` function is in this way able to inspect `img:exposure_duration`'s unit (e.g. seconds or milliseconds) and act accordingly.

Project Status and Future Work: Passthrough is a relatively new project under active development, and is slated for use by several of the processors that will comprise the ground pipeline for the ESA/Roscosmos ExoMars 2022 mission's PanCam instrument [7]. The software should at present be considered to be in the late alpha stage; while major changes to the language specification are not foreseen, the template handler API and list of common extension functions are not yet stable. Additionally, more work is required to increase the support for `File_Area_*` components (instantiating blank payload structures from the template, generating checksums during export, etc.). Finally, an automated test suite with continuous integration must be implemented before Passthrough can be recommended for mission critical applications.

Conclusions: Passthrough represents a departure from conventional template solutions, and is intended to explore new possibilities in this space afforded by the increasing adoption of the PDS4 standard. Feedback and contributions from the community are welcome and appreciated. The project is open source under the MIT licence and can be found together with more comprehensive documentation on GitHub [8].

Acknowledgements: This work was funded by the UK Space Agency, grant numbers ST/T000058/1 and ST/V002686/1.

References:

[1] Nagdimunov, L. (2017), 3rd PDW, Abstract #7065, https://github.com/Small-Bodies-Node/pds4_tools
[2] Abarca, H. et al. (2019), 4th PDW, Abstract #7112
[3] https://github.com/NASA-PDS/mi-label
[4] Deen, R. et al (2019), 4th PDW, Abstract #7052
[5] https://velocity.apache.org
[6] Algermissen, S. (2017), 3rd PDW, Abstract #7030
[7] Coates, A. J. et al. (2017), Astrobiology, 17, 6-7
[8] https://github.com/ExoMars-PanCam/passthrough