Home Assistant Custom Component Cookiecutter

Oncleben31

Jan 24, 2021

CONTENTS

CHAPTER

ONE

QUICKSTART GUIDE

1.1 Requirements

Install Cookiecutter:

```
$ pipx install cookiecutter
```

pipx is preferred, but you can also install with `pip install --user`.

It is recommended to set up Python 3.7, 3.8, or 3.9 using pyenv.

1.2 Creating a project

Generate a Home Assistant custom component project by using the following command:

```
$ cookiecutter gh:oncleben31/cookiecutter-homeassistant-custom-component␣
↪--checkout=2021.1.1
```

Follow the instructions to customize the generated project

Setting	Definition
`friendly_name`	Integration name used in configuration UI.
`project_name`	Project name on GitHub.
`domain_name`	Integration domain name
`class_name_prefix`	Prefix to be use in classes name
`github_user`	GitHub user hosting the repository
`version`	Initial version of the component

Change to the root directory of your new project, and create a Git repository:

```
$ git init
$ git add .
$ git commit
```

1.3 Setup the development container

The development container allows to work in a local and dedicated Home Assitant instance to test your custom component. To launch it you need to have already installed Docker, Visual Studio Code (VSC) and the Visual Studio Code Remote - Containers extension.

Open your local copy of the repository with VSC:

```
$ code .
```

Visual Studio Code starts and you are asked to "Reopen in Container", this will start the build of the container.

When done, you can launch the local instance of Home Assistant by running the task `Run Home Assistant on port 9123`.

Use your preferred browser to open the URL `http://localhost:9123`.

Initalize your Home Assistant local instance by following the onboarding workflow.

When setup, you can go to **Configuration** -> **Integrations** menu, clic the + button and search the name you have given to the custom component.

Follow the config flow of the custom component to integrate it in Home Assistant.

Now you are all set to modify the code and develop your ideas !

1.4 Advanced usages

1.4.1 Add a logo

You have the possibility to add a logo to be used in the integrations configuration UI. To do so, visit the home-assistant/brands repository on GitHub and follow the instructions.

1.4.2 Deploy with HACS

HACS is the community store. You can ease the installation of your custom component by making it compatible with HACS.

The template have already the tools do do that: `hacs.json` and `info.md` files. The Publish documentation explains how to set those files and the different options you have to integrate your custom component in the HACS network.

1.4.3 Step by step debugging

If you choose to use Visual Studio Code with the development container, you can test your custom component in Home Assistant with step by step debugging.

You need to modify the `configuration.yaml` file in `.devcontainer` folder by uncommenting the line:

```
# debugpy:
```

Then launch the task `Run Home Assistant on port 9123`, and launch the debbuger with the existing debugging configuration `Python: Attach Local`.

For more information, look at the Remote Python Debugger integration documentation.

1.5 Known limitations

- **If you plan to host the generated repository in a GitHub organization you will need manual modifications**.

  Currently the template work well when the repostory is hosted in a GitHub individual account, where URL name and code owner are the same. If you want to use an organization, it is recommended to use the name of this organization for `github_user` settings and modify manually where it's needed afer generation with Cookiecutter.

CHAPTER

TWO

USER GUIDE

This is the user guide for the Home Assistant Custom Component Cookiecutter, a Python template for Home Assistant based on the Integration Blueprint repository.

If you're in a hurry, check out the *quickstart guide*.

2.1 Introduction

In progress

2.1.1 About this project

2.1.2 Features

Here is a detailed list of features for this Python template:

- Ready to use Home Assistant custom component
- UI configuration with config Flow
- Translations
- Development, testing and step by step debugging in Visual Studio Code development container
- HACS ready
- Continuous integration with GitHub Actions
- Settings for pre-commit
- Optional tests suite with pytest and code coverage

You can find a repository created with this cookiecutter template in the cookiecutter-homeassistant-custom-component-instance example.

2.1.3 Release cadence

In progress

2.2 Installation

In progress

2.3 Project creation

In progress

2.4 Project overview

This repository contains multiple files, here is a overview:

Table 1: Files list

`.devcontainer/*`	Used for development/testing with VSCODE, more info in the readme file in that dir
`.github/ISSUE_TEMPLATE/feature_request.md`	Template for Feature Requests
`.github/ISSUE_TEMPLATE/issue.md`	Template for issues
`.github/settings.yml`	Probot settings to control the repository settings.
`.vscode/tasks.json`	Tasks for the devcontainer
`custom_components/[DOMAIN NAME]/translations/*`	Translation files
`custom_components/[DOMAIN NAME]/__init__.py`	The component file for the integration
`custom_components/[DOMAIN NAME]/api.py`	This is a sample API client
`custom_components/[DOMAIN NAME]/binary_sensor.py`	Binary sensor platform for the integration
`custom_components/[DOMAIN NAME]/config_flow.py`	Config flow file, this adds the UI configuration possibilities
`custom_components/[DOMAIN NAME]/const.py`	A file to hold shared variables/constants for the entire integration
`custom_components/[DOMAIN NAME]/manifest.json`	A manifest file for Home Assistant
`custom_components/[DOMAIN NAME]/sensor.py`	Sensor platform for the integration
`custom_components/[DOMAIN NAME]/switch.py`	Switch sensor platform for the integration
`custom_components/[DOMAIN NAME]/tests/__init__.py`	Makes the *tests* folder a Python package
`custom_components/[DOMAIN NAME]/tests/conftest.py`	Global fixtures used in tests to patch functions
`custom_components/[DOMAIN NAME]/tests/test_api.py`	Tests for *custom_components/[DOMAIN NAME]/api.py*
`custom_components/[DOMAIN NAME]/tests/test_config_flow.py`	Tests for *custom_components/[DOMAIN NAME]/config_flow.py*
`custom_components/[DOMAIN NAME]/tests/test_init.py`	Tests for *custom_components/[DOMAIN NAME]/__init__.py*
`custom_components/[DOMAIN NAME]/tests/test_switch.py`	Tests for *custom_components/[DOMAIN NAME]/switch.py*
`CONTRIBUTING.md`	Guidelines on how to contribute
`example.png`	Screenshot that demonstrate how it might look in the UI
`info.md`	An example on a info file (used by HACS)
`LICENSE`	The license file for the project
`README.md`	The file you are reading now, should contain info about the integration, installation and configuration instructions
`requirements_dev.txt`	Python packages used to provide IntelliSense/code hints during development of this integration, typically includes packages in `requirements.txt` but

If you want to use all the potential and features of this blueprint template you should use Visual Studio Code to develop in a container. In this container you will have all the tools to ease your python development and a dedicated Home Assistant core instance to run your integration. See `.devcontainer/README.md` for more information.

If you need to work on a Python library in parallel of this integration, there are different options. The following one seems easy to implement:

- Create a dedicated branch for your python library on a public git repository (example: branch *dev*)
- Update in the `manifest.json` file the `requirements` key to point on your development branch (example: `"requirements": ["git+https://github.com/ludeeus/sampleclient.git@dev#devp==0.0.1beta1"]`)
- Each time you need to make a modification to your Python library, push it to your development branch and increase the number of the Python library version in `manifest.json` file to ensure Home Assistant update the code of the python library. (example `"requirements": ["git+https://...==0.0.1beta2"]`).

2.5 Developing in Visual Studio Code with a development container

The easiest way to get started with custom integration development is to use Visual Studio Code with development containers. This approach will create a preconfigured development environment with all the tools you need.

In the container you will have a dedicated Home Assistant core instance running with your custom component code. You can configure this instance by updating the `./devcontainer/configuration.yaml` file.

2.5.1 Prerequisites

- git installed
- Docker - For Linux, macOS, or Windows 10 Pro/Enterprise/Education use the current release version of Docker - Windows 10 Home requires WSL 2 and the current Edge version of Docker Desktop (see instructions here). This can also be used for Windows Pro/Enterprise/Education.
- Visual Studio code
- Remote - Containers (VSC Extension)

More info about requirements and devcontainer in general

2.5.2 Getting started

1. Clone the repository to your computer.
2. Open the repository using Visual Studio code.

When you open this repository with Visual Studio code you are asked to “Reopen in Container”, this will start the build of the container.

If you don’t see this notification, open the command palette and select ``Remote-Containers: Reopen Folder in Container``.

2.5.3 Tasks

The devcontainer comes with some useful tasks to help you with development. You can start these tasks by opening the command palette and select `Tasks: Run Task` then select the one you want to run.

When a task is currently running, it can be restarted by opening the command palette and selecting `Tasks: Restart Running Task`, then select the task you want to restart.

The available tasks are:

Table 2: Tasks list

Task	Description
Run Home Assistant on port 9123	Launch Home Assistant with your custom component code and the configuration defined in `.devcontainer/configuration.yaml`.
Run Home Assistant configuration against /config	Check the configuration.
Upgrade Home Assistant to latest dev	Upgrade the Home Assistant core version in the container to the latest version of the `dev` branch.
Install a specific version of Home Assistant	Install a specific version of Home Assistant core in the container.

2.5.4 Step by Step debugging

With the development container, you can test your custom component in Home Assistant with step by step debugging.

You need to modify the `configuration.yaml` file in `.devcontainer` folder by uncommenting the line:

```
# debugpy:
```

Then launch the task `Run Home Assistant on port 9123`, and launch the debbuger with the existing debugging configuration `Python: Attach Local`.

For more information, look at the Remote Python Debugger integration documentation.

2.6 Linting

In progress

2.7 GitHub Actions Workflows

In progress

2.8 Add a logo

In progress

2.9 Deploy on HACS

In progress

CHAPTER

THREE

CONTRIBUTOR GUIDE

Thank you for your interest in improving the Home Assistant Custom Component Cookiecutter. This project is open-source under the MIT license and welcomes contributions in the form of bug reports, feature requests, and pull requests.

Here is a list of important resources for contributors:

- Source Code
- Documentation
- Issue Tracker
- Code of Conduct

3.1 How to report a bug

Report bugs on the Issue Tracker.

When filing an issue, make sure to answer these questions:

- Which operating system and Python version are you using?
- Which version of this project are you using?
- What did you do?
- What did you expect to see?
- What did you see instead?

The best way to get your bug fixed is to provide a test case, and/or steps to reproduce the issue.

3.2 How to request a feature

Request features on the Issue Tracker.

3.3 How to set up your development environment

You need Python 3.7+ and the following tools:

- Cookiecutter
- Nox
- Docker
- Visual Studio Code

Fork the repository on GitHub, and clone the fork to your local machine. You can now generate a project from your development version:

```
$ cookiecutter path/to/cookiecutter-homeassistant-custom-component
```

3.4 How to test the project

This template repository has a Continuous Integration workflow that generate a default project from the cookiecutter template and run tests on it.

You can also manually test the generated integration with different tools implemented in Github workflows.

This template repository is linked to a generated instance repository for testing purpose. You can generate this instance by using the following command:

```
$ ./scripts/gen_instance
```

You will be able to use the GitHub workflow if you push the generated repository on GitHub. You can run pre-commit or pytest suite manually and you can ensure the integration generated is working in Home Assistant when launched by Visual Studio Code in a devcontainer.

Plese refere to generated project `CONTRIBUTING.rst` file for detailed instructions on testing.

3.5 How to submit changes

Open a pull request to submit changes to this project.

Your pull request needs to meet the following guidelines for acceptance:

- The Nox test suite must pass without errors and warnings.
- Include unit tests. This project maintains 100% code coverage.
- If your changes add functionality, update the documentation accordingly.

Feel free to submit early, though—we can always iterate on this.

It is recommended to open an issue before starting work on anything. This will allow a chance to talk it over with the owners and validate your approach.

3.6 How to accept changes

You need to be a project maintainer to accept changes.

Before accepting a pull request, go through the following checklist:

- The PR must pass all checks.
- The PR must have a descriptive title.
- The PR should be labelled with the kind of change (see below).

Release notes are pre-filled with titles and authors of merged pull requests. Labels group the pull requests into sections. The following list shows the available sections, with associated labels in parentheses:

- Breaking Changes (`breaking`)
- Features (`enhancement`)
- Removals and Deprecations (`removal`)
- Fixes (`bug`)
- Performance (`performance`)
- Testing (`testing`)
- Continuous Integration (`ci`)
- Documentation (`documentation`)
- Refactoring (`refactoring`)
- Style (`style`)
- Dependencies (`dependencies`)

To merge the pull request, follow these steps:

1. Click **Squash and Merge**. (Select this option from the dropdown menu of the merge button, if it is not shown.)
2. Click **Confirm squash and merge**.
3. Click **Delete branch**.

3.7 How to make a release

You need to be a project maintainer to make a release.

Before making a release, go through the following checklist:

- All pull requests for the release have been merged.
- The default branch passes all checks.

Releases are made by publishing a GitHub Release. A draft release is being maintained based on merged pull requests. To publish the release, follow these steps:

1. Click **Edit** next to the draft release.
2. Enter a tag with the new version.
3. Enter the release title, also the new version.
4. Edit the release description, if required.

5. Click **Publish Release**.

Version numbers adhere to Calendar Versioning, of the form `YYYY.MM.Micro`.

After publishing the release, the following automated steps are triggered:

- The Git tag is applied to the repository.
- Read the Docs builds a new stable version of the documentation.

CHAPTER

FOUR

CONTRIBUTOR COVENANT CODE OF CONDUCT

4.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

4.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

4.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

4.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

4.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at oncleben31@gmail.com. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

4.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

4.6.1 1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

4.6.2 2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

4.6.3 3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4.6.4 4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

4.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

CHAPTER

FIVE

MIT LICENSE

This repository is a Cookiecutter template for a Home Assistant custom component based on the integration_blueprint template.
This project is sort of fusion of cookiecutter-homeassistant-component, integration_blueprint and cookiecutter-hypermodern-python projects.

CHAPTER
SIX

USAGE

It is recommended to use the latest stable version by using the command:

```
$ cookiecutter gh:oncleben31/homeassistant-custom-component \
  --checkout=2021.1.1
```

CHAPTER
SEVEN

FEATURES

- Ready to use Home Assistant custom component
- UI configuration with config Flow
- Translations
- Development, testing and step by step debugging in Visual Studio Code development container
- HACS ready
- Continuous integration with GitHub Actions
- Settings for pre-commit
- Optional tests suite with pytest and code coverage

You can find a repository created with this cookiecutter template in the
cookiecutter-homeassistant-custom-component-instance example.

CHAPTER
EIGHT

FAQ

What is this project about?

The mission of this project is to provide Home Assistant custom component developers a ready-to-use template with best practices from Home Assistant developers documentation and from Hypermodern Python blog articles.

Where does the custom component come from?

The code of the custom component generated by this cookiecuter has been intitated and maintained by @Ludeeus. I try to keep synchronized with it and implement additional features.