Anaconda Recipe Parser - Introduction

Schuyler Martin <smartin@anaconda.com>

January 17th, 2024

Background

Personal Introduction

Schuyler Martin

- Based in Denver, CO, USA
  - GitHub: https://github.com/schuylermartin45
- Been at Anaconda for ~10 months
  - Have been working on/with our package builders from Day 1
  - Founder and current lead of the Packaging Automation and Tooling (PAT) Team
  - Previously worked on IoT & embedded systems for 6+ years
  - BS/MS Computer Science from the Rochester Institute of Technology (RIT)

History of the Project

Package Building

- Anaconda offers and builds Python packages for paid and free-tiered customers
- These packages use "recipe" files to describe the build process
  - Our publicly available recipe files are found in the AnacondaRecipes Organization on GitHub

All of the following projects are written in Python and now target 3.11

anaconda-linter

- Started as a hack day project by former Anaconda employee, Jerimiah Willhite in July 2022
- This project provides automated checks to validate recipes found in AnacondaRecipes
  - The intent is to ensure quality and consistency with all of our recipe files
  - The tool provides feedback on common mistakes and warns package builders of potential issues
  - Until recently, all errors/warnings emitted by the linter had to be manually fixed by our package builders
- Marco Esters eventually took ownership of the linter as a side project
- In September 2023, the PAT Team took on this project full-time

History of the Project

percy

- Started as a hack day project by another Anaconda employee, Charles Bousseau
  - Initially built to be a library to help "render" recipes for the Python 3.11 build-out
  - Needed some ability to examine dependencies outside of conda-build
- *Rendering*: to produce the list of recipe files effectively used by conda-build on a per-platform basis
  - These "outputs" are called recipe variants
  - In general, there is 1 variant per system architecture/platform
- As a side project, Charles developed a proof-of-concept set of tools to edit recipe files
  - This approach focused on examining the rendered output and made changes on the pre-rendered (original) recipe file
  - Created a `--fix` flag to automatically fix some of the errors/warnings provided by the linter
- In October 2023, the PAT Team took on this project full-time
  - To date, ~10/62 rules have some ability to auto-fix themselves

History of the Project

percy, recipe parser module

- Recipe parser work started in September 2023 by Schuyler Martin
- We started to analyze and categorize the automation opportunities in anaconda-linter
    - We found that there were many opportunities, BUT the current auto-fixing tools in percy were not adequate
    - We were not aware of any existing conda recipe parsers
- So we started building a recipe parser!
    - The anaconda-linter project would act as the existing practical application for the library
    - There was the assumption this library would be useful in future automation efforts

History of the Project

percy, recipe parser module

- The primary goal is to make it as easy as possible to modify recipe files programmatically
    - The library contains a number of convenience functions to reduce common and repeated work
        - Access/mutate some JINJA variables
        - Access/mutate recipe selector comments
        - Access/mutate values in in the file
            - Access can return with or without JINJA variable substitutions
            - Types returned are of the evaluated type (i.e. 1 vs "1")
            - There is some support for JINJA "pipe functions", like "|lower"
        - Find dependencies in multi-output recipes
        - Find fields that match a regex
        - Provide a pre/post change diff of the recipe
- The library also contains the ability to patch fields directly using JSON patch syntax
    - We are nearly RFC-6902 compliant (if you ignore using JSON to edit YAML)
- The module uses test-driven development with an emphasis on using automated code formatting and checking tools

How the Recipe Parser works

Node-based Parse Tree

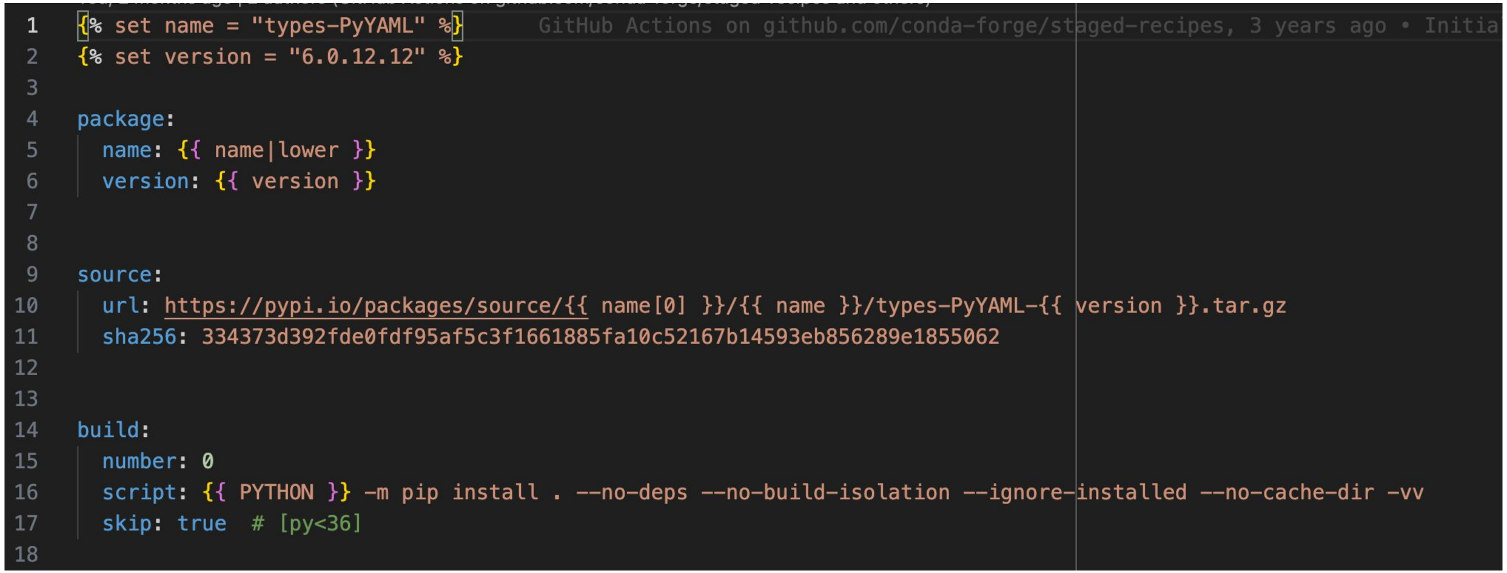
- We use a tree data-structure to capture the *semantics* of the file
  - Trees show the relationship between keys and fields in a file
  - Trees are relatively easy to manipulate
  - To get the results, simply interpret the tree and build-back a YAML file
- Each key or field in the YAML file is represented by a node object
- Each node contains a list of child nodes, forming a tree
  - There is an invisible "root" node that represents the top-level of the documented
- Nodes contain various flags and auxiliary data to describe the intricacies of our YAML-based recipes
  - Comments
  - Whether or not the node is a list member, key, a multi-line string, etc
- Tree paths are described as file-path-like strings by calling-code
  - i.e. `/build/number` or `/requirements/host/1`

Node-based Parse Tree

- We track supplemental data structures for other aspects of the recipe format
  - Look-up table for JINJA variables and values
    - JINJA is a templating language we use in our current recipe format
  - Look-up table for recipe selectors
    - Selectors are comments that conditionally remove lines of our recipe files based on a boolean Python expression
    -
      ```
      build:
          number: 0 # [osx]
          number: 1 # [not osx]
      ```

Node-based Parse Tree (Visual Example)

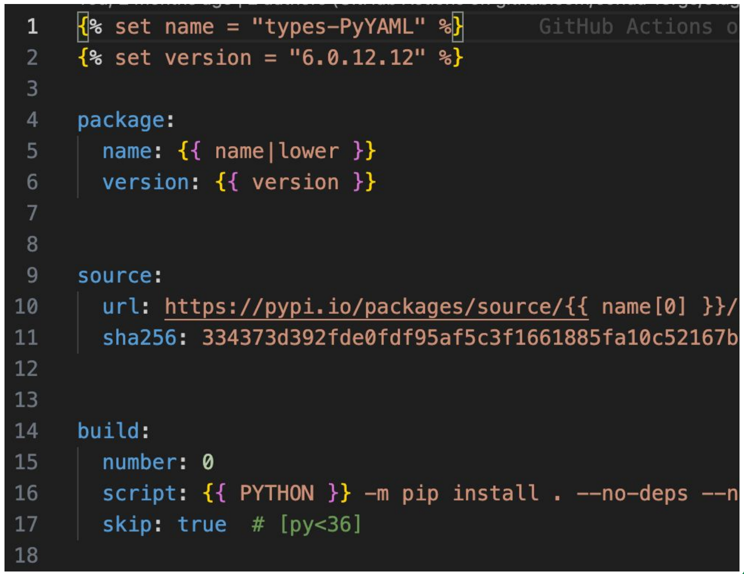
types-pyyaml-feedstock/recipe/meta.yaml



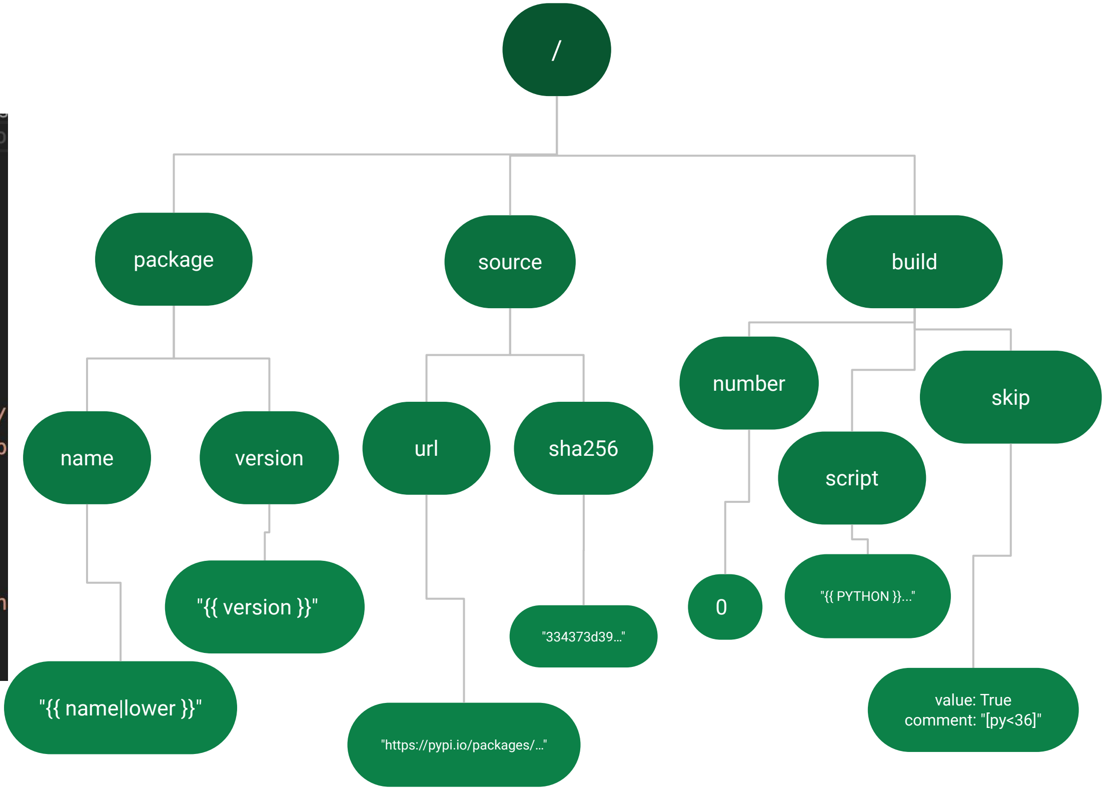
```
{% set name = "types-PyYAML" %}      GitHub Actions on github.com/conda-forge/staged-recipes, 3 years ago • Initia
{% set version = "6.0.12.12" %}

package:
  name: {{ name|lower }}
  version: {{ version }}


source:
  url: https://pypi.io/packages/source/{{ name[0] }}/{{ name }}/types-PyYAML-{{ version }}.tar.gz
  sha256: 334373d392fde0fdf95af5c3f1661885fa10c52167b14593eb856289e1855062


build:
  number: 0
  script: {{ PYTHON }} -m pip install . --no-deps --no-build-isolation --ignore-installed --no-cache-dir -vv
  skip: true  # [py<36]

```

Node-based Parse Tree (Abbreviated Example)

types-pyyaml-feedstock/recipe/meta.yaml

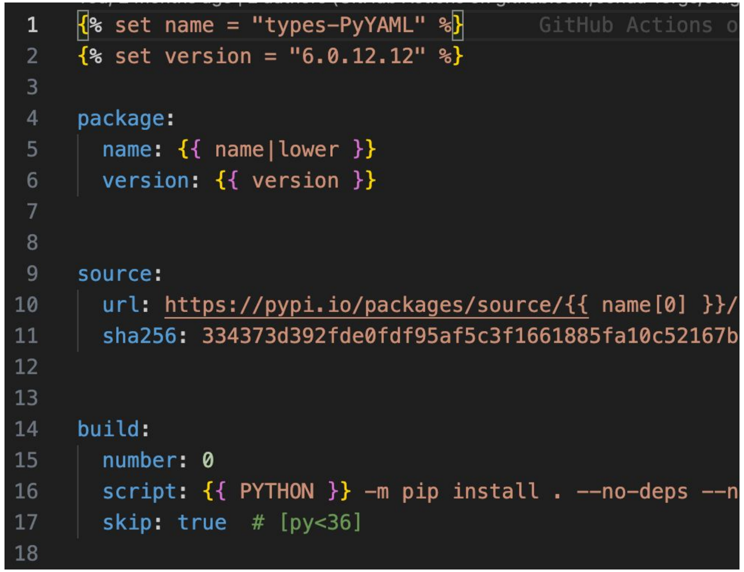


```
{% set name = "types-PyYAML" %}        GitHub Actions o
{% set version = "6.0.12.12" %}

package:
  name: {{ name|lower }}
  version: {{ version }}


source:
  url: https://pypi.io/packages/source/{{ name[0] }}/
  sha256: 334373d392fde0fdf95af5c3f1661885fa10c52167b


build:
  number: 0
  script: {{ PYTHON }} -m pip install . --no-deps --n
  skip: true  # [py<36]

```



Node-based Parse Tree (Abbreviated Example)

types-pyyaml-feedstock/recipe/meta.yaml



```
{% set name = "types-PyYAML" %}        GitHub Actions o
{% set version = "6.0.12.12" %}

package:
  name: {{ name|lower }}
  version: {{ version }}


source:
  url: https://pypi.io/packages/source/{{ name[0] }}/
  sha256: 334373d392fde0fdf95af5c3f1661885fa10c52167b


build:
  number: 0
  script: {{ PYTHON }} -m pip install . --no-deps --n
  skip: true  # [py<36]

```

JINJA Variable Table

name -> "types-PyYAML"
version -> "6.0.12.12"

Note: These do not have to be strings. We will resolve the type of the variable.

Selector Table

```
"py<36" -> [
        SelectorInfo(Node("skip"), "/build/skip"),
        SelectorInfo(Node(True), "/build/skip")
]
```

Current Limitations

- The parser is indent-based
  - conda-forge and AnacondaRecipe files tend to be pretty well indented/formatted, so we took advantage/abused that fact

- When last calculated, ~95% of the ~2600 recipes found in AnacondaRecipes can be parsed
  - The other 5% crash the library when parsed
  - We don't currently have any specifics on how accurately the 95% of recipes are parsed
  - We have a utility script in percy that can perform some categorization of the remaining failures

- There is currently no support for more advanced JINJA macros/features
  - Conditional statements and loops are not supported
  - Self-referencing JINJA variables and list/collection-type variables are not supported
  - Few recipes fall into this category. Likely 5-10% of AnacondaRecipes, if not, far fewer.

Current Limitations

- There is no current support for editing non-selector YAML comments

- PyYAML parses individual lines for us to handle type resolution
  - JINJA expressions, comments, and selectors are parsed by regular expressions

How to use the Recipe Parser

Full development setup and installation instructions
are available in the README

Example 1: Simple Operations

```
# Removing an entry in the file
parser.patch({"op": "remove", "path": "/about/license_url"})

# Accessing a variable and defaulting when one is not found.
parser.get_value("/outputs/1/build/number", 42)

# Return the list of locations where a JINJA variable is found
parser.get_variable_references("name")

# Return True if there is a selector on this path
parser.contains_selector_at_path("/requirements/host/3")
```

Example 2: Removing `git` as a dependency on Windows

```
paths: Final[list[str]] = parser.find_value("git")
for path in paths:
    # Attempt to filter-out false-positives
    if "/requirements" not in path:
        continue
    parser.add_selector(path, "[not win]", SelectorConflictMode.AND)
```

Future Plans/Q & A

Direction & Questions for the Community

- If any of these projects sound interesting to you, please get in contact with Schuyler Martin <smartin@anaconda.com>, we could certainly use some more hands.
  - Some issues are tracked in GitHub, most are tracked in our internal JIRA boards

- Should we move the parser to a separate repo?
  - Increasingly looks like the parser should be broken out of percy and maintained separately

- How can we help with the conda recipe format change?
  - The PAT Team's priorities may happen to align with auto-upgrade tools
  - Since we have a tree-representation of the recipe, transferring to a new format should be "easy"
    - Traverse the tree and write to a new format

- Is this work redundant when compared to existing projects? Should we combine efforts?
  - rattler-build's parser
  - souschef

Notes from the Q & A

- The expectation is this parser work will be unnecessary within a year
  - If/when the new format is universally adopted, this project will no longer be necessary
  - Some manual work will likely need to be done with the most complicated recipes
  - Don't bother moving the parser out of percy for this reason
- It is likely that multiple existing tools will be needed to transfer between recipe formats