Cookiecutter V2 Context Specification Documentation

Release 0.0.1

E.R. Uber

Nov 15, 2017

Contents:

CHAPTER 1

Introduction

This document describes a new Cookiecutter version 2 template format based on two sources:

1. A proof-of-concept submitted via Cookiecutter Pull Request #848 by hackebrot. Specific format information from this pull request is contained herein as notes by hackebrot.
2. Additional format features and functionality described herein are based on implementation and usage experience associated with a specific reference implementation of Cookiecutter by eruber that supports this proposed template format.

This document is not meant to serve as a formal template format specification; it exists simply to describe the template format proposed and its evolution based on solving practical issues encountered when using the format to implement a large complex project template.

All errors (via omission or commission), oversights, and/or misguided implementation decisions reflected herein, rest solely on the shoulders of the primary author, eruber.

CHAPTER 2

Format Proposal

The Cookiecutter version 1 template is a simple JSON file defining a dictionary of key/value pairs that identify variables used in a jinja2 context.

The version 2 template adds additional template **metadata** that exists outside the jinja2 context, but is expected to support features in the future that will enhance the overall user experience.

2.1 Required Metadata Template Fields

Currently the minimum number of required metadata fields is three:

- *name*
- *cookiecutter_version*
- *variables*

2.1.1 name Field

The **name** field is a string identifying the name of the template.

For example:

```
"name": "the-most-famous-template-of-all",
```

Note: hackebrot: We can use this for dumping the JSON context for –replay. Currently we make a good guess based on the directory name of the cloned git repository, which is not great for local templates or relative paths. This could be something like an ID.

2.1.2 cookiecutter_version Field

The **cookiecutter_version** field is a string identifying version information

The current implementation assumes this field identifies the minimum version of Cookiecutter required to process the template, as in:

```
"cookiecutter_version": "2.0.0",
```

There are other meanings of this field to consider, see the note below.

Note: **hackebrot**: This either indicats the version of cookiecutter that this template requires or the version of the spec itself. Not entirely sure what's better in this case and if we want to separate them. Going forward this will allow us to exit early if the used cookiecutter CLI is not the latest one, but the template depends on a new built-in extension or new fields.

2.1.3 variables Field

The **variables** field is an array of objects implementated as an array of ordered dictionaries (OrderedDict). Each element of the array, being an OrderedDict, is a set of key/value pairs associated with a unique variable that is part of the jinja2 context.

The various required and optional key/value pairs associated with a variable will be identified in the *Variables Array* section later in this document.

Note: **hackebrot**: The elements of this array represent a single variable, similarly to what you currently find in a **cookiecutter.json** file.

2.1.4 The Minium Cookiecutter Template

Based on what has been disclosed so far, an example of a minium legal (though relatively useless) Cookiecutter version 2 template would look like this:

```
{
    "name": "template-name",
    "cookiecutter_version": "2.0.0",
    "variables": []
}
```

2.2 Optional Metadata Template Fields

The following fields are optional:

- *description*
- *version*
- *authors*
- *license*

- *keywords*
- *url*

2.2.1 description Field (Optional)

The **description** field is a string containing a human readable description of the template.

Note: **hackebrot**: This can be used for user facing aspects, like a welcome message when running cookiecutter.

2.2.2 version Field (Optional)

The **version** field is a string containing a version identifier; ideally conforming the the Semantic Versioning specification (semver). This version identifier is used to version control the template.

Note: **hackebrot**: This will help us generate helpful error messages.

2.2.3 authors Field (Optional)

The **authors** is an array of strings that identify the template's maintainers.

Note: **hackebrot**: Again this will help users in case they encounter issues. Currently users tend to raise issues on the cookiecutter project rather than the template. I would like to emphasize that template authors need to make sure that their templates work.

2.2.4 license Field (Optional)

The **license** field is a string identifying the license for the template code.

Note: **hackebrot**: The template itself is not runnable software, but contains source code. So I would argue that it should specify a license. Obviously this is not binding if the repository is missing a LICENSE file or w/e the license in question requires. We don't need this for a Minimal Viable Product.

2.2.5 keywords Field (Optional)

The **keywords** field is an array of strings similar in spirit to PyPI keywords.

Note: **hackebrot**: Providing keywords in a template makes it easier for tools, such as the new Cookiecutter Explorer in Visual Studio or Cibopath, to search for templates. Currently users need to go to the template repo and scan through the README or even the template code to see if a template uses certain frameworks.

2.2.6 url Field (Optional)

The **url** field is a string URL for the template project.

Note: **hackebrot**: We can use this to point users to the project if they encounter an error. This would certainly be optional.

2.2.7 Example Cookiecutter Template

Below is an example Cookiecutter template showing all the required and optional metadata fields; note that the variables array is still empty, but not for long:

```
{
    "name": "python-project-skeleton-template",
    "cookiecutter_version": "2.0.0",

    "description": "Cookiecutter template for a general purpose Python project␣
↪skeleton",
    "authors": ["E.R. Uber"],
    "version": "0.3.7",
    "license": "MIT",
    "keywords": ["cookiecutter","python", "project", "template", "skeleton"],
    "url": "http://python-project-skeleton.readthedocs.io/en/latest/index.html",

    "variables": []
}
```

2.3 Variables Array

The **variables** field is an array of ordered dictionaries (OrderedDict). Each dictionary represents a varible in the jinja2 context.

2.3.1 Required Variable Fields

The following fields are **required** to be defined for each variable:

- *name*
- *default*

name Variable Field

The **name** variable field is a string defining the name of the variable in the jinja2 context.

For example:

```
{
   "name": "project_repo",
   ...
}
```

Note: hackebrot: This is nothing different from what we have in the current **cookiecutter.json** as keys. **These must not be templated!**

default Variable Field

The **default** variable field can be of any legal default value type and is the default value of the variable named in the previous section.

The various legal types supported will be addressed in a later section.

For example, the variable named 'project_repo', may have a default value of "cookiecutter-template-converter" as in:

```
{
   "name": "project_repo",
   "default": "cookiecutter-template-converter",
   ...
}
```

Note: hackebrot: Again this is what we already have as values. If a default is a string, we must assume it is templated, so we render it before prompting the user.

2.3.2 Optional Variable Fields

The following variable fields are optional:

- *type*
- *description*
- *prompt*
- *prompt_user*
- *hide_input*
- *choices*
- *skip_if*
- *do_if*
- *if_yes_skip_to*
- *if_no_skip_to*
- *validation*
- *validation_flags*
- *validation_msg*

type Variable Field (Optional)

The **type** variable field is a string that defines the type of the variable.

The **type** field's default value is: string.

Note: **hackebrot**: This defaults to string, which reflects the current behaviour (right now we cast every value to string, so we can render it). Having a type allows us not only to make use of Click types for prompts, but we can also cast the values after they have been rendered.

The reference implementation supports the following default value types:

- string
- boolean
- yes_no
- int
- json
- **float**
- **uuid**

Note: **eruber**: The proof-of-concept proposal omitted types **float** and **uuid**, but they were added to the Cookiecutter reference implementation since they are both inherently supported by the underlying user prompt functionality provided by Click.

description Variable Field (Optional)

The **description** variable field is a string used to describe what the variable means.

The **description** field's default value is: None.

Note: **hackebrot**: We can show this if the users runs verbose mode, to make it even clearer for what a variable is used for and potentially indicate what the requirements for a field are.

Note: **eruber**: It would appear that in Cookiecutter v1.6.0 (upon which the reference implementation of Cookiecutter v2 is based) does not pass the command line –verbose option to the main cookiecutter API call (its just used to control the logging level). So in the reference implementation, it is hardwired to True. Thus if a description is defined, it will be emitted prior to a user prompt.

The reason the Cookiecutter reference implementation does pass the verbose option into the Cookiecutter API is because the reference implementation has a set of implementation guidelines and one of those guidelines was NOT to change the Cookiecutter API.

prompt Variable Field (Optional)

The **prompt** variable field is a string that will be used to prompt the user for input.

The **prompt** field's default value is rendered by jinja2 as:

```
'Please enter a value for "{variable.name}"'
```

Note: hackebrot: Currently we show variable [default]:, but a template author could provide a more friendly message allowing for a better user experience.

prompt_user Variable Field (Optional)

The **prompt_user** variable field is a boolean that if true will show user prompts; and if false will not prompt the user for input.

The **prompt_user** field's default value is: True

Note: hackebrot: This can be used to hide prompts from a user if the template author wishes to use these fields but retrieve the information from somewhere else, for example the current year. This is currently supported with a hack by prepending a variable name with _.

Note: eruber: The reference implementation also still honors this hack – a variable name prefixed with an underscore does not generate a user prompt – it has the same effect as **"prompt_user" : false** being specified in the template.

hide_input Variable Field (Optional)

The **hide_input** variable field is a boolean - when specified as true will allow user input, but will not echo the user's keystrokes back to the console. This makes it suitable for entering sensitive information like passwords.

The **hide_input** field's default value is: False

Note: eruber: Though not documented in the his pull request write-up, the actual proof-of-concept code for context.py by hackebrot does implement the hide_input field - and thus, so does the Cookiecutter reference implementation.

choices Variable Field (Optional)

The **choices** variable field is an array of string, boolean, or number which lists valid choice values for that variable.

The **choices** field's default value is: []

Note: hackebrot: This is currently supported with lists in cookiecutter.json. However this field would be optional for a variable and is different from type in the sense that a choice will still be processed to have the specified type when stored to the context

skip_if Variable Field (Optional)

The **skip_if** variable field is a string that holds conditionals based on other fields. The conditional logic is rendered by jinja2.

The **skip_if** field's default value is: ''

If the conditional in the **skip_if** string evaluates to True, then this variable is skipped – the user will see no prompt to enter data for this variable.

Note: **hackebrot**: This one is a bit tricky. In it's current form it would be a string containing a jinja2 template. When prompting the user this is rendered and checked for equality against "True". This allows us to skip variables based on previously entered information.

---

do_if Variable Field (Optional)

The **do_if** variable field is a string that holds conditionals based on other fields. The conditional logic is rendered by jinja2.

The **do_if** field's default value is: ''

If the conditional in the **do_if** string evaluates to True, then this variable is NOT skipped – the user will be prompted to enter data for this variable.

Note: **eruber**: This field was added to the reference implementation to offer a balance to the **skip_if** field – sometimes its just more convenient to express the logic in terms of what variable should be processed rather than what variable should be skipped.

---

if_yes_skip_to Variable Field (Optional)

The **if_yes_skip_to** variable field is a string that names a variable to process next if the value of the current variable is True (yes).

This field is used with **yes_no** type variables to allow conditional processing that can skip multiple variables.

The **if_yes_skip_to** field's default value is: None

Note: **eruber**: Added to the Cookiecutter reference implementation. Having only a **skip_if** mechanism became logically complex when trying to skip multiple variables. This field makes skipping over mulitple variables very easy.

---

if_no_skip_to Variable Field (Optional)

The **if_no_skip_to** variable field is a string that names a variable to process next if the value of the current variable is False (no).

This field is used with **yes_no** type variables to allow conditional processing that can skip multiple variables.

The **if_no_skip_to** field's default value is: None

Note: **eruber**: Added to the Cookiecutter reference implementation as a logical balance to the **if_yes_skip_to** field.

---

validation Variable Field (Optional)

The **validation** variable field is a string containing a regular expression used to validate the user input.

The **validation** field's default value is: None

Note: **hackebrot**: This would allow us to have some additional checks for accepting user input. Think of PEP8 compliant names for Python modules. Rather than using a post_gen_project hook and abort generation, we could ask the user to try entering another value.

validation_flags Variable Field (Optional)

The **validation_flags** variable field is a list of strings. Each item in the list names a validation flag that can be specified to control the behaviour of the **validation** field's validation check. Specifying a flag in this list is equivalent to setting the validation flag to True, not specifying a flag is equivalent to setting it to False.

The **validation_flags** field's default value is: []

The default value of this variable has no effect on the validation check.

The validation flags supported are:

- **ascii** - enabling re.ASCII
- **debug** - enabling re.DEBUG
- **ignorecase** - enabling re.IGNORECASE
- **locale** - enabling re.LOCALE
- **mulitline** - enabling re.MULTILINE
- **dotall** - enabling re.DOTALL
- **verbose** - enabling re.VERBOSE

See: https://docs.python.org/3/library/re.html#re.compile

Note: **eruber**: This field was added to the Cookiecutter reference implementation to complete the **validation** field's functionality.

For example, to perform input vaildation that ignores case and enables verbose, do this:

```
"validation": "SOME-REALLY-MIND-ALTERING-REGULAR-EXPRESSION",
"validation_flags": ["ignorecase", "verbose"]
```

validation_msg Variable Field (Optional)

The **validation_msg** variable field is a string that can be used to specify a more user friendly message to be issued when input validation fails.

The **validation_msg** field's default value is: None

Note: **eruber**: This field was added to the Cookiecutter reference implementation when it became apparent that the normal validation failure message that emits the validation regular expression, can at times, use some additional validation input hints – especially if the validation regular expression is complex. See the example below.

For example, to support validation of a semantic version number with all of its features, the following variable might be defined:

```
{
    "name": "project_version",
    "default": "0.0.1",
    "description": "Enter the project's semantic version number (see: semver.org).",
    "prompt": "A semantic version number is of the basic form: MAJOR.MINOR.PATCHLEVEL
↪",
    "validation": "^([0-9]|[1-9]+[0-9]*)\\.([0-9]|[1-9]+[0-9]*)\\.([0-9]|[1-9]+[0-
↪9]*)(-)?(-[0-9A-Za-z-\\.]*)*(\\+)?(\\+[0-9A-Za-z-\\.]*)*$",
    "validation_msg": "Follow the form X.Y.Z where X, Y, and Z are non-negative␣
↪integers, and MUST NOT contain leading zeroes.",
    "type": "string"
}
```

As you can see the validation's regular expression is somewhat daunting, so if a **validation_msg** is specified it will be issued in addition to the default validation failure message that emits the regular expression.

A console session that illustrates would look like:

```
Enter the project's semantic version number (see: semver.org).
A semantic version number is of the basic form: MAJOR.MINOR.PATCHLEVEL [0.0.1]: 0.01.
↪001
Input validation failure against regex: '^([0-9]|[1-9]+[0-9]*)\.([0-9]|[1-9]+[0-9]*)\.
↪([0-9]|[1-9]+[0-9]*)(-)?(-[0-9A-Za-z-\.]*)*(\+)?(\+[0-9A-Za-z-\.]*)*$', try again!
Follow the form X.Y.Z where X, Y, and Z are non-negative integers, and MUST NOT␣
↪contain leading zeroes.
A semantic version number is of the basic form: MAJOR.MINOR.PATCHLEVEL [0.0.1]: 0.1.1
```

CHAPTER 3

Extra Context Overwrite Considerations

This section identifies further functionality in the Cookiecutter reference implemenation because the new template format requires new solutions in the area of context overwriting.

Context overwriting occurs when the [EXTRA_CONTEXT] is specified on the command line as explained in the Cookiecutter docs section Injecting Extra Context.

Note: eruber: Note that because the new template's jinja2 context is an array of OrderedDict elements – one for each variable in the jinja2 context; the EXTRA_CONTEXT specified by the user, must also be an array of OrderedDict elements – one for each variable that the EXTRA_CONTEXT wishes to overwrite.

3.1 Overwrite Considerations Regarding 'default' & 'choices' Fields

When a variable is defined that has both the **default** and the **choices** fields, these two fields influence each other. If one of these fields is updated, but not the other field, then the other field will be automatically updated by the overwrite logic.

If both fields are updated, then the **default** value will be moved to the first location of the **choices** field if it exists elsewhere in the list; if the **default** value is not in the list, it will be added to the first location in the **choices** list. The overwrite logic will take care of this even though the **extra context choices** list does not explicitly specify this behavior.

3.1.1 Use Case #1 - Update 'default' field - 'choices' gets updated

For example, if **default** and **choices** fields of a **variable** named "director_name" look like this:

```
{
   "name": "director_name",
   ...
   "default": "Allan Smithe",
```

```
   "choices": ["Allan Smithe", "Ridley Scott", "Victor Fleming", "John Ford", "John␣
↪Houston"],
   '''
}
```

and the **extra context** dictionary specified by the user looks like this (injecting an update to the **default** field):

```
'extra_context': [
    {
        'name': 'director_name',
        'default': 'John Ford',
    }
]
```

then the overwrite logic will leave the fields looking like this:

```
{
   "name": "director_name",
   ...
   "default": "John Ford",
   "choices", ["John Ford", "Allan Smithe", "Ridley Scott", "Victor Fleming", "John␣
↪Houston"],
   ...
}
```

3.1.2 Use Case #2 - Update 'choices' field - 'default' field gets updated

For example, if **default** and **choices** fields of a **variable** named "director_name" look like this:

```
{
   "name": "director_name",
   ...
   "default": "Allan Smithe",
   "choices": ["Allan Smithe", "Ridley Scott", "Victor Fleming", "John Ford", "John␣
↪Houston"],
   ...
}
```

and the **extra context** dictionary looks like this (injecting an update to the **choices** field):

```
'extra_context': [
    {
        'name': 'director_name',
        'choices': ['Ridley Scott', 'Allan Smithe', 'Victor Fleming', 'John Ford',
↪'John Houston'],
    }
]
```

then the overwrite logic will leave the fields looking like this:

```
{
   "name": "director_name",
   ...
   "default": "Ridley Scott",
   "choices": ["Ridley Scott", "Allan Smithe", "Victor Fleming", "John Ford", "John␣
↪Houston"],
```

```
    ...
}
```

3.1.3 Use Case #3 - Update both 'choices' & 'default' fields

For example, if **default** and **choices** fields of a **variable** named "director_name" look like this:

```
{
    "name": "director_name",
    ...
    "default": "Allan Smithe",
    "choices": ["Allan Smithe", "Ridley Scott", "Victor Fleming", "John Ford", "John␣
↪Houston"],
    ...
}
```

and the **extra context** looks like this (injecting updates to both the **default** and the **choices** fields):

```
'extra_context': [
    {
        'name': 'director_name',
        'default': 'Victor Fleming',
        'choices': ['Ridley Scott', 'Allan Smithe', 'Victor Fleming', 'John Ford',
↪'John Houston'],
    }
]
```

then the overwrite logic will leave the **choices** and **default** fields updated as follows:

```
{
    "name": "director_name",
    ...
    "default": "Victor Fleming",
    "choices": ["Victor Fleming", "Allan Smithe", "Ridley Scott", "John Ford", "John␣
↪Houston"],
    ...
}
```

3.1.4 Use Case #4 - Update 'default' field, but its not in the 'choices' list

For example, if **default** and **choices** fields of a **variable** named "director_name" look like this:

```
{
    "name": "director_name",
    ...
    "default": "Allan Smithe",
    "choices": ["Allan Smithe", "Ridley Scott", "Victor Fleming", "John Ford", "John␣
↪Houston"],
    ...
}
```

and the **extra context** looks like this (injecting a director name that is not in the **choices** list):

```
'extra_context': [
    {
        'name': 'director_name',
        'default': 'Otto Preminger',
    }
]
```

then the overwrite logic will leave the **choices** and **default** fields updated as follows:

```
{
   "name": "director_name",
   ...
   "default": "Otto Preminger",
   "choices": ["Otto Preminger", "Allan Smithe", "Ridley Scott", "Victor Fleming",
↪"John Ford", "John Houston"],
   ...
}
```

3.2 Special Overwrite Syntax for Renaming a Variable

Because the algorithm chosen to find a variable's dictionary entry (in the variables list of OrderDicts) uses the variable's 'name' field; it could not be used to simultaneously hold a new 'name' field value.

Therefore the following **extra context** dictionary entry snytax was introduced to allow the 'name' field of a variable to be changed:

```
{
   'name': 'CURRENT_VARIABLE_NAME::NEW_VARIABLE_NAME',
}
```

The variable's current name is post-fixed with a double colon (::) followed by the new name of the variable.

For example, to change a variable's 'name' field from 'director_credit' to 'producer_credit', would require:

```
{
   'name': 'director_credit::producer_credit',
}
```

The overwrite logic also takes care of updating in other references to the variable's name that might exists elsewhere in the variable – for example, if the variable's name were used in an a **skip_if** field.

3.3 Special Overwrite Syntax for Removing a Field from a Variable

It is possible that a previous **extra context** overwrite requires that a subsequent variable field be removed.

In order to accomplish this a **remove field token** is used in the **extra context** as follows:

```
{
   'name': 'director_cut',
   'skip_if': '<<REMOVE::FIELD>>',
}
```

In the example above, the **extra context** overwrite results in the variable named 'director_cut' having it's 'skip_if' field removed.

Of course the **name** field and the **default** field cannot be removed from a variable, their existence is mandatory. Any attempt to remove one of these fields will result in an exception.

CHAPTER 4

Document Repository

The source for this document is on GitHub.

CHAPTER 5

Appendix

This appendix houses the primary Cookiecutter project template that drove the implementation decisions made in the reference implementation referenced in this guide:

```
{
    "name": "python-project-skeleton-template",
    "cookiecutter_version": "2.0.0",
    "_inception": "Transformed by cctconvert 1.0.1 Fri Nov  3 20:17:29 2017",
    "description": "Cookiecutter template for a general purpose Python project␣
↪skeleton",
    "authors": ["E.R. Uber"],
    "license": "MIT",
    "keywords": ["cookiecutter", "python", "project", "template", "skeleton"],
    "url": "https://github.com/eruber/python-project-skeleton",
    "variables": [
        {
            "name": "author_name",
            "default": "E.R. Uber",
            "description": "Identify the author of this project.",
            "prompt": "Enter the author's name",
            "type": "string"
        },
        {
            "name": "author_email",
            "default": "eruber@gmail.com",
            "prompt": "Enter the author's email address",
            "type": "string"
        },
        {
            "name": "project_name",
            "default": "Project Skeleton",
            "prompt": "Enter a short, space delimited, name for the project",
            "type": "string"
        },
        {
            "name": "project_version",
```

```
            "default": "0.0.1",
            "description": "Enter the project's semantic version number (see: semver.
↪org).",
            "prompt": "A semantic version number is of the basic form: MAJOR.MINOR.
↪PATCHLEVEL",
            "validation": "^([0-9]|[1-9]+[0-9]*)\\.([0-9]|[1-9]+[0-9]*)\\.([0-9]|[1-
↪9]+[0-9]*)(-)?(-[0-9A-Za-z-\\.]*)*(\\+)?(\\+[0-9A-Za-z-\\.]*)*$",
            "validation_msg": "Follow the form X.Y.Z where X, Y, and Z are non-
↪negative integers, and MUST NOT contain leading zeroes.",
            "type": "string"
        },
        {
            "name": "project_dist",
            "default": "{{ cookiecutter.project_name.lower().replace(' ', '-') }}-{{␣
↪cookiecutter.project_version }}",
            "prompt_user": false,
            "type": "string"
        },
        {
            "name": "project_repo",
            "default": "python-{{ cookiecutter.project_name.lower().replace(' ', '-')␣
↪}}",
            "prompt": "Enter the project's repository name",
            "type": "string"
        },
        {
            "name": "project_pkg",
            "default": "{{ cookiecutter.project_repo.replace('-', '_') }}",
            "prompt": "Enter the project's Python package name",
            "type": "string"
        },
        {
            "name": "project_description",
            "default": "A general purpose Python project skeleton",
            "prompt": "Enter a short description of the project",
            "type": "string"
        },
        {
            "name": "project_license",
            "default": "Apache2",
            "prompt": "Select the project's Open Source License",
            "type": "string",
            "choices": [
                "Apache2",
                "BSD3",
                "ISC",
                "MIT",
                "GNU-GPL-v3"
            ]
        },
        {
            "name": "project_cmdline_interface",
            "default": "none",
            "description": "Select a Command Line Interface for the project.",
            "prompt": "If the project will have no Command Line Interface, select none
↪",
            "type": "string",
            "choices": [
```

```
                "none",
                "click"
            ]
        },
        {
            "name": "project_graphical_inteface",
            "default": "none",
            "description": "Select a Graphical User Interface for the project.",
            "prompt": "If the project will have no Graphical User Interface, select␣
↪none",
            "type": "string",
            "choices": [
                "none",
                "tk",
                "wxwidgets",
                "kivy"
            ]
        },
        {
            "name": "project_shell_interface",
            "default": "none",
            "description": "Select a Shell Interface for the project.",
            "prompt": "If the project will have no Shell Interface, select none",
            "type": "string",
            "choices": [
                "none",
                "cmd",
                "shellocity"
            ]
        },
        {
            "name": "project_machine_interface",
            "default": "none",
            "description": "Select a Machine Interface for the project.",
            "prompt": "If the project will have no Machine Interface, select none",
            "type": "string",
            "choices": [
                "none",
                "api"
            ]
        },
        {
            "name": "project_configuration_enabled",
            "default": true,
            "prompt": "Will this project require a configuration file?",
            "type": "yes_no",
            "if_no_skip_to": "project_uses_existing_logging_facilities"
        },
        {
            "name": "project_config_format",
            "default": "toml",
            "prompt": "Select a configuration file format.",
            "type": "string",
            "choices": [
                "toml",
                "yaml",
                "json",
                "ini"
```

```
            ]
        },
        {
            "name": "project_uses_existing_logging_facilities",
            "default": false,
            "prompt": "Will this project use existing external logging facilities?",
            "type": "yes_no",
            "if_yes_skip_to": "github_username"
        },
        {
            "name": "project_logging_enabled",
            "default": true,
            "prompt": "Will this project provide its own logging facilities?",
            "type": "yes_no",
            "if_no_skip_to": "github_username"
        },
        {
            "name": "project_console_logging_enabled",
            "default": true,
            "prompt": "Will the project's logging facilities include logging to the␣
↪console?",
            "type": "yes_no",
            "if_no_skip_to": "project_file_logging_enabled",
            "do_if": "{{cookiecutter.project_logging_enabled == True}}"
        },
        {
            "name": "project_console_logging_level",
            "default": "WARN",
            "prompt": "Select the minimum logging level to log to the console.",
            "type": "string",
            "choices": [
                "WARN",
                "INFO",
                "DEBUG",
                "ERROR"
            ],
            "do_if": "{{cookiecutter.project_logging_enabled == True}}"
        },
        {
            "name": "project_file_logging_enabled",
            "default": true,
            "prompt": "Will the project's logging facilities include logging to a␣
↪file?",
            "type": "yes_no",
            "if_no_skip_to": "github_username",
            "do_if": "{{cookiecutter.project_logging_enabled == True}}"
        },
        {
            "name": "project_file_logging_level",
            "default": "DEBUG",
            "prompt": "Select the minimum logging level to log to a file",
            "type": "string",
            "choices": [
                "DEBUG",
                "INFO",
                "WARN",
                "ERROR"
            ],
```

```
            "do_if": "{{cookiecutter.project_logging_enabled == True}}"
        },
        {
            "name": "project_file_logging_type",
            "default": "log_to_single_file",
            "prompt": "Select what type of file logging should be used",
            "type": "string",
            "choices": [
                "log_to_single_file",
                "log_to_rotating_file"
            ],
            "do_if": "{{cookiecutter.project_logging_enabled == True}}"
        },
        {
            "name": "github_username",
            "default": "eruber",
            "prompt": "Enter your GitHub User Name",
            "type": "string"
        },
        {
            "name": "test_framework",
            "default": "pytest",
            "description": "Select what type of test framework to use.",
            "prompt": "Selecting none will generate no test framework support",
            "type": "string",
            "choices": [
                "pytest",
                "none"
            ]
        },
        {
            "name": "test_coverage_enabled",
            "default": true,
            "prompt": "Will this project's testing report on test coverage?",
            "type": "yes_no"
        },
        {
            "name": "ci_travis_enabled",
            "default": true,
            "prompt": "Will this project use Continuous Integration facilities␣
↪provided by Travis?",
            "type": "yes_no"
        },
        {
            "name": "ci_appveyor_enabled",
            "prompt": "Will this project use Continuous Integration facilities␣
↪provided by AppVeyor?",
            "default": true,
            "type": "yes_no"
        },
        {
            "name": "project_coding_standards",
            "default": "flake8",
            "description": "Select a coding standards support tool.",
            "prompt": "Selecing none will have the effect of running no code quality␣
↪scans",
            "type": "string",
            "choices": [
```

```
                "flake8",
                "pylama",
                "none"]
        },
        {
            "name": "project_complexity_enabled",
            "prompt": "Should a pytest plugin to run McCabe Code Complexity Checker␣
↪be added to this project?",
            "default": true,
            "type": "yes_no",
            "do_if": "{{ cookiecutter.test_framework == 'pytest' }}"
        },
        {
            "name": "deploy_pypi_enabled",
            "default": true,
            "prompt": "Will this project ultimately be deloyed to Python's Package␣
↪Index site?",
            "type": "yes_no"
        },
        {
            "name": "deploy_readthedocs_enabled",
            "default": true,
            "prompt": "Will this project's documentation ultimately be deployed to␣
↪ReadTheDocs.org?",
            "type": "yes_no"
        },
        {
            "name": "_derived",
            "type": "json",
            "default": {
                "author": "{{ cookiecutter.author_name }} <{{ cookiecutter.author_
↪email }}>",
                "incept_date": "{% now 'local', '%c' %}",
                "project_file_logging_rotating_file_count": "5",
                "github": {
                    "url": "https://github.com/{{ cookiecutter.github_username }}/{{␣
↪cookiecutter.project_repo }}"
                },
                "ci": {
                    "travis": {
                        "username": "{{ cookiecutter.github_username }}",
                        "url": "https://travis-ci.org/{{ cookiecutter.travis_username␣
↪}}/{{ cookiecutter.project_repo }}"
                    },
                    "appveyor": {
                        "username": "{{ cookiecutter.github_username }}",
                        "url": "https://travis-ci.org/{{ cookiecutter.travis_username␣
↪}}/{{ cookiecutter.project_repo }}"
                    }
                },
                "deploy": {
                    "pypi": {
                        "username": "{{ cookiecutter.github_username }}",
                        "url": "https://pypi.python.org/pypi"
                    },
                    "readthedocs": {
                        "username": "{{ cookiecutter.github_username }}",
                        "url_project": "https://readthedocs.org/projects/{{␣
↪cookiecutter.project_repo }}/",
```

```
                        "url_docs": "http://{{ cookiecutter.project_repo }}.
↪readthedocs.io/en/latest/"
                    }
                }
            },
            "prompt_user": false
        }
    ]
}
```

- genindex

Index