Awesome Panel Documentation

Marc Skov Madsen

Oct 19, 2022

Awesome Panel!

This is the documentation of the **Awesome Panel Project** including the

- Awesome Panel Resources List on GitHub
- Repo on GitHub
- App at awesome-panel.org
- Docs on Read The Docs
- Python Package on PyPi
- Docker Image on Docker Hub

CHAPTER 1

Awesome Panel Awesome

A powerful, high-level app and dashboarding solution for **Python**!

A repository for sharing knowledge on the use of Panel for developing **awesome analytics apps** in Python.

This project provides

- A curated list of Awesome Panel **resources**. See below.
- An **awesome Panel application** with a **gallery** of Awesome Panel Apps.
  - Feel free to add your awesome app to the gallery via a Pull request. It's easy (see below).
- A **best practices** example and **starter template** of an awesome, multipage app with an automated CI/ CD pipeline, deployed to the cloud and running in a Docker container.

Visit the app at awesome-panel.org!

Awesome Panel Org Animation

1.1 The Power of Panel

The only way to truly understand how powerful Panel is to play around with it. But if you need to be convinced first, then here is the **30 minute introduction** to Panel!

Afterwards you can go to the Panel Getting Started Guide or visit the Panel Gallery.

Introduction to Panel

Panel is completely open source, available under a BSD license freely for both commercial and non-commercial use. Panel is part of the HoloViz ecosystem and works well with all the HoloViz tools..

Panel is developed and maintained by Anaconda developers and community contributors

Anaconda

1.2 Awesome Resources

A curated list of awesome panel resources. Inspired by awesome-python and awesome-pandas.

1.2.1 Alternatives

- Bokeh by Bokeh (#Alternatives)
- Jupyter Voila by Voila (#Alternatives)
- Plotly Dash by Plotly (#Alternatives)
- Streamlit by Streamlit (#Alternatives)

1.2.2 App

- Building Dashboards. Introduction to Data Analysis in Biological Sciences. by Justin Bois (#App, #Inspiration)
- Color Dropper App by Andrew Huang (#App)
- XrViz by Intake (#App, #Code, #Inspiration)

1.2.3 Article

- A tour (of a small part) of the Python visualization landscape by Philipp Rudiger (#Article)
- Dashboards with PyViz Panel for interactive web apps by Damien Farrell (#Article)

1.2.4 Awesome-panel.org

- Awesome-panel.org by Marc Skov Madsen (#Awesome-panel.org)
- Docker by Marc Skov Madsen (#Awesome-panel.org)
- Docs by Marc Skov Madsen (#Awesome-panel.org)
- Github by Marc Skov Madsen (#Awesome-panel.org)
- PyPi by Marc Skov Madsen (#Awesome-panel.org)

1.2.5 Code

- Elvis - Golden Layout by Leon van Kouwen (#Code)

1.2.6 Inspiration

- Information is Beautiful by Information is beautiful (#Inspiration)
- Our World in Data by Our World in Data (#Inspiration)

1.2.7 Panel

- Announcing Article by Philipp Rudiger (#Panel)
- Discourse by panel (#Panel)
- Gallery by panel (#Panel)
- Getting Started by panel (#Panel)
- GitHub by panel (#Panel)
- Panel by panel (#Panel)
- Reference Gallery by panel (#Panel)
- User Guide by panel (#Panel)

1.2.8 Sister Sites

- awesome-streamlit.org by Marc Skov Madsen (#Sister Sites)

1.2.9 Tutorial

- HoloViz.org - Awesome Resources and Tutorials by HoloViz (#Tutorial)
- Information is Beautiful by Bokeh (#Article, #Tutorial)
- VTK Examples by xavArtley by xavArtely (#Inspiration, #Tutorial, #VTK)

1.2.10 Video

- Open Source Directions ep. 29: Panel by Quansight (#Video)
- Turn any Notebook into a Deployable Dashboard | SciPy 2019 | James Bednar by James A. Bednar (#Tutorial, #Video)
- Turn any notebook into a deployable dashboard|PyData Berlin 2019 by Philipp Rudiger (#Tutorial, #Video)
- Visualize any Data Easily, from Notebooks to Dashboards by James A. Bednar (#Tutorial, #Video)

1.3 Contribute

GitHub Issues and Pull requests are very welcome!

If you believe Awesome Panel is awesome and would like to join as a Core Developer feel free to reach out via datamodelsanalytics.com

1.3.1 How to contribute to the Panel Community

Please join the community in the PyViz/PyvViz channel on Gitter. There as a feature request for a Discuss site to replace Gitter.

1.3.2 How to contribute to the Panel Package and Web Site

You can contribute to the Panel package on GitHub/pyviz/panel or sponsor it by contacting sales@anaconda.com. For more information see the Official About Panel page.

1.3.3 How to Contribute an URL to the Resources List

- Fork this repo
- add the URL to the files
  - `package\awesome_panel\database\resources.py`
    - This includes adding you as an Author in the `package/awesome_panel/database/authors.py` file.
    - This might include creating one or more Tags in the `package/awesome_panel/database/tags.py` file.
  - `README.md`
- Create a pull request.

The above is the perfect scenario. If this is not possible do the best you can and then reach out. I would really like to include your Awesome Panel URL to the resources list.

1.3.4 How to Contribute an App to the Gallery

- Fork this repo and follow the *Getting Started Instructions* below.
- Create a new folder and file `src/pages/gallery/<my_awesome_app>/<my_awesome_app.py` for your app.
  - Your `<my_awesome_app.py>` file should contain a function `def view() -> panel.Column:` that returns your app as a column.
  - Add additional files to the folder if you need it.
- Add your app to the `APPS_IN_GALLERY` list in the `package/awesome_panel/database/apps_in_gallery.py` file.
  - This includes adding you as an Author in the `package/awesome_panel/database/authors.py` file.
  - This might include creating one or more Tags in the `package/awesome_panel/database/tags.py` file.
  - This includes creating a Thumbnail of your app and saving it to `assets/images/thumbnails/` folder.
- Run `panel serve app.py` and manully test your app
- Run `invoke test.all` and fix all errors. Also fix any warnings if possible.
- Create a pull request.

The above is the perfect scenario. If this is not possible do the best you can and then reach out. I would really like to include your Awesome Panel App in the Gallery.

1.3.5 How to Sponsor the Awesome Panel Project

If you would like to **sponsor my time or the infrastructure** the platform is running on, feel free to reach out. You can find my contact details at datamodelsanalytics.com.

You can also appreciate the work that has already been done if you

Buy me a coffee

Thanks

1.4 Governance

This repo is maintained by me :-)

I'm Marc, Skov, Madsen, PhD, CFA®, Lead Data Scientist Developer at Ørsted

You can learn more about me at datamodelsanalytics.com

I try my best to govern and maintain this project in the spirit of the Zen of Python.

But **i'm not an experienced open source maintainer** so helpfull suggestions are appreciated.

Thanks

1.4.1 LICENSE

Apache 2.0 License

1.5 Getting Started with the Awesome Panel Repository

1.5.1 Prerequisites

- An Operating System like Windows, OsX or Linux
- A working Python installation.
  - We recommend using 64bit Python 3.7.4.
- a Shell
  - We recommend Git Bash for Windows 8.1
  - We recommend wsl for For Windows 10
- an Editor
  - We recommend VS Code (Preferred) or PyCharm.
- The Git cli

1.5.2 Installation

Clone the repo

```
git clone https://github.com/MarcSkovMadsen/awesome-panel.git
```

cd into the project root folder

```
cd awesome-panel
```

Create virtual environment and install Requirements

via python

Then you should create a virtual environment named .venv

```
python -m venv .venv
```

and activate the environment.

On Linux, OsX or in a Windows Git Bash terminal it's

```
source .venv/Scripts/activate
```

or alternatively

```
source .venv/bin/activate
```

In a Windows terminal it's

```
.venv/Scripts/activate.bat
```

On windows please manually install the geopandas requirements as described in using-geopandas-windows

Then you should install the local requirements

```
pip install -r requirements_local.txt -f https://download.pytorch.org/whl/torch_
↪stable.html
```

or via Anaconda

Create virtual environment named awesome-panel

```
conda create -n awesome-panel python=3.7.4
```

and activate environment.

```
activate awesome-panel
```

On windows please manually install the geopandas requirements as described in using-geopandas-windows

Then you should install the local requirements

```
conda install --file requirements_local.txt
```

1.5.3 Build and run the Application Locally

```
panel serve app.py
```

or in a jupyter notebook

```
jupyter notebook app.ipynb
```

or as a Docker container via

```
invoke docker.build --rebuild
invoke docker.run-server
```

1.5.4 Run the Application using the image on Dockerhub

If you don't want to clone the repo and build the docker container you can just use `docker run` to run the image from Dockerhub

To run the panel interactively on port 80

```
docker run -it -p 80:80 marcskovmadsen/awesome-panel:latest
```

To run bash interactively

```
docker run -it -p 80:80 --entrypoint "/bin/bash" marcskovmadsen/awesome-panel:latest
```

1.6 Build and Deploy the Awesome Panel Package

You can build the package using

```
cd package
python setup.py sdist bdist_wheel
```

If you want to publish the package to PyPi you should first

update the version number in the setup.py file. The format is `YYYYmmdd.version`. For example `20191208.1`

Then you run

```
twine upload dist/awesome-panel-YYYYmmdd.version.tar.gz -u <the-pypi-username> -p
↪<the-pypi-password>
```

1.6.1 Code quality and Tests

We use

- isort for sorting import statements
- autoflake to remove unused imports and unused variables
- black the opinionated code formatter
- pylint for static analysis
- mypy for static type checking

- pytest for unit to functional tests

to ensure a high quality of our code and application.

You can run all tests using

```
invoke test.all
```

1.6.2 Workflow

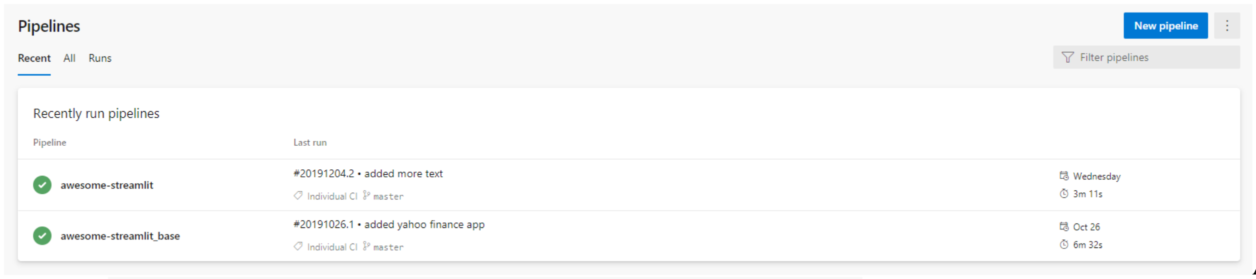
We use the power of Invoke to semi-automate the local workflow. You can see the list of available commands using

```
$ invoke --list
Available tasks:

  docker.build                              Build Docker image
  docker.push                               Push the Docker container
  docker.remove-unused                      Removes all unused containers to free up␣
↪space
  docker.run                                Run the Docker container bash terminal␣
↪interactively.
  docker.run-server                         Run the Docker image with the Panel server.
  docker.run-server-with-ping               Run the docker image with Panel server and
  docker.system-prune                       The docker system prune command will free␣
↪up space
  jupyter.notebook                          Run jupyter notebook
  package.build                             Builds the awesome-panel package)
  sphinx.build                              Build local version of site and open in a␣
↪browser
  sphinx.copy-from-project-root             We need to copy files like README.md into␣
↪docs/_copy_of_project_root
  sphinx.livereload                         Start autobild documentation server and␣
↪open in browser.
  sphinx.test                               Checks for broken internal and external␣
↪links and
  test.all (test.pre-commit, test.test)     Runs isort, autoflake, black, pylint, mypy␣
↪and pytest
  test.autoflake                            Runs autoflake to remove unused imports on␣
↪all .py files recursively
  test.bandit                               Runs Bandit the security linter from PyCQA.
  test.black                                Runs black (autoformatter) on all .py files␣
↪recursively
  test.isort                                Runs isort (import sorter) on all .py files␣
↪recursively
  test.mypy                                 Runs mypy (static type checker) on all .py␣
↪files recursively
  test.pylint                               Runs pylint (linter) on all .py files␣
↪recursively to identify coding errors
  test.pytest                               Runs pytest to identify failing tests
```

1.6.3 CI/ CD and Hosting

The application is

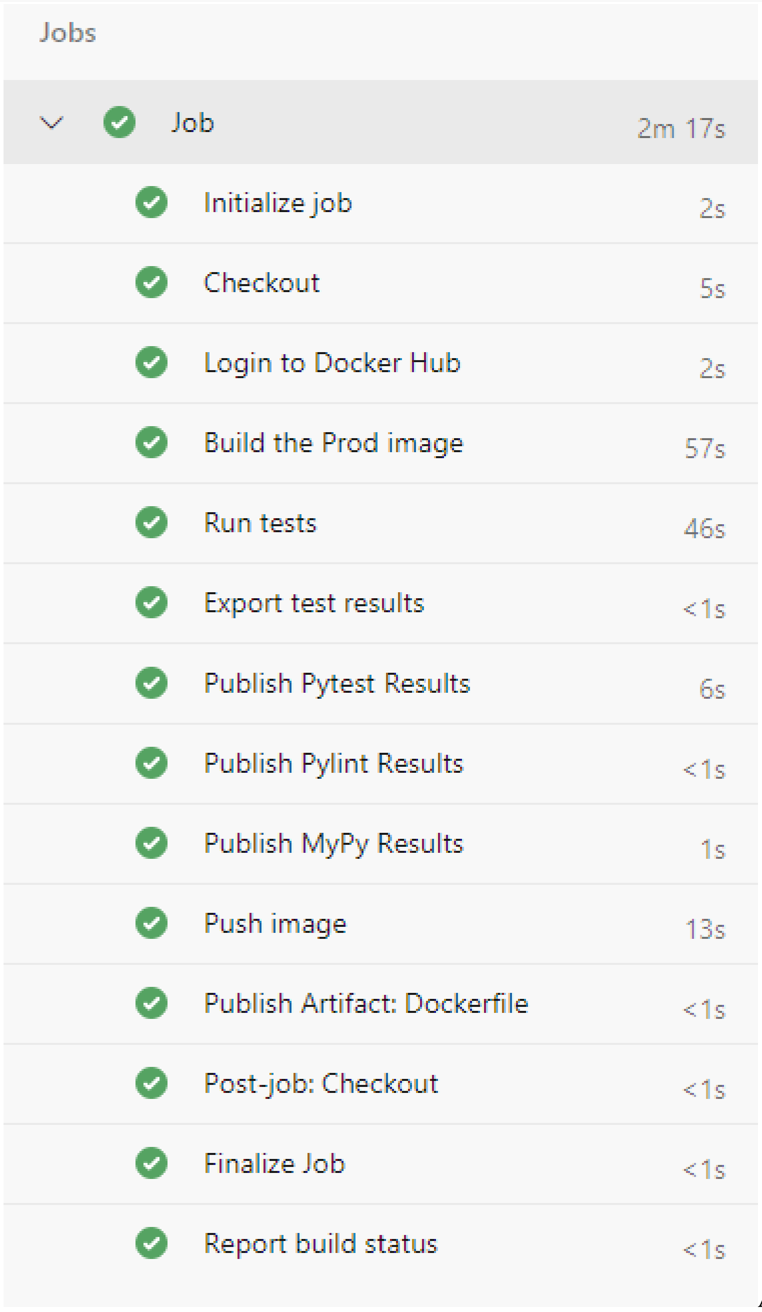
- built as a Docker image and tested via Azure Pipelines.
  - You can find the Dockerfiles here and the Azure pipelines yml files here.


Pipelines

Recently run pipelines

Pipeline	Last run	
awesome-streamlit	#20191204.2 • added more text Individual CI master	Wednesday 3m 11s
awesome-streamlit_base	#20191026.1 • added yahoo finance app Individual CI master	Oct 26 6m 32s

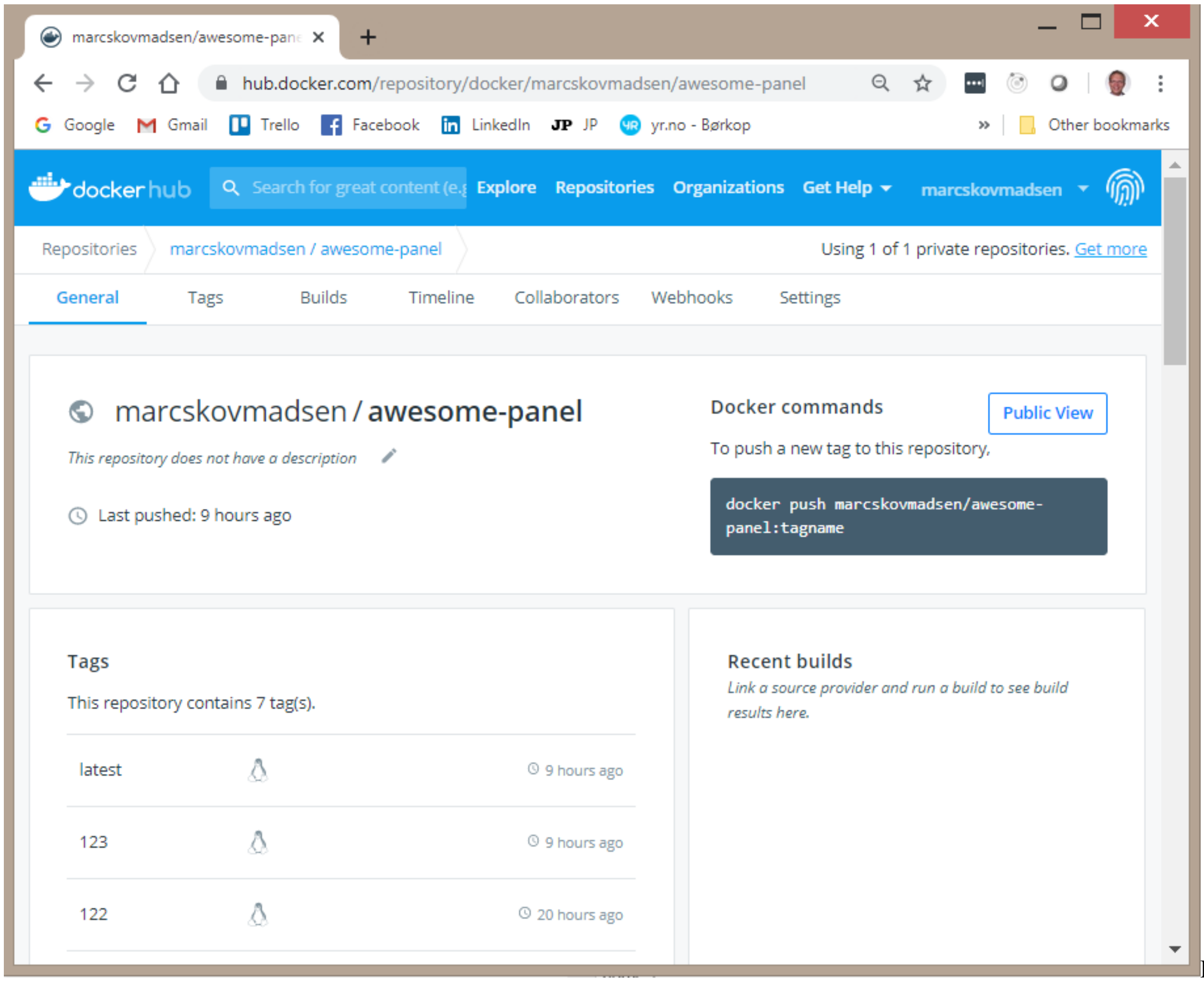
Azure


Jobs

Job	2m 17s
Initialize job	2s
Checkout	5s
Login to Docker Hub	2s
Build the Prod image	57s
Run tests	46s
Export test results	<1s
Publish Pytest Results	6s
Publish Pylint Results	<1s
Publish MyPy Results	1s
Push image	13s
Publish Artifact: Dockerfile	<1s
Post-job: Checkout	<1s
Finalize Job	<1s
Report build status	<1s

Pipelines Azure Pipelines Build and Test

- pushed to the Dockerhub repository marcskovmadsen/awesome-panel.



marcskovmadsen / awesome-panel

This repository does not have a description

Last pushed: 9 hours ago

Docker commands

To push a new tag to this repository,

```
docker push marcskovmadsen/awesome-panel:tagname
```

Tags

This repository contains 7 tag(s).

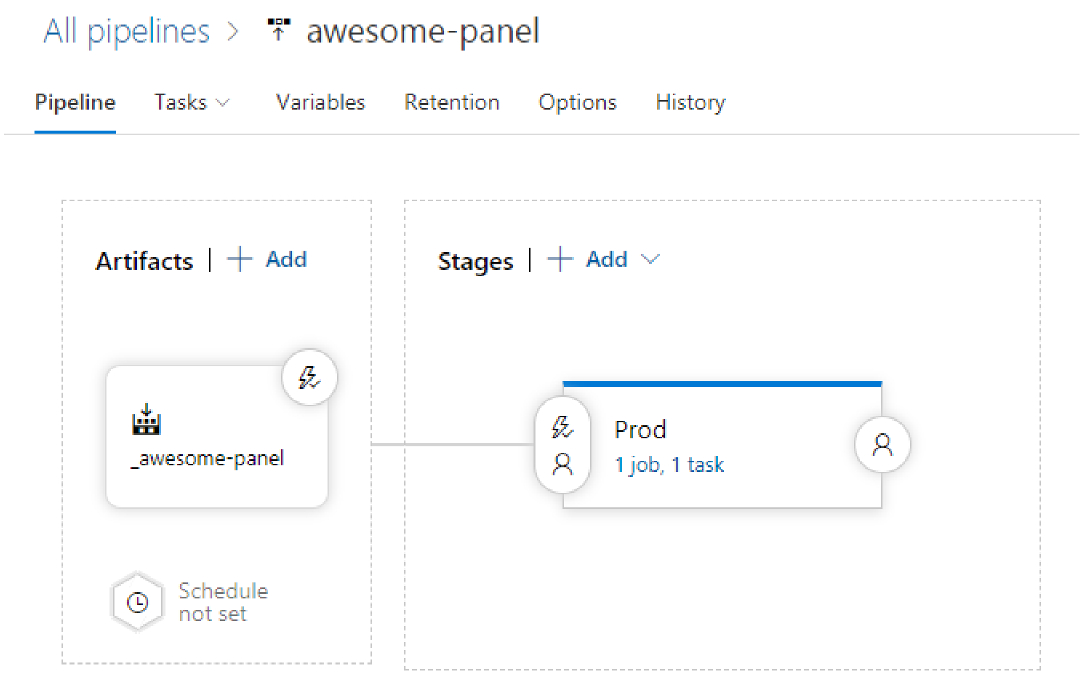
latest	9 hours ago
123	9 hours ago
122	20 hours ago

Recent builds

Link a source provider and run a build to see build results here.

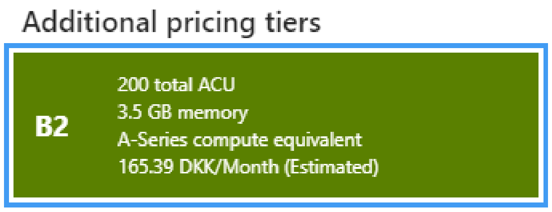
Dockerhub

- released via Azure Pipelines



Azure Pipelines

- to a web app for containers service on Azure on the cheapest non-free pricing tier


Additional pricing tiers

B2
200 total ACU
3.5 GB memory
A-Series compute equivalent
165.39 DKK/Month (Estimated)

Azure Pipelines

1.6.4 Project Layout

The basic layout of a application is as simple as

```
.
└── app.py
```

As our application grows we would refactor our app.py file into multiple folders and files.

- *assets* here we keep our css and images assets.
- *models* - Defines the layout of our data in the form of
  - Classes: Name, attribute names, types
  - DataFrame Schemas: column and index names, dtypes
  - SQLAlchemy Tables: columns names, types
- *pages* - Defines the different pages of the Panel app
- *services* - Organizes and shares business logic, models, data and functions with different pages of the Panel App.
  - Database interactions: Select, Insert, Update, Delete

  - REST API interactions, get, post, put, delete
  - Pandas transformations

and end up with a project structure like

```
.
├── app.py
└── src
    └── assets
    |     └── css
    |     |    ├── app.css
    |     |    ├── component1.css
    |     |    ├── component2.css
    |     |    ├── page1.css
    |     |    └── page2.css
    |     └── images
    |     |    ├── image1.png
    |     |    └── image2.png
    ├── core
    |    └── services
    |        ├── service1.py
    |        └── service2.py
    └── pages
    |    ├── page1.py
    |    └── page2.py
    └── shared
        └── models
        |    ├── model1.py
        |    └── model2.py
        └── components
            ├── component1.py
            └── component2.py
```

Further refactoring is guided by by this blog post and the Angular Style Guide.

We place our tests in a `test` folder in the root folder organized with folders similar to the `app` folder and file names with a `test_` prefix.

```
.
└── test
    ├── test_app.py
    ├── core
    |   └── services
    |       ├── test_service1.py
    |       └── test_service2.py
    └── pages
    |   └── pages
    |       ├── page1
    |       |   └── test_page1.py
    |       └── page2
    └── shared
        └── models
        |   ├── test_model1.py
        |   └── test_model2.py
        └── components
            ├── test_component1.py
            └── test_component2.py
```

CHAPTER 2

Awesome Panel Resources Awesome

CHAPTER 3

Contribution Guidelines

Please note that this project is released with a Contributor Code of Conduct. By participating in this project you agree to abide by its terms.

** The pull request should have a useful title.

3.1 Table of Contents

3.2 Adding to this list

Please ensure your pull request adheres to the following guidelines:

- Search previous suggestions before making a new one, as yours may be a duplicate.
- Make sure the list is useful before submitting. That implies it has enough content and every item has a good succinct description.
- Make an individual pull request for each suggestion.
- Use title-casing (AP style).
- Use the following format: `[List Name](link)`
- Link additions should be added to the bottom of the relevant category.
- New categories or improvements to the existing categorization are welcome.
- Check your spelling and grammar.

- Make sure your text editor is set to remove trailing whitespace.
- The pull request and commit should have a useful title.
- The body of your commit message should contain a link to the repository.

Thank you for your suggestions!

3.3 Creating your own awesome list

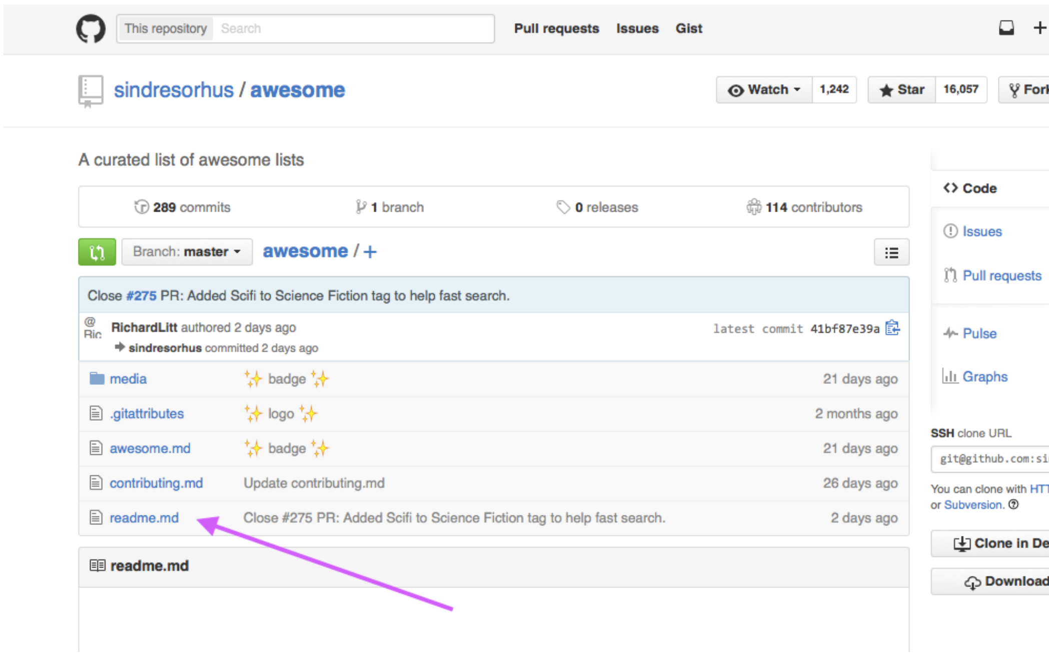
To create your own list, check out the instructions.

3.4 Adding something to an awesome list

If you have something awesome to contribute to an awesome list, this is how you do it.

You’ll need a GitHub account!

1. Access the awesome list’s GitHub page. For example: https://github.com/sindresorhus/awesome



A curated list of awesome lists

289 commits · 1 branch · 0 releases · 114 contributors

Branch: master · awesome / +

Close #275 PR: Added Scifi to Science Fiction tag to help fast search.

RichardLitt authored 2 days ago · latest commit 41bf87e39a
sindresorhus committed 2 days ago

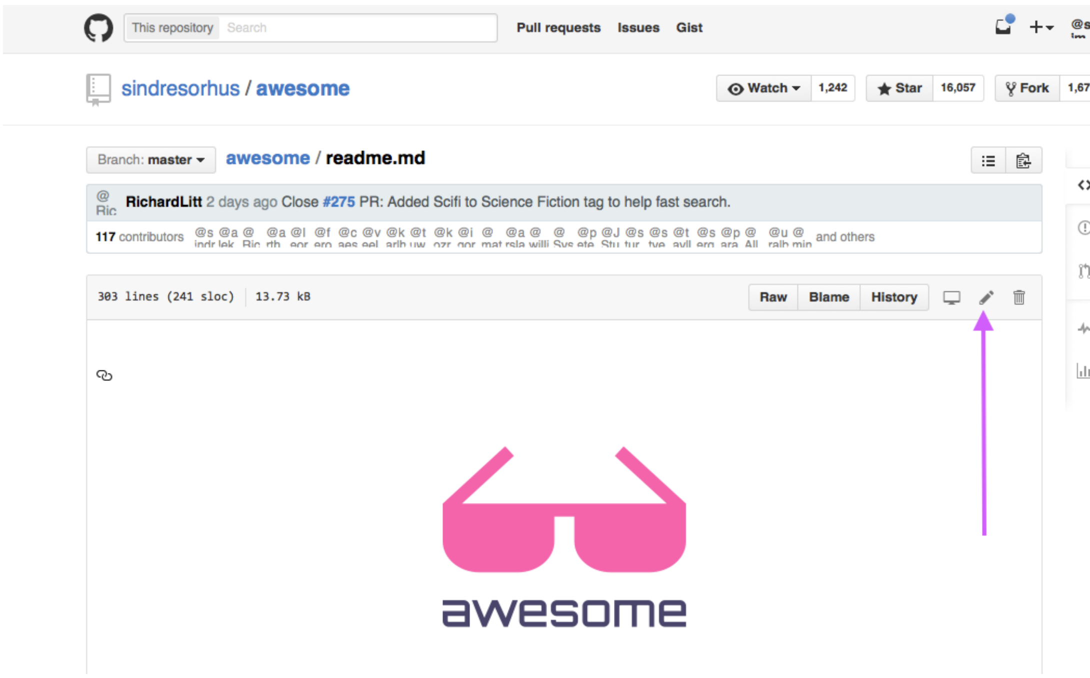
media	✨ badge ✨	21 days ago
.gitattributes	✨ logo ✨	2 months ago
awesome.md	✨ badge ✨	21 days ago
contributing.md	Update contributing.md	26 days ago
readme.md	Close #275 PR: Added Scifi to Science Fiction tag to help fast search.	2 days ago

readme.md

git@github.com:si

You can clone with HT or Subversion.

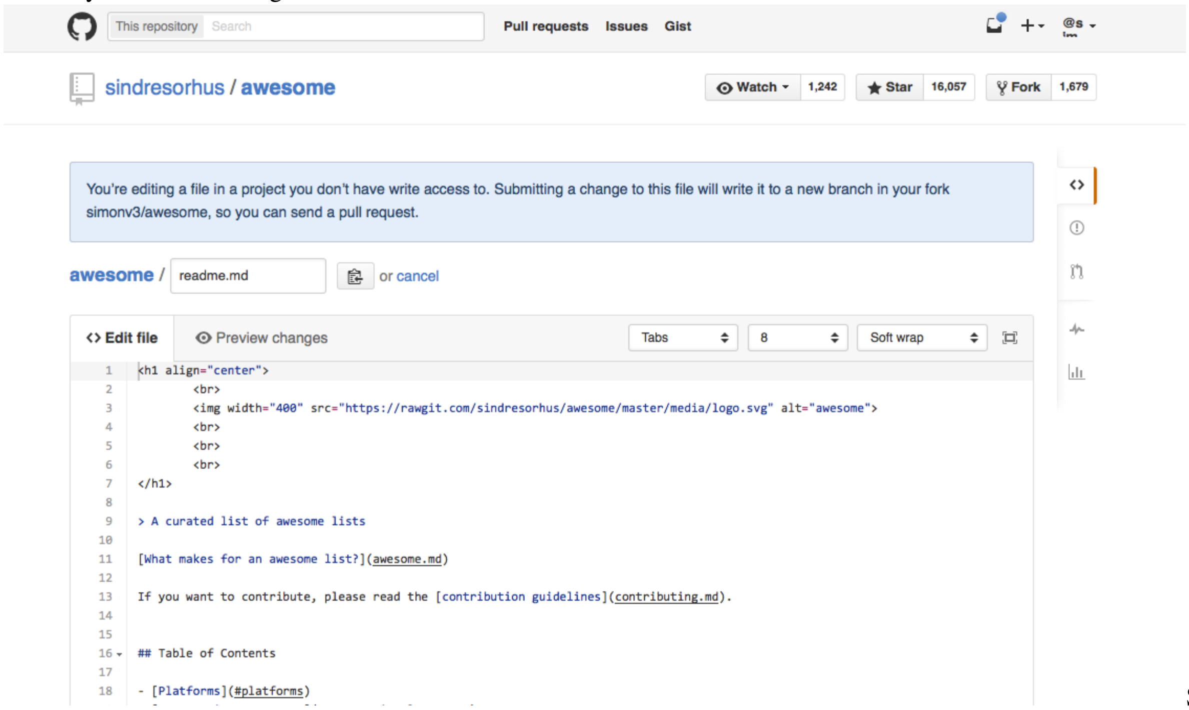
2. Click on the `readme.md` file:
   2 Click on Readme.md



RichardLitt 2 days ago Close #275 PR: Added Scifi to Science Fiction tag to help fast search.

303 lines (241 sloc) 13.73 kB

3. Now click on the edit icon.
   3 - Click on Edit
4. You can start editing the text of the file in the in-browser editor. Make sure you follow guidelines above. You can use GitHub Flavored Markdown.

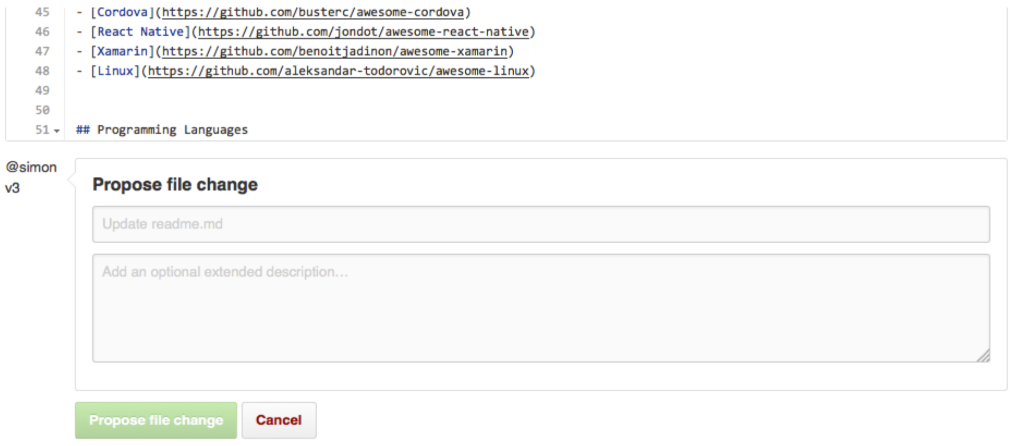


You're editing a file in a project you don't have write access to. Submitting a change to this file will write it to a new branch in your fork simonv3/awesome, so you can send a pull request.

```
<h1 align="center">
	<br>
	<img width="400" src="https://rawgit.com/sindresorhus/awesome/master/media/logo.svg" alt="awesome">
	<br>
	<br>
	<br>
</h1>

> A curated list of awesome lists

[What makes for an awesome list?](awesome.md)

If you want to contribute, please read the [contribution guidelines](contributing.md).


## Table of Contents

- [Platforms](#platforms)
```

Step 4 - Edit the file

5. Say why you're proposing the changes, and then click on "Propose file change".



```
- [Cordova](https://github.com/busterc/awesome-cordova)
- [React Native](https://github.com/jondot/awesome-react-native)
- [Xamarin](https://github.com/benoitjadinon/awesome-xamarin)
- [Linux](https://github.com/aleksandar-todorovic/awesome-linux)


## Programming Languages
```

Step 5 - Propose Changes

6. Submit the pull request!

3.5 Updating your Pull Request

Sometimes, a maintainer of an awesome list will ask you to edit your Pull Request before it is included. This is normally due to spelling errors or because your PR didn't match the awesome-* list guidelines.

Here is a write up on how to change a Pull Request, and the different ways you can do that.

CHAPTER 4

Contributor Covenant Code of Conduct

4.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

4.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

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

4.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers 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, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

4.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

4.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at florian.kromer@mailbox.org. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

4.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at http://contributor-covenant.org/version/1/4

CHAPTER 5

Awesome Panel Extensions Package

The Awesome Panel Extensions package contains Panel Extensions that add to the power of Panel.

5.1 Installation

You can install the package via

```
pip install awesome-panel-extensions
```

Each individual extension might depend on additional packages. For example the `awesome_panel_extensions.panes.PandasProfileReport` depends on the Pandas Profiling package.

5.2 Reference Gallery

Check out the extensions by clicking the images below.

5.2.1 Models

Icon

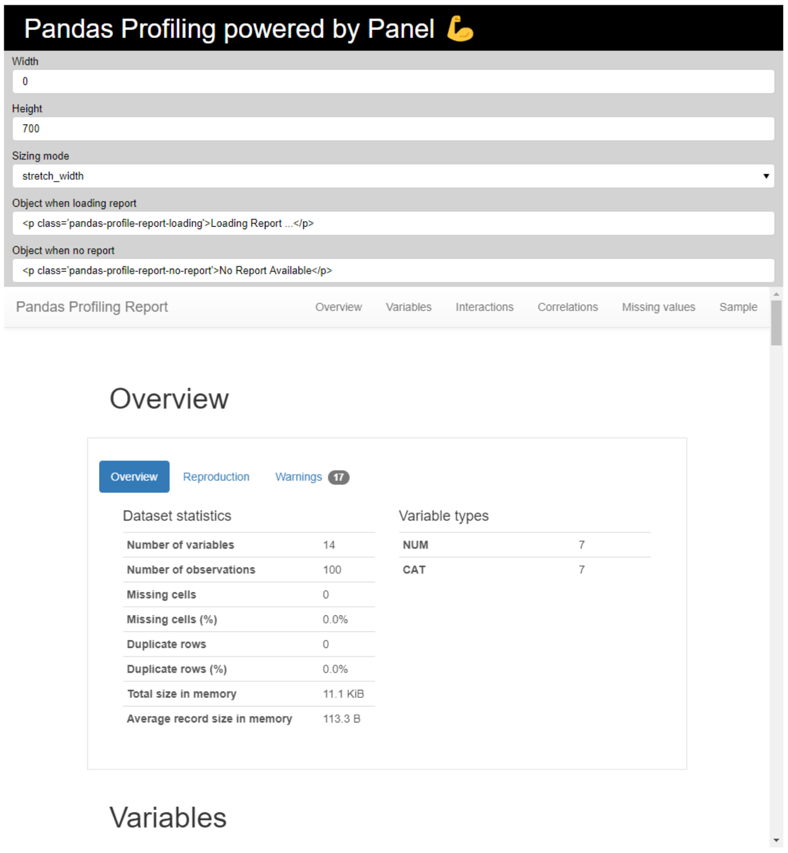
Use it via `from awesome_panel_extensions.models import Icon`.

Icon

Panes

PandasProfileReport

Use it via `from awesome_panel_extensions.pane import PandasProfileReport`.


Pandas Profiling powered by Panel

Width
0

Height
700

Sizing mode
stretch_width

Object when loading report
<p class='pandas-profile-report-loading'>Loading Report ...</p>

Object when no report
<p class='pandas-profile-report-no-report'>No Report Available</p>

Dataset statistics

Number of variables	14
Number of observations	100
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	11.1 KiB
Average record size in memory	113.3 B

Variable types

NUM	7
CAT	7

PandasProfileReport

WebComponent

Use it via `from awesome_panel_extensions.web_component import WebComponent`.

You can think of the `WebComponent` as a `HTML` pane that supports bidirectional communication and large data transfer.

You can use the `WebComponent` to quickly **plugin web component or javascript libraries**.

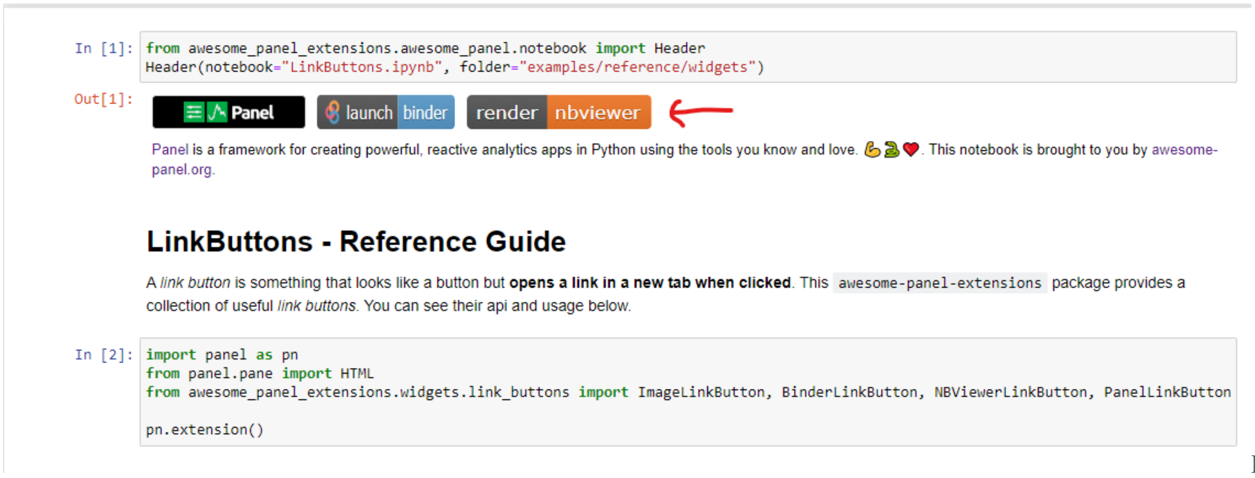
So if you are not satisfied with the look and feel of the existing Panel widgets then use the `WebComponent` to plug in your favourite set of widgets. Or if the `DataFrame` pane or widget is not enough for your use case, then plugin an alternative data table.

For an introduction to *web components* see Web Components: the secret ingredient helping Power the web.

5.2.2 Widgets

LinkButtons

Use them via `from awesome_panel_extensions.widgets import link_buttons`.



```
In [1]: from awesome_panel_extensions.awesome_panel.notebook import Header
        Header(notebook="LinkButtons.ipynb", folder="examples/reference/widgets")
```

Panel is a framework for creating powerful, reactive analytics apps in Python using the tools you know and love. 💪🐍❤️. This notebook is brought to you by awesome-panel.org.

LinkButtons - Reference Guide

A *link button* is something that looks like a button but **opens a link in a new tab when clicked**. This `awesome-panel-extensions` package provides a collection of useful *link buttons*. You can see their api and usage below.

```
In [2]: import panel as pn
        from panel.pane import HTML
        from awesome_panel_extensions.widgets.link_buttons import ImageLinkButton, BinderLinkButton, NBViewerLinkButton, PanelLinkButton

        pn.extension()
```

LinkButtons

PivotTable

The `PivotTable` is a nice, interactive widget for getting insights from data.

Use it via `from awesome_panel_extensions.widgets.pivot_table import PivotTable`

5.2.3 Frameworks

Fast

The Fast extensions are based on the fast.design web components web component which are open sourced by Microsoft and probably will power the VS Code and Microsoft Office experience in the future.

Fast Logo

Please note that you can only use the Fast components inside a custom Panel template that

- Loads the Fast `javascript` library.
- Wraps the content of the `<body>` html tag inside the fast-design-system-provider tag.

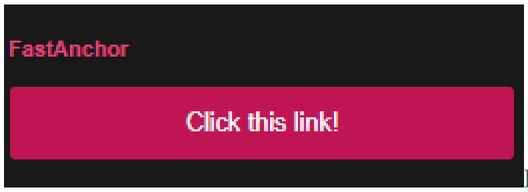
We provide the `FastTemplate` for easy usage.

You can also develop your own custom Panel template if you need something special. For example combining it with more fast.design web components and the Fluent Design System to create **VS Code** and **Microsoft Office** like experiences.

Please also note that the Fast components do not work on legacy browser like Internet Explorer.

FastAnchor

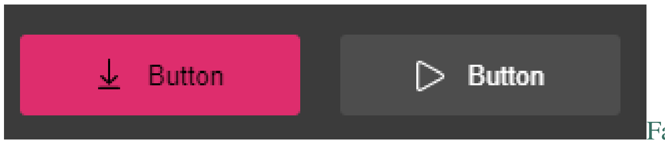
Use it via `from awesome_panel_extensions.frameworks.fast import FastAnchor`



Fast Anchor

FastButton

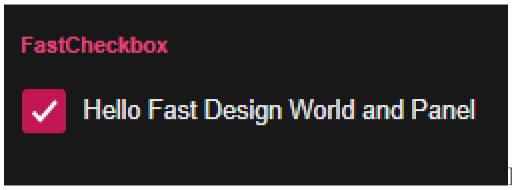
Use it via `from awesome_panel_extensions.frameworks.fast import FastButton`



Fast Button

FastCheckbox

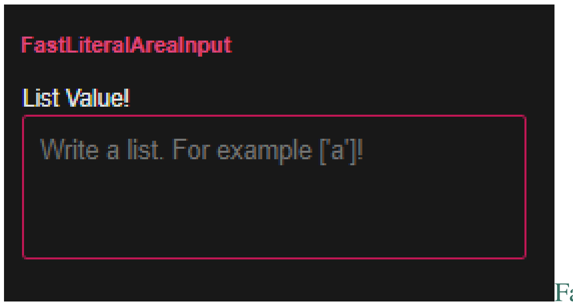
Use it via `from awesome_panel_extensions.frameworks.fast import FastCheckbox`



Fast Checkbox

FastLiteralAreaInput

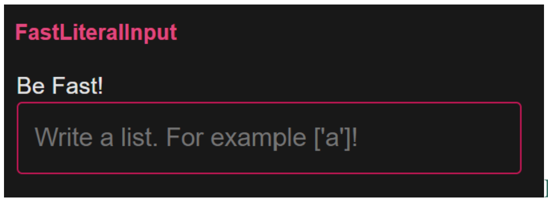
Use it via `from awesome_panel_extensions.frameworks.fast import FastLiteralAreaInput`



FastLiteralAreaInput

FastLiteralInput

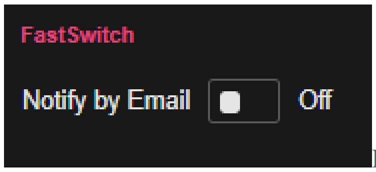
Use it via `from awesome_panel_extensions.frameworks.fast import FastLiteralInput`



Fast LiteralInput

FastSwitch

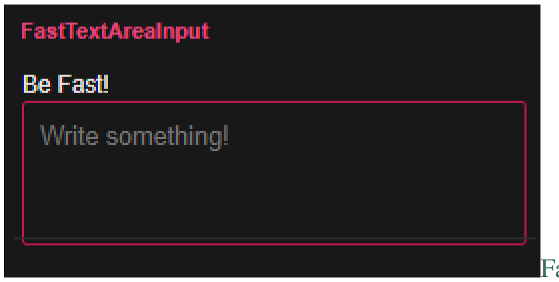
Use it via `from awesome_panel_extensions.frameworks.fast import FastSwitch`



Fast Switch

FastTextAreaInput

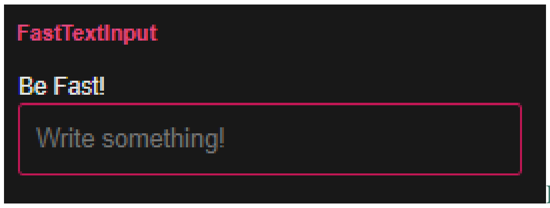
Use it via `from awesome_panel_extensions.frameworks.fast import FastTextAreaInput`



FastTextAreaInput

FastTextInput

Use it via `from awesome_panel_extensions.frameworks.fast import FastTextInput`



Fast TextInput

Material

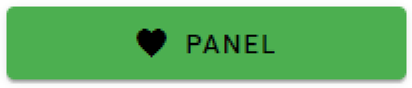
The Material Extensions are based on the MWC Material Web Components and Material Design.

Please note that you need to add the Material `Extension` and `Stylesheet` components to you app to setup the js and css requirements.

Please also note that the Material Widgets **do not work in older browsers** like Internet Explorer.

Material Button

Use it via `from awesome_panel_extensions.frameworks.material import Button`



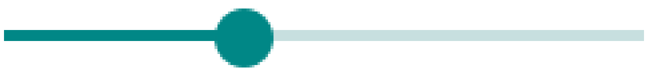
Material Button

Material CircularProgress

Use it via `from awesome_panel_extensions.frameworks.material import CircularProgress`.

Material CircularProgress

Material IntSlider


Material IntSlider

Use it via `awesome_panel_extensions.frameworks.material.IntSlider`.

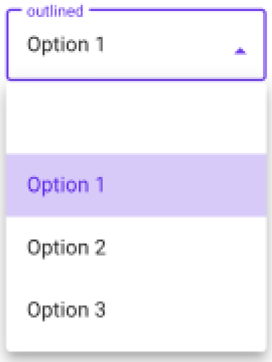
Material LinearProgress

Use it via `from awesome_panel_extensions.frameworks.material import LinearProgress`.

Material LinearProgress

Material Select

Use it via `from awesome_panel_extensions.frameworks.material import Select`



Material Button

5.2.4 Developer Tools

Awesome Panel Designer

Use it via `from awesome_panel_extensions.developer_tools.designer import Designer`.

The *Awesome Panel Designer* is my attempt to create an **efficient workflow for data exploration and development of data apps** in Python **from an editor or IDE**.

This is for **developing any Python object that Panel can display**:

- Strings, Markdown, HTML/ Css/ Javascript
- DataFrames
- Matplotlib, Vega/ Altair, ECharts, Deck.gl, Bokeh, Plotly, HvPlot/ HoloViews, . . .
- Panel layouts, widgets, extensions and apps

For more info click the link below.

- Awesome Panel Designer Docs

CHAPTER 6

The Awesome Panel Extensions Guide

Maybe your code base has started to grow and you start to think about how you can refactor it into smaller reusable components or extensions. For example like this user who asks How to create a self-contained custom Panel?.

Maybe you have started wondering how you can share your extensions with your team? Or maybe even with the Panel community?

Or maybe you are a part of an open source project or company wondering how you can give Panel users easy access to your package, tool or solution in order to increase the usage?

This guide answers your questions. It provides material for developing, testing and deploying **your own Awesome Panel Extensions**.

6.1 Overview

The table below summarizes the types of extensions that Panel supports.

Extension Type	Communication	Datasets	Wrap External JS library	Skill level* (You can do it)
Inheritence Extension				
- Pane				
• HTML	One way	Small	Yes	Basic HTML, CSS and/ or JS
• Markdown	One way	Small	Yes	Markdown
• WebComponent	Bidirectional	Large	Yes	Basic HTML, CSS and/ or JS
• Templates				Jinja, Basic HTML, CSS and/ or JS
- Layout	Bidirectional	Large	Normally No	Panel
View Extension				Same as Inheritance Extensions
Bokeh Extension	Bidirectional	Large	Yes	JS and Typescript
IPyWidget Extension	Bidirectional	Large	Yes	IPyWidget, JS

Inheritence Extensions are extensions that are created by inheriting from an existing layout, pane or widget. Please note that the extension created is often a widget even though its created by inheriting from a layout or pane. Inheritance Extensions are a bit more difficult to develop than View extensions because you need to be a bit more carefull when you inherit. See the detailed guides for more info.

- An important sub category of Inheritence Extensions is called **HTML Extensions**. You create these when you inherit from the *HTML* pane. You can use HTML, CSS and/ or JS to create amazing extensions to Panel. Often the resulting extension works as a widget and not as a pane. The **HTML extensions** cannot communicate from the browser (Javascript) back to the server (Python). The extension developed is often a widget and not a pane.
- Another important sub category of inheritence extensions is called **Layout Extensions**. These extensions are created by inheriting from a Layout and filling it with panes, layouts and widgets. The extension developed is often a widget and not a layout.
- An upcoming, important category of Inheritance Extensions are called **Web Component Extensions**. The *WebComponent* is essentially a *HTML* pane that supports bidirectional communication. It will provide you with the super powers of the Bokeh Extensions below for 80% of your use cases. But they require a minimum of javascript skills and are faster to develop.

View Extensions are developed almost in the same way as **Inheritance Extensions**. Their api is different though. You use *ViewExtension().view* to view a View Extension. View extensions are less quirky to develop and a bit more quirky to use compared to Inheritance Extensions.

Bokeh Extensions supports efficient, bidirectional communication from server (Python) to the browser (Javascript) and back. It also gives you access to develop using all the super powers of modern front end framework languages (js or typescript), tooling and frameworks (React, Vue and Angular). The layouts, panes and widgets that ships with Panel are Bokeh extensions.

IPyWidgets Extensions. The upcoming IPyWidget Pane enables users to use IPyWidgets in Panel. Therefore a developer might develop a Panel extension as an IPyWidget. This might come at a performance cost in relation to bundle size and general performance. If this matters in practice is yet to be confirmed.

6.1.1 HTML Extensions

HTML Extensions are created by inheriting from the `HTML` pane. You can use HTML, CSS and/ or JS to create amazing extensions to Panel. These extensions cannot communicate from the browser (Javascript) back to the server (Python).

Example

In this example we will develop a `Dynamic Number` extension that can display a number with the fontsize and green+alpha color ratios depending on the value.

Dynamic Number Video

We start by importing the dependencies

```
import panel as pn
import param
```

Then we implement the HTML extension.

```
class DynamicNumber(pn.pane.HTML):
    """Extension Implementation"""
    value = param.Integer(default=30, bounds=(0,100))

    # In order to not be selected by the `pn.panel` selection process
    # Cf. https://github.com/holoviz/panel/issues/1494#issuecomment-663219654
    priority = 0
    # The _rename dict is used to keep track of Panel parameters to sync to Bokeh␣
↪properties.
    # As value is not a property on the Bokeh model, we set the it to None
    _rename = dict(pn.pane.HTML._rename, value=None)

    def __init__(self, **params):
        super().__init__(**params)
        self._update_object_from_parameters()

    # Don't name the function
    # `_update`, `_update_object`, `_update_model` or `_update_pane`
    # as this will override a function in the parent class.
    @param.depends("value", watch=True)
    def _update_object_from_parameters(self, *events):
        self.object = self._get_html(self.value)

    def _get_html(self, value):
        """Main functionality of Extension"""
        font_size = value
        alpha = 1-value/100
        green = int(value*255/100)
        return f"""
    <div style="font-size: {font_size}px;color: rgba(0,{green},0,{alpha}">{value}</
↪div>
    """
```

Finally we try out the extension

```
# Create app
extension = DynamicNumber(width=125, height=125)
```

(continues on next page)

(continued from previous page)

```python
app = pn.Column(
    extension,
    extension.param.value,
    width=150,
)
# Serve the app
app.servable()
```

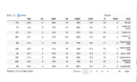
More Examples

Click the images below to see the code.

Pandas Profile Report

Official Panel Examples

The Panel Gallery contains more examples in the section called *External libraries*. Please note that these are not implemented by inheriting from the HTML pane. They just use it. It's not difficult to see how the examples could be converted to inheritance examples though.


Datatable

Folium

Deck.Gl
External Libraries.

6.1.2 Markdown Extensions

Markdown Extensions are created by inheriting from the `Markdown` pane. You can use Markdown (including HTML, CSS and/ or JS) to create amazing extensions to Panel. These extensions cannot communicate from the browser (Javascript) back to the server (Python).

Markdown Example

In this example we will develop a `BinderButton` extension. It could have been implemented as a HTML extension just as well.

Binder Button

We start by importing the dependencies

```python
import param
import panel as pn
```

Then we implement the Markdown extension.

```python
class BinderButton(pn.pane.Markdown):
    """The BinderButton displayes the Binder badge and if clicked opens the Notebook on Binder
    in a new tab"""
    repository = param.String()
    branch = param.String()
```

(continues on next page)

(continued from previous page)

```
    folder = param.String()
    notebook = param.String()

    width = param.Integer(default=200, bounds=(0, None), doc="""
        The width of the component (in pixels). This can be either
        fixed or preferred width, depending on width sizing policy.""")


    # In order to not be selected by the `pn.panel` selection process
    # Cf. https://github.com/holoviz/panel/issues/1494#issuecomment-663219654
    priority = 0

    # The _rename dict is used to keep track of Panel parameters to sync to Bokeh␣
↪properties.
    # As repository etc. is not a property on the Bokeh model we should set it to None
    _rename = dict(pn.pane.Markdown._rename, repository=None, branch=None,␣
↪folder=None, notebook=None)

    def __init__(self, **params):
        super().__init__(**params)

        self._update_object_from_parameters()

    # Note:
    # Don't name the function
    # `_update`, `_update_object`, `_update_model` or `_update_pane`
    # as this will override a function in the parent class.
    @param.depends(
        "repository", "branch", "folder", "notebook", "height", "width", "sizing_mode
↪", watch=True
    )
    def _update_object_from_parameters(self, *events):
        if self.sizing_mode == "fixed":
            style = f"height:{self.height}px;width:{self.width}px;"
        elif self.sizing_mode == "stretch_width":
            style = f"width:{self.width}px;"
        elif self.sizing_mode == "stretch_height":
            style = f"height:{self.height}px;"
        else:
            style = f"height:100%;width:100%;"

        self.object = self.to_markdown(
            repository=self.repository,
            branch=self.branch,
            folder=self.folder,
            notebook=self.notebook,
            style=style,
        )

    @classmethod
    def to_markdown(self, repository: str, branch: str, folder: str, notebook: str,␣
↪style: str = None):
        folder = folder.replace("/", "%2F").replace("\\", "%2F")
        url = f"https://mybinder.org/v2/gh/{repository}/{branch}?filepath={folder}%2F
↪{notebook}"
        if style:
            image = f'<img src="https://mybinder.org/badge_logo.svg" style="{style}">'
```

(continues on next page)

(continued from previous page)

```
        else:
            image = f'<img src="https://mybinder.org/badge_logo.svg">'
        markdown = f"[{image}]({url})"
        return markdown
```

Finally we try out the extension

```
# Create the app
button = BinderButton(
    repository="marcskovmadsen/awesome-panel-extensions",
    branch="master",
    folder="examples/panes",
    notebook="WebComponent.ipynb",
)
settings_pane = pn.WidgetBox(
    pn.Param(
        button, parameters=["repository", "branch", "folder", "notebook", "height",
↪"width", "sizing_mode", "margin"], sizing_mode="stretch_width"
    )
)
app = pn.Column(button, settings_pane, width=500, height=800)
# Serve the app
app.servable()
```

6.1.3 Layout Extensions

Layout Extensions are created by inheriting from a layout. The extensions developed is composed of panes, layouts and widgets. The extension developed is often a widget and not a layout.

Example

In this example we will inherit from `panel.Column`. We will develop a `DataFramePlotter` extension that enables a Panel user to select a column of a given DataFrame and see the associated `distplot`.

Data FramePlotter

We start by importing the requirements

```
import matplotlib.pyplot as plt
import pandas as pd
import panel as pn
import param
import seaborn as sns
```

Then we implement the *Layout Extension*.

```
class DataFramePlotter(pn.Column):
    """Extension Implementation"""
    column = param.Selector()

    # The _rename dict is used to keep track of Panel parameters to sync to Bokeh␣
↪properties.
    # As column is not a property on the Bokeh model we should set it to None
    _rename = {
```

(continues on next page)

(continued from previous page)

```
            **pn.pane.Column._rename,
            'value': None,
        }

    def __init__(self, data, **params):
        super().__init__(**params)

        self._plot_pane = pn.pane.Matplotlib(background="blue", sizing_mode="stretch_
↪both")
        self[:] = [self.param.column, self._plot_pane]

        # Please note that the alternative of setting
        # @param.depends("column", watch=True)
        # on _update_plot_pane does not work.
        # See https://github.com/holoviz/panel/issues/1060
        self.param.watch(self._update_plot_pane, "column")

        columns = data.columns.values
        self.param.column.objects = columns
        # I need to set self.column to show a plot initially
        self.column = columns[0]

    def _update_plot_pane(self, _):
        # - I get exception if plt.close is below ax line. See https://github.com/
↪holoviz/panel/issues/1482
        # - The plot does not change if I remove plot.close() fully.
        plt.close()

        ax = sns.distplot(df[self.column])
        self._plot_pane.object = ax.figure
```

Finally we can use the extension.

```
df = pd.DataFrame(data={"x": [1, 2, 3, 4, 5, 6, 7], "y": [1, 2, 2, 4, 5, 9, 7]})
DataFramePlotter(df, width=300, height=300).servable()
```

More Examples

Click the images below to see the code.

COMING UP

Official Panel Examples

COMING UP

6.1.4 WebComponent Extensions

You can think of the `WebComponent` pane as **a `HTML` pane that supports bidirectional communication and large data transfer**. You can use the `WebComponent` to quickly **plugin web component or javascript libraries**.

For example you can use the `WebComponent` pane to plug in your favourite set of widgets. For example if the `DataFrame` pane or widget is not enough for your use case, then plugin an alternative grid.

For an introduction to *web components* see Web Components: the secret ingredient helping Power the web.

The `WebComponent` is currently distributed via the `awesome-panel-extensions` package. But it is also on the **roadmap for Panel**. So we need your help to identify bugs and improvements or suggestions for improving the api. You can contribute your comments and suggestions via Github PR 1252.

Example

In this example we will develop a mwc-button. It's based on the Material Design Specification and is a part of the MWC library of layouts and widgets.

You will need to install the required packages via `pip install panel param awesome-panel-extensions`.

We will start by importing the dependencies

```
import panel as pn
import param
from awesome_panel_extensions.web_component import WebComponent
```

Then we define the `MWCButton`.

```
MWC_ICONS = [None, "accessibility", "code", "favorite"]

class MWCButton(WebComponent):
    html = param.String("<mwc-button></mwc-button")
    attributes_to_watch = param.Dict({"label": "name", "icon": "icon", "raised":
↪"raised"})
    events_to_watch = param.Dict({"click": "clicks"})

    raised=param.Boolean(default=True)
    icon=param.ObjectSelector(default="favorite", objects=MWC_ICONS, allow_None=True)
    clicks = param.Integer()

    height = param.Integer(default=30)

mwc_button = MWCButton(name="Click Me!")
```

The key part to notice is that we configure the `label`, `icon` and `raised` attributes of the `mwc-button` to the `name`, `icon` and `raised` parameters of the `MWCButton`.

The we need to include the `.js` and `.css` files needed for the `mwc-button`. Currently `panel.extension` does not support importing `module .js` files. So we just use an invisible `HTML` pane.

```
MWC_EXTENSIONS = """
<script type='module' src='https://www.unpkg.com/@material/mwc-button?module'></
↪script>
<link href='https://fonts.googleapis.com/css?family=Roboto:300,400,500' rel=
↪'stylesheet'>
<link href='https://fonts.googleapis.com/css?family=Material+Icons&display=block' rel=
↪'stylesheet'>
<style>
:root {
    --mdc-theme-primary: green;
    --mdc-theme-secondary: purple*;
}
</style>
```

(continues on next page)

(continued from previous page)

```
"""
extensions_pane = pn.pane.Markdown(MWC_EXTENSIONS, height=0, width=0, sizing_mode=
↪"fixed", margin=0)
```

Then we define the app

```
settings_pane = pn.Param(
    mwc_button, parameters=["name", "icon", "raised", "height", "clicks"]
)
app = pn.Column(
    extensions_pane, mwc_button, settings_pane
)
app.servable()
```

and finally we can `panel serve` the app.

MWC Button Image

Reference Guide

Click the links below to get a deeper understanding of the `WebComponent`.

More Examples

COMING UP

6.1.5 View Extensions

View Extensions are `param.Parameterized` classes with a `view` attribute or function. View Extensions are developed almost in the same way as **Inheritance Extensions**. Their api is different though. You use `ViewExtension().view` to view a View Extension and `InheritanceExtension()` to view an Inheritance Extension.

View extensions avoids some of the technical quirks that the Inheritance Extensions comes with at the cost of having to communicate to users they need to append `.view` to view the extension.

Example

In this example we will develop a `Dynamic Number` extension that can display a number with the fontsize and green+alpha color ratios depending on the value.

Dynamic Number Video

We start by importing the dependencies

```
import panel as pn
import param
```

Then we implement the View extension.

```
class DynamicNumber(param.Parameterized):
    """Extension Implementation"""
    value = param.Integer(default=30, bounds=(0,100))
    view = param.ClassSelector(class_=pn.layout.Reactive)

    def __init__(self, **params):
        super().__init__(**params)
        self.view = pn.pane.HTML()
        self._update_object()

    @param.depends("value", watch=True)
    def _update_object(self, *events):
        self.view.object = self._get_html(self.value)

    def _get_html(self, value):
        """Main functionality of Extension"""
        font_size = value
        alpha = 1-value/100
        green = int(value*255/100)
        return f"""
    <div style="font-size: {font_size}px;color: rgba(0,{green},0,{alpha}">{value}</
↪div>
    """
```

Finally we try out the extension

```
# Create app
extension = DynamicNumber()
extension.view.width=125
extension.view.height=125
app = pn.Column(
    extension.view,
    extension.param.value,
    width=150,
)
# Serve the app
app.servable()
```

If you compare this example to corresponding `HTML` Extension example you will notice that

- Developing the `DynamicNumber` class has fewer lines and is less quirky.
- Using the `DynamicNumber` class uses more lines and is a bit more quirky.

So the question really is. Should you make it easier for the developer or the user of the extension? In the first case you would develop it as a View Extension in the Second as a HTML extension.

If you have an opinion please join the discussions

- Discourse - How to create a self-contained, custom "Panel"?
- Github - Document subclassing of different component types
- Github - Please support concept of HTML Extension
- Github - Please support concept of Layout Extension

More Examples

Click the images below to see the code.

Echarts Gauge Video

Official Panel Examples

The Panel Gallery contains more examples in the section called *External libraries*.

External Libraries.

6.1.6 Bokeh Extensions

Bokeh Extensions supports efficient, bidirectional communication from the server (Python) to the browser (Javascript) and back. It also gives you access to all the super powers of modern front end framework languages (js or typescript), tooling and frameworks like React, Vue and Angular. The layouts, panes and widgets that ships with Panel are Bokeh extensions.

Please note that in order for Bokeh Extensions to compile you will need to have node.js installed. You can install it directly from their web site or via `conda install -c conda-forge nodejs`.

Before you read on I would ask you to quickly study the offical Bokeh documentation Extending Bokeh. You don't need to code and run the examples. But you need to get a basic understanding of

- the existence and location of official Bokeh documentation
- what a Bokeh extension is and how it is developed.

We will now focus on Bokeh Extensions in a Panel context.

Example

In this example we will create a Panel `HTMLButton` extension that enables a user to catch a click event from any HTML element he/ she would like as shown below.

html_button.py

The implentation consists of 3 files

- Panel extension file: html_button.py.
- Bokeh extensions files: html_button_model.py and html_button_model.ts

html_button.py

This is the Panel specific file. We need to import the Bokeh python extension and wrap that into a Panel extension.

```python
import panel as pn
from panel.widgets.base import Widget
from . import html_button_model
import param

class HTMLButton(Widget):
    # Set the Bokeh model to use
    _widget_type = html_button_model.HTMLButton

    # Rename Panel Parameters -> Bokeh Model properties
```

(continues on next page)

(continued from previous page)

```
    # Parameters like title that does not exist on the Bokeh model should be renamed␣
↪to None
    _rename = {
        "title": None,
    }

    # Parameters to be mapped to Bokeh model properties
    object = param.String(default=html_button_model.DEFAULT_OBJECT)
    clicks = param.Integer(default=0)
```

html_button_model.py

```
import pathlib

from bokeh.core.properties import Int, String
from bokeh.layouts import column
from bokeh.models import HTMLBox

CUSTOM_TS = pathlib.Path(__file__).parent / "html_button_model.ts"
CUSTOM_TS_STR = str(CUSTOM_TS.resolve())

DEFAULT_OBJECT = "<button style='width:100%'>Click Me</button>"


class HTMLButton(HTMLBox):
    """Example implementation of a Custom Bokeh Model"""

    __implementation__ = CUSTOM_TS_STR

    object = String(default=DEFAULT_OBJECT)
    clicks = Int(default=0)
```

html_button_model.ts

```
// See https://docs.bokeh.org/en/latest/docs/reference/models/layouts.html
import { HTMLBox, HTMLBoxView } from "models/layouts/html_box"

// See https://docs.bokeh.org/en/latest/docs/reference/core/properties.html
import * as p from "core/properties"

// The view of the Bokeh extension/ HTML element
// Here you can define how to render the model as well as react to model changes or␣
↪View events.
export class HTMLButtonView extends HTMLBoxView {
    model: HTMLButton
    objectElement: any // Element

    connect_signals(): void {
        super.connect_signals()

        this.connect(this.model.properties.object.change, () => {
            this.render();
        })
    }

    render(): void {
        console.log("render")
```

(continues on next page)

(continued from previous page)

```
        console.log(this.model)
        super.render()
        this.el.innerHTML = this.model.object
        this.objectElement = this.el.firstElementChild

        this.objectElement.addEventListener("click", () => {this.model.clicks+=1;},␣
↪false)
    }
}

export namespace HTMLButton {
    export type Attrs = p.AttrsOf<Props>
    export type Props = HTMLBox.Props & {
        object: p.Property<string>,
        clicks: p.Property<number>,
    }
}

export interface HTMLButton extends HTMLButton.Attrs { }

// The Bokeh .ts model corresponding to the Bokeh .py model
export class HTMLButton extends HTMLBox {
    properties: HTMLButton.Props

    constructor(attrs?: Partial<HTMLButton.Attrs>) {
        super(attrs)
    }

    static init_HTMLButton(): void {
        this.prototype.default_view = HTMLButtonView;

        this.define<HTMLButton.Props>({
            object: [p.String, "<button style='width:100%'>Click Me</button>"],
            clicks: [p.Int, 0],
        })
    }
}
```

Finally we can use the new Widget in an example app.

```
def _example_app():
    # Default Button
    html_button = HTMLButton()

    # Material Button
    material_js = (
        "https://cdn.jsdelivr.net/gh/marcskovmadsen/awesome-panel"
        "@be59521090b7c9d9ba5eb16e936034e412e2c86b/assets/js/mwc.bundled.js"
    )
    pn.config.js_files["material"]=material_js
    material_html = """\
<link href="https://fonts.googleapis.com/css?family=Roboto:300,400,500" rel=
↪"stylesheet">
<link href="https://fonts.googleapis.com/css?family=Material+Icons&display=block" rel=
↪"stylesheet">
<style>
mwc-button {
```

(continues on next page)

(continued from previous page)

```
    --mdc-theme-primary: #4CAF50;
    --mdc-theme-on-primary: white;
}
</style>
    """
    material_html_pane = pn.pane.HTML(material_html, width=0, height=0, margin=0,␣
↪sizing_mode="fixed")
    material_button = HTMLButton(object="<mwc-button style='width:100%' raised label=
↪'Panel' icon='favorite'></mwc-button>", height=40)

    # Image Button
    src = "https://github.com/holoviz/panel/raw/master/doc/_static/logo_stacked.png"
    image_style = "height:95%;cursor: pointer;border: 1px solid #ddd;border-radius:␣
↪4px;padding: 5px;"
    image_html = f"<img class='image-button' src='{src}' style='{image_style}'>"
    image_button = HTMLButton(object=image_html, height=100, align="center")

    # Bar
    bar = pn.pane.Markdown(
        "## Panel Extension: HTMLButton",
        background="black",
        sizing_mode="stretch_width",
        style={"color": "white", "padding-left": "25px", "padding-top": "10px"},
    )

    app = pn.Column(
        bar,
        material_html_pane,
        html_button,
        html_button.param.clicks,
        material_button,
        material_button.param.clicks,
        image_button,
        image_button.param.clicks,
        width=500,
    )
    return app

_example_app().servable()
```

Other Examples

Click the images below to see the code.

Custom Bokeh Model

NOTE: THE CUSTOM BOKEH MODEL EXAMPLES NEEDS TO BE WRAPPED INTO A PANEL OBJECT. COMING UP.

Official Panel Examples

Every layout, pane or widget in Panel is essentially a Bokeh Extension so a good place to get inspiration is to navigate the Panel Reference Gallery to find an extension similar to the one you would like to implement and then study the code

Panel Reference Gallery

You can find the code of the Panel components on Github via

- Panel Layouts
- Panel Panes
- Panel Widgets

and the underlying Bokeh extensions via

- Bokeh Model Widgets
- Panel Bokeh Models

Prebuilt Bokeh Extensions

There are two ways in which the Bokeh `.ts` models can be built.

- **Automatically** when you run the code.
  1. If you instantiate your extension before running `.servable` then the extension will automatically be built and registered by Panel/ Bokeh.
- **Manually** up front using the `panel build` or `bokeh build`command. 2. This is referred to as *prebuilt Bokeh extensions*.

In this document I will describe how I setup the awesome-panel-extensions package for **prebuilt bokeh extensions**. **This was nescessary to distribute the extensions as a package.**

I hope this description can help others who would like to develop and share Bokeh Extensions for Panel.

Setting up prebuilt extensions using `Bokeh init --interactive` is described in the Bokeh Docs. See Bokeh Pre-built extensions.

Steps for the `awesome-panel-extensions` Package as of 20200721 Panel 0.9.7/ Bokeh 2.1.1

I navigated to the `awesome_panel_extensions` inside the project.

```
cd awesome_panel_extensions
```

I ran `bokeh init --interactive`

```
$ bokeh init --interactive
Working directory: ...\awesome_panel_extensions
Wrote ...\awesome_panel_extensions\bokeh.ext.json
Create package.json? This will allow you to specify external dependencies. [y/n] y
  What's the extension's name? [awesome_panel_extensions]
  What's the extension's version? [0.0.1]
  What's the extension's description? [] A collection of awesome extensions for Panel
Wrote ...\awesome_panel_extensions\package.json
Create tsconfig.json? This will allow for customized configuration and improved IDE␣
↪experience. [y/n] y
Wrote ...\awesome_panel_extensions\tsconfig.json
Created empty index.ts. This is the entry point of your extension.
You can build your extension with bokeh build
All done.
```

In the `package.json` I replaced

```
"dependencies": {
    "bokehjs": "^2.1.1"
  },
```

with

```
"dependencies": {
    "@bokeh/bokehjs": "^2.1.1"
  },
```

in order to import from bokehjs in the same way as Panel does. See bokeh init issue for more info.

I also replaced the `tsconfig.json` contents with

```
{
  "compilerOptions": {
    "noImplicitAny": true,
    "noImplicitThis": true,
    "noImplicitReturns": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "strictNullChecks": true,
    "strictBindCallApply": false,
    "strictFunctionTypes": false,
    "strictPropertyInitialization": false,
    "alwaysStrict": true,
    "noErrorTruncation": true,
    "noEmitOnError": false,
    "declaration": true,
    "sourceMap": true,
    "importHelpers": false,
    "experimentalDecorators": true,
    "module": "esnext",
    "moduleResolution": "node",
    "esModuleInterop": true,
    "resolveJsonModule": true,
    "skipLibCheck": true,
    "target": "ES2017",
    "lib": ["es2017", "dom", "dom.iterable"],
    "baseUrl": ".",
    "outDir": "./dist/lib",
    "paths": {
      "@bokehjs/*": [
        "./node_modules/@bokeh/bokehjs/build/js/lib/*",
        "./node_modules/@bokeh/bokehjs/build/js/types/*"
      ]
    }
  },
  "include": ["./**/*.ts"]
}
```

At least including the `paths` section is needed to be able to `import { div, label } from "@bokehjs/core/dom"` like @philippjfr does in Panel.

In the `index.ts` file I imported my models

```
import * as AwesomePanelExtensions from "./bokeh_extensions/"
export {AwesomePanelExtensions}
```

(continues on next page)

(continued from previous page)

```
import {register_models} from "@bokehjs/base"
register_models(AwesomePanelExtensions as any)
```

In the `bokeh_extensions/index.ts` file I exported the `WebComponent`.

```
export {WebComponent} from "./web_component"
```

Then I could `build` my extension

```
$ panel build
Working directory: C:\repos\private\awesome-panel\package\awesome_panel
Using C:\repos\private\awesome-panel\package\awesome_panel\tsconfig.json
Compiling TypeScript (3 files)
Linking modules
Output written to C:\repos\private\awesome-panel\package\awesome_panel\dist
All done.
```

The result is in the `dist` folder.

I discovered I did not even have to do anything special to `serve` the `awesome_panel_extensions.js` file. It just works.

Finally I added `awesome_panel_extensions/node_modules/*`to my `.gitignore` file.

FAQ

Should I define default values on the Bokeh .ts, Bokeh .py or Panel .py Model

COMING UP.

6.1.7 Sharing Panel Extensions

You can share your awesome Panel extension(s) in the following ways

- Share working code examples with screenshots on HoloViz Discourse.
- Contribute as a notebook to the Panel Gallery.
- Contribute as code to the Panel Repository.
- Distribute as a Python package PyPi, Conda or similar.
- Share as a blog post on Medium or similar
- Share as a repository on GitHub or similar.
- Share on social media like Twitter, LinkedIn or similar.
- Contribute it to the Gallery at awesome-panel.org or the awesome-panel-extensions package.

You contributions matter. Thanks.

Distributing Your Extension on PyPi

Sharing one or more extensions as a package on PyPi requires packaging your Python project as you would do for any other Python project.

The Packaging Python Projects guide describes this.

You can also study the Awesome Panel Extensions Repository to see how a specific Panel Extensions Package is set up. You can find the `awesome-panel-extensions` package on PyPi here.

If your extension contains Bokeh extensions, you have to make sure your bokeh `bokeh.ext.json` and your build `dist`files are shipped with your package.

- In setup.py you need to set `include_package_data=True` to enable the use of a `Manifest.in` file.
- Your Manifest.in file then needs to include something like

```
include awesome_panel_extensions/*.json
include awesome_panel_extensions/index.ts
include awesome_panel_extensions/bokeh_extensions/*.ts
graft awesome_panel_extensions/dist
```

Contributing Your Extension to Panel

It's as easy as suggesting it as a Feature Request or providing it as a Pull request on the Panel Github site.

For more information on getting started as **Panel Developer** see the Panel Developer Guide.

6.1.8 Other

Resources

Awesome Extensions for Panel

COMING UP

Awesome Extensions for Other Frameworks

- Streamlit Component Gallery
- Streamlit Embed Code
- Jupyter/ IpyWidgets/ Voila - TBD
- Dash - TBD

Ideas for Extensions

The below is a list of Awesome Extensions I could come up with that I have currently (20200718) not seen examples of.

Feel free to use them as inspiration for a learning or contributing to the community.

Feel free to implement them in any of the awesome Python Frameworks (Bokeh, Dash, Panel, Streamlit or Voila). If they are implemented in one framework parts of the work can be reused across the frameworks.

Python in the Browser - BrythonComponent

Wouldn't it be awesome if you could use Python in your browser instead of on the server only? Well it might be possible with Brython.

I would like to be able to write something like

```
BrythonComponent(python_code_string)
```

and see something like

Brython Tutorial Calculator

or

Brython Snake Game

powered by Python running in the Browser.

Maybe the extension can also support bidirectional communication?

I hope this could help you and the Python community create awesome things. I also hope it could help to get Python working in the browser in general.

MORE IDEAS COMING UP. FEEL FREE TO SHARE YOURS.

Python Scientific Stack in the Browser - PyodideComponent

Wouldn't it be awesome if you could use the Python Scientific Stack in the Browser? Well maybe you can with Pyodide.

I would like to be able to write something like

```
PyodideComponent(python_code_string)
```

and see something like

Pyodide Random Walk

powered by the Python Scientific Stack running in the browser.

Maybe the extension can also support bidirectional communication?

I hope this could help you and the Python community create awesome things. I also hope it could help to get Python working in the browser in general.

Tips & Tricks

Start With a Working Example and Iterate

Developing extensions and Bokeh extensions in particular can be a bit tricky until you get familiar with it. You might get error messages that you don't understand or know how to solve. For me the best way to start a new extension is to

- Copy a simple example into your project.
    1. For Bokeh extensions the HTMLButton Extension is a good, simple example to start with.
- Test that it works via `panel serve` or similar and solve any problems that you might find.
- Stage (`git add`) the changes when the example works.

Then you do very small iterations of develop-test-stage. For example

- Rename folder. Test. Stage.
- Rename files. Test. Stage.
- Rename class (and similar) names in the files. Test. Stage.
- Add incremental functionality. Test. Stage.

Everytime you need to add incremental functionality, you can find the inspiration by studying the documentation or a similar example.

Use Your Extension Across Frameworks

Wouldn't it be cool if your awesome panel extension could be used in another framework like Streamlit, Bokeh, Voila or Dash?

This is actually becoming more and more of a possibility.

The figure below provides an overview of how components currently can be used across frameworks.

To be determined:

- How to convert Plotly Dash? jupyter-plotly-dash?

TBD

Roadmap

- How to Test
- How to Debug
- How to use VS Code efficiently to develop extensions
- How to use frameworks like React, Vue and maybe Angular
- Tips & Tricks
- FAQ
- Convert examples to notebooks.
- Integrate with official Panel site
  - For example as example Notebooks in the Gallery?

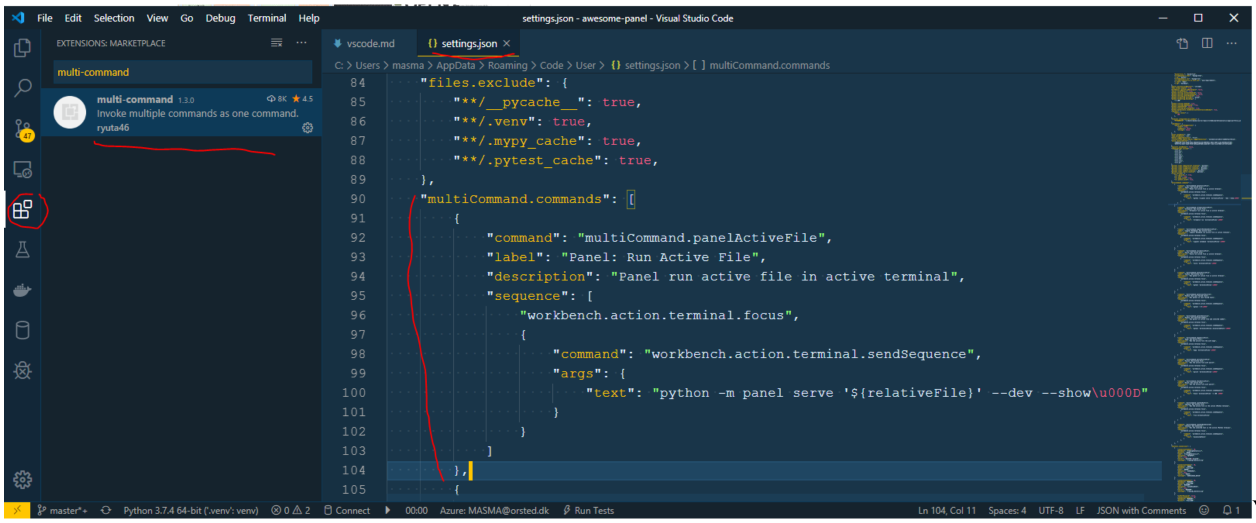
CHAPTER 7

How to use Panel with VS Code

7.1 Running Your Panel App

You can use the multi-command extension to configure a keyboard short cut to execute `panel serve <relativeFile.py>` with livereload.

You start by installing the multi-command extension and adding the configuration shown to your settings.json file.



```
    "files.exclude": {
        "**/__pycache__": true,
        "**/.venv": true,
        "**/.mypy_cache": true,
        "**/.pytest_cache": true,
    },
    "multiCommand.commands": [
        {
            "command": "multiCommand.panelActiveFile",
            "label": "Panel: Run Active File",
            "description": "Panel run active file in active terminal",
            "sequence": [
                "workbench.action.terminal.focus",
                {
                    "command": "workbench.action.terminal.sendSequence",
                    "args": {
                        "text": "python -m panel serve '${relativeFile}' --dev --show\u000D"
                    }
                }
            ]
        },
        {
```

VS Code multi-command

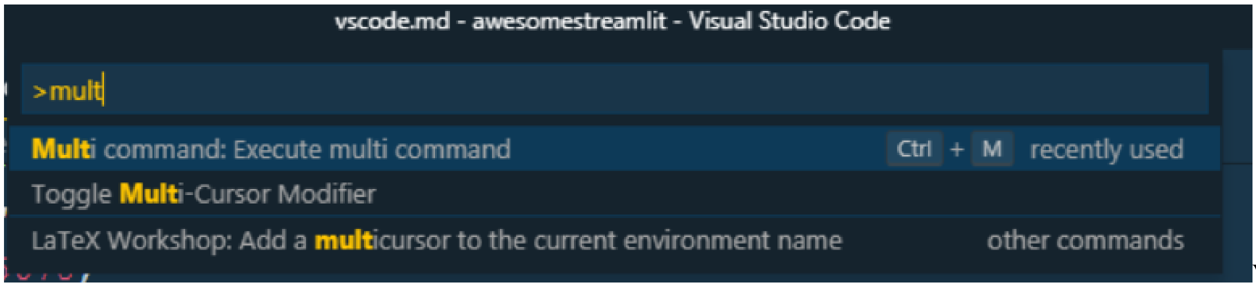
```
{
    "command": "multiCommand.panelActiveFile",
    "label": "Panel: Run Active File",
    "description": "Panel run active file in active terminal",
    "sequence": [
        "workbench.action.terminal.focus",
        {
```

(continues on next page)

(continued from previous page)

```
            "command": "workbench.action.terminal.sendSequence",
            "args": {
                "text": "python -m panel serve '${relativeFile}' --dev --show\u000D"
            }
        }
    ]
},
```

Then you can execute your *panel serve* command via the command palette (CTRL+SHIFT+P)



vscode.md - awesomestreamlit - Visual Studio Code

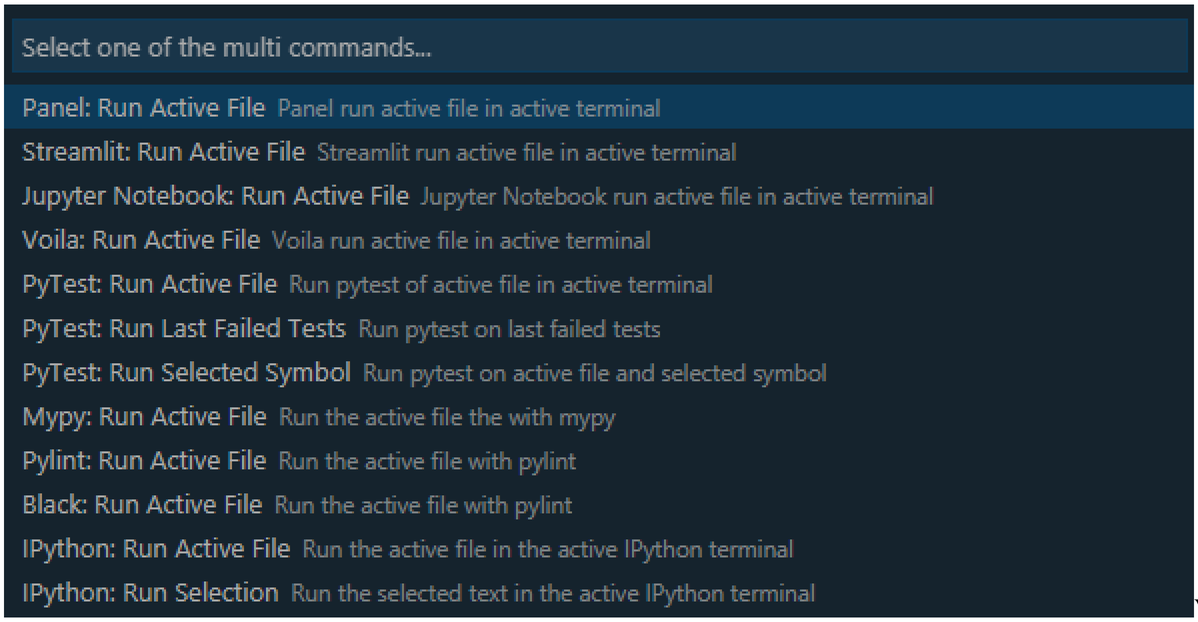
>mult

Multi command: Execute multi command Ctrl + M recently used

Toggle Multi-Cursor Modifier

LaTeX Workshop: Add a multicursor to the current environment name other commands

VS Code multi-command execute



Select one of the multi commands...

Panel: Run Active File Panel run active file in active terminal

Streamlit: Run Active File Streamlit run active file in active terminal

Jupyter Notebook: Run Active File Jupyter Notebook run active file in active terminal

Voila: Run Active File Voila run active file in active terminal

PyTest: Run Active File Run pytest of active file in active terminal

PyTest: Run Last Failed Tests Run pytest on last failed tests

PyTest: Run Selected Symbol Run pytest on active file and selected symbol

Mypy: Run Active File Run the active file the with mypy

Pylint: Run Active File Run the active file with pylint

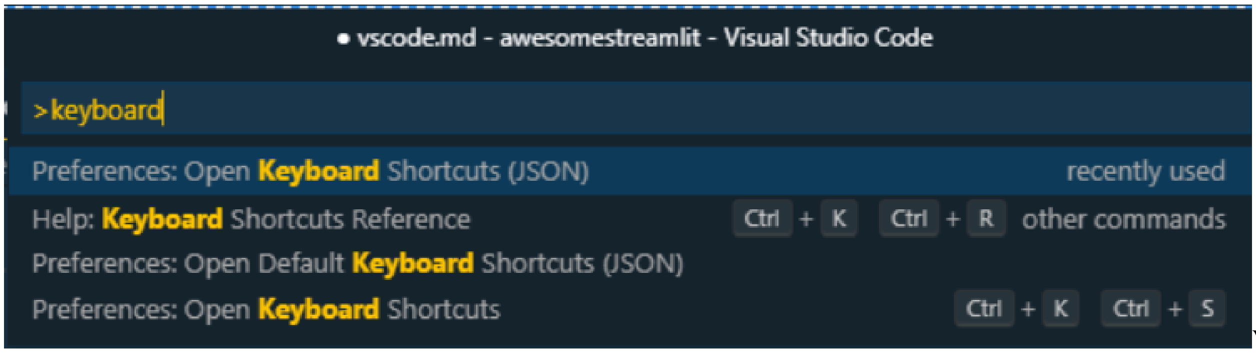
Black: Run Active File Run the active file with pylint

IPython: Run Active File Run the active file in the active IPython terminal

IPython: Run Selection Run the selected text in the active IPython terminal

VS Code multi-command Panel serve

Or you can setup a keyboard shortcut in your keybindings.json file to serve Panel



• vscode.md - awesomestreamlit - Visual Studio Code

>keyboard

Preferences: Open Keyboard Shortcuts (JSON) recently used

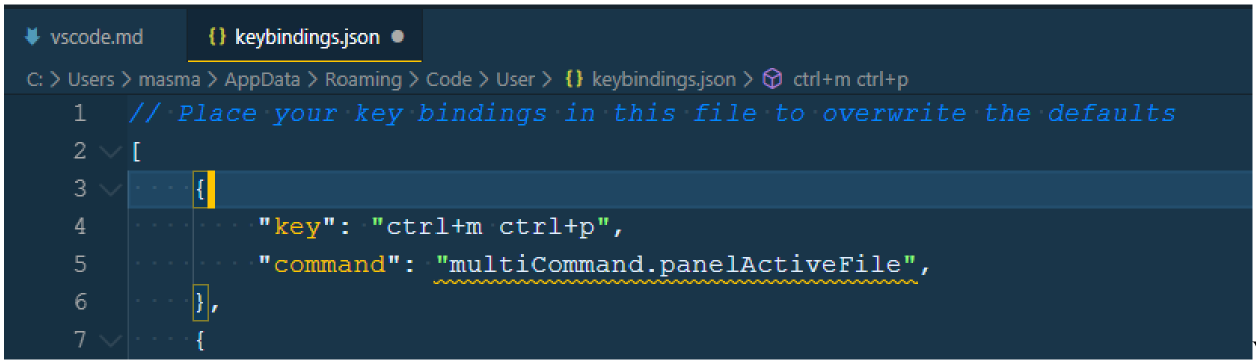
Help: Keyboard Shortcuts Reference Ctrl + K Ctrl + R other commands

Preferences: Open Default Keyboard Shortcuts (JSON)

Preferences: Open Keyboard Shortcuts Ctrl + K Ctrl + S

VS

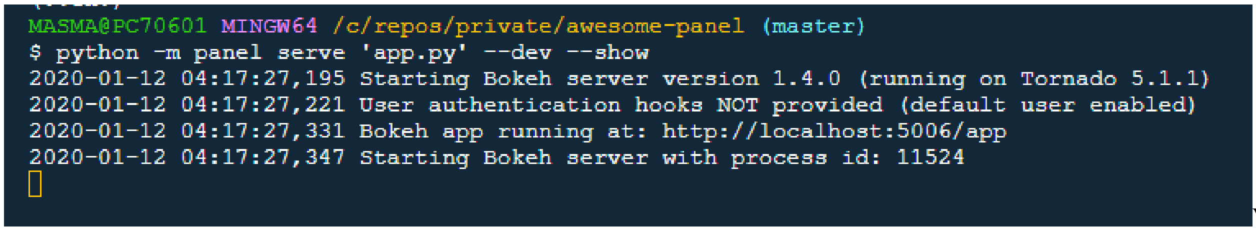
Code open keyboard settings


vscode.md
keybindings.json
C: > Users > masma > AppData > Roaming > Code > User > keybindings.json > ctrl+m ctrl+p

```
// Place your key bindings in this file to overwrite the defaults
[
    {
        "key": "ctrl+m ctrl+p",
        "command": "multiCommand.panelActiveFile",
    },
    {
```

VS Code keybindings

```
{
    "key": "ctrl+m ctrl+p",
    "command": "multiCommand.panelActiveFile",
},
```



```
MASMA@PC70601 MINGW64 /c/repos/private/awesome-panel (master)
$ python -m panel serve 'app.py' --dev --show
2020-01-12 04:17:27,195 Starting Bokeh server version 1.4.0 (running on Tornado 5.1.1)
2020-01-12 04:17:27,221 User authentication hooks NOT provided (default user enabled)
2020-01-12 04:17:27,331 Bokeh app running at: http://localhost:5006/app
2020-01-12 04:17:27,347 Starting Bokeh server with process id: 11524
```

VS Code terminal

Please **note** that I run

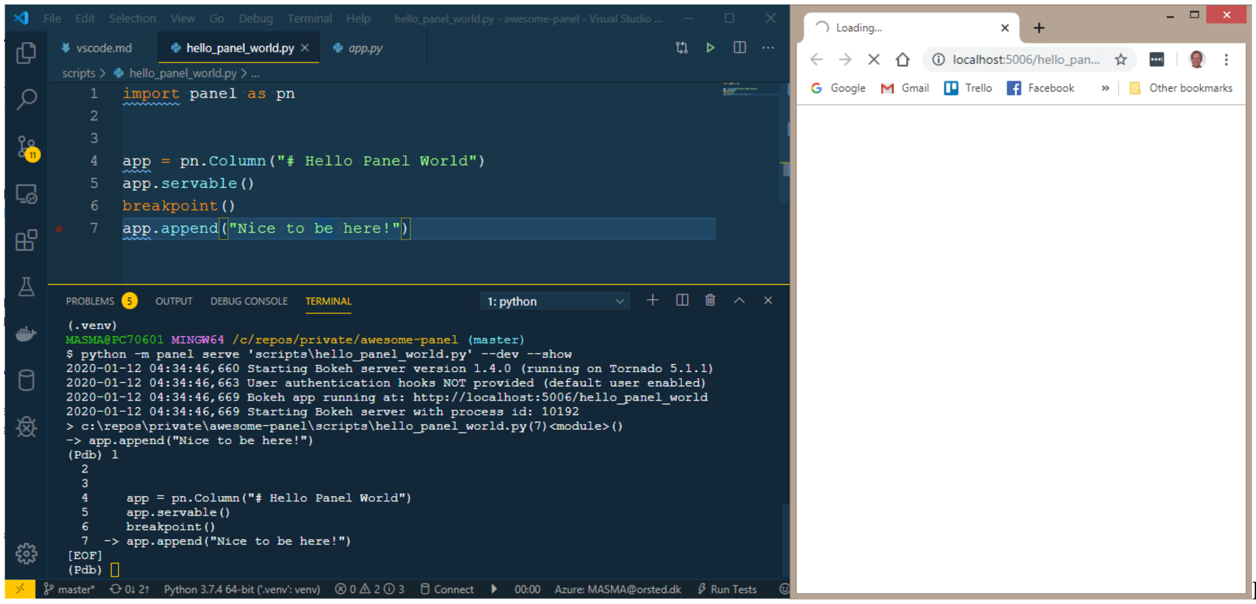
- `panel serve` as a module via `python -m` because its needed on Windows to make the livereload work via the `--dev` option. See Panel Issue 790.
- with the `--show` option to open a new tab in my browser after the server reload. It saves me a click to *Reload* the page.
- I have opened Bokeh Issue 9501 to suggest a faster reload of the server and an automatic refresh of the browser tab. Please upvote if you like it.

7.2 Debugging

7.2.1 Manual Debugging

You can **debug mannually** by inserting a `breakpoint()` (Python 3.7+) or `import pdb;pdb.set_trace()` (Python 3.6 or below) in your code.

Please **note** that I often experience that my terminal freezes during debugging of Panel apps. So inserting a `breakpoint()` manually is not something I often do. As an alternative I might use a `print` or `logging.debug` statement for debugging.



```python
import panel as pn


app = pn.Column("# Hello Panel World")
app.servable()
breakpoint()
app.append("Nice to be here!")
```

```
(.venv)
MASMA@PC70601 MINGW64 /c/repos/private/awesome-panel (master)
$ python -m panel serve 'scripts\hello_panel_world.py' --dev --show
2020-01-12 04:34:46,660 Starting Bokeh server version 1.4.0 (running on Tornado 5.1.1)
2020-01-12 04:34:46,663 User authentication hooks NOT provided (default user enabled)
2020-01-12 04:34:46,669 Bokeh app running at: http://localhost:5006/hello_panel_world
2020-01-12 04:34:46,669 Starting Bokeh server with process id: 10192
> c:\repos\private\awesome-panel\scripts\hello_panel_world.py(7)<module>()
-> app.append("Nice to be here!")
(Pdb) l
  2
  3
  4     app = pn.Column("# Hello Panel World")
  5     app.servable()
  6     breakpoint()
  7  -> app.append("Nice to be here!")
[EOF]
(Pdb)
```

File Edit Selection View Go Debug Terminal Help
hello_panel_world.py - awesome-panel - Visual Studio ...
vscode.md
hello_panel_world.py
app.py
scripts > hello_panel_world.py > ...
PROBLEMS 5
OUTPUT
DEBUG CONSOLE
TERMINAL
1: python
master* 0↓ 2↑ Python 3.7.4 64-bit ('.venv': venv) ⊗ 0 ⚠ 2 ⓘ 3 Connect 00:00 Azure: MASMA@orsted.dk Run Tests
Loading...
localhost:5006/hello_pan...
Google Gmail Trello Facebook Other bookmarks

Debugging via breakpoint

7.2.2 Integrated Debugging

You can also use the **integrated debugger** in VS Code via the ptvsd Python package

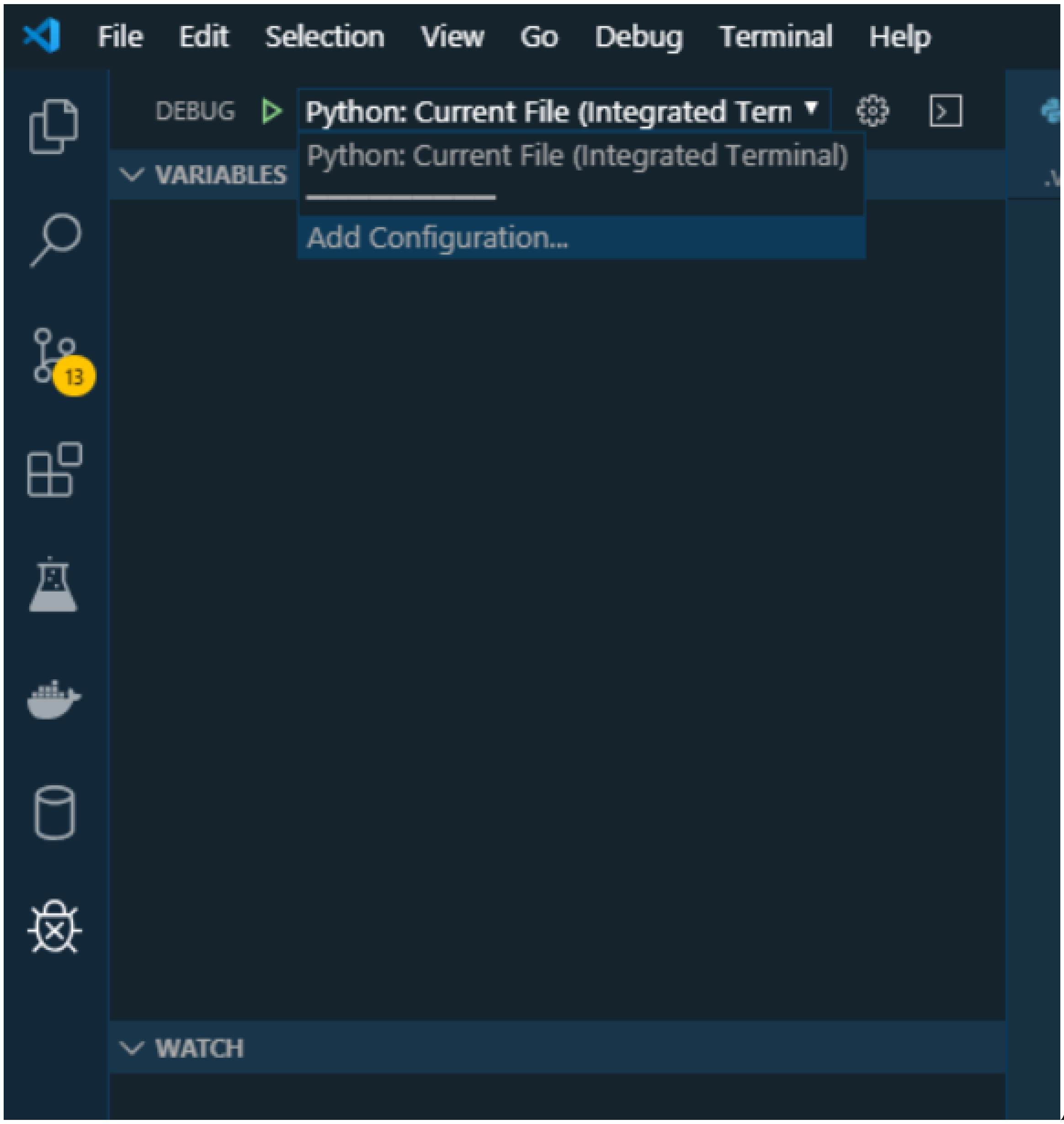
Debugging with ptvsd on Windows with python 3.7.4 is working really well but there are reports in the Streamlit community that running ptvsd on ubuntu 18.04.3 LTS with Python 3.6.8 does not work. See Streamlit Issue 648.

First you should `pip install ptvsd`.

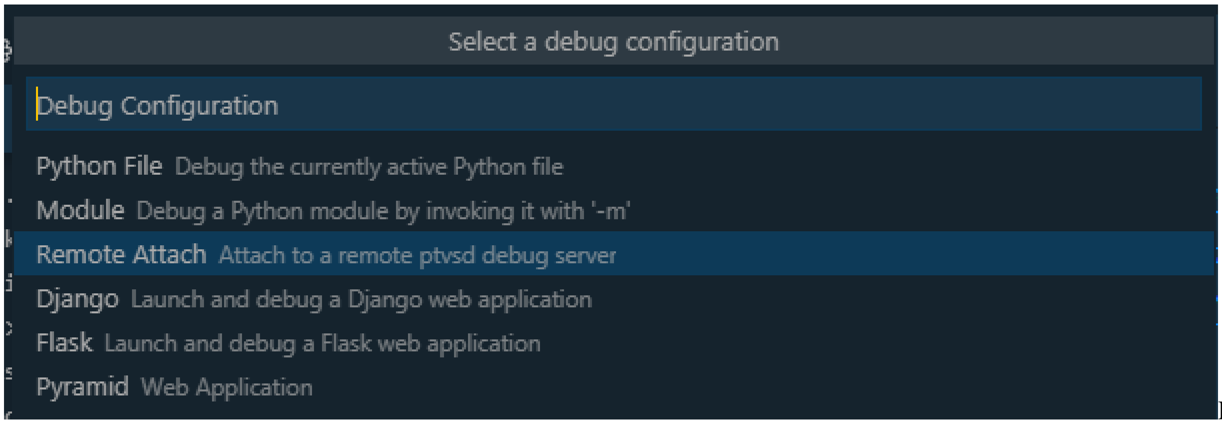
Then you need to insert the following snippet in your `<your-app_name>.py` file.

```python
import ptvsd
ptvsd.enable_attach(address=('localhost', 5678))
ptvsd.wait_for_attach() # Only include this line if you always want to attach the␣
↪debugger
```

Then you should configure your *Remote Attach: debug PTVSD option*



Add Configuration


Select a debug configuration

Debug Configuration

Python File Debug the currently active Python file
Module Debug a Python module by invoking it with '-m'
Remote Attach Attach to a remote ptvsd debug server
Django Launch and debug a Django web application
Flask Launch and debug a Flask web application
Pyramid Web Application

Debug Configuration

and update to the below in your launch.json file. Please make sure that you manually insert the *redirectOutput* setting below.

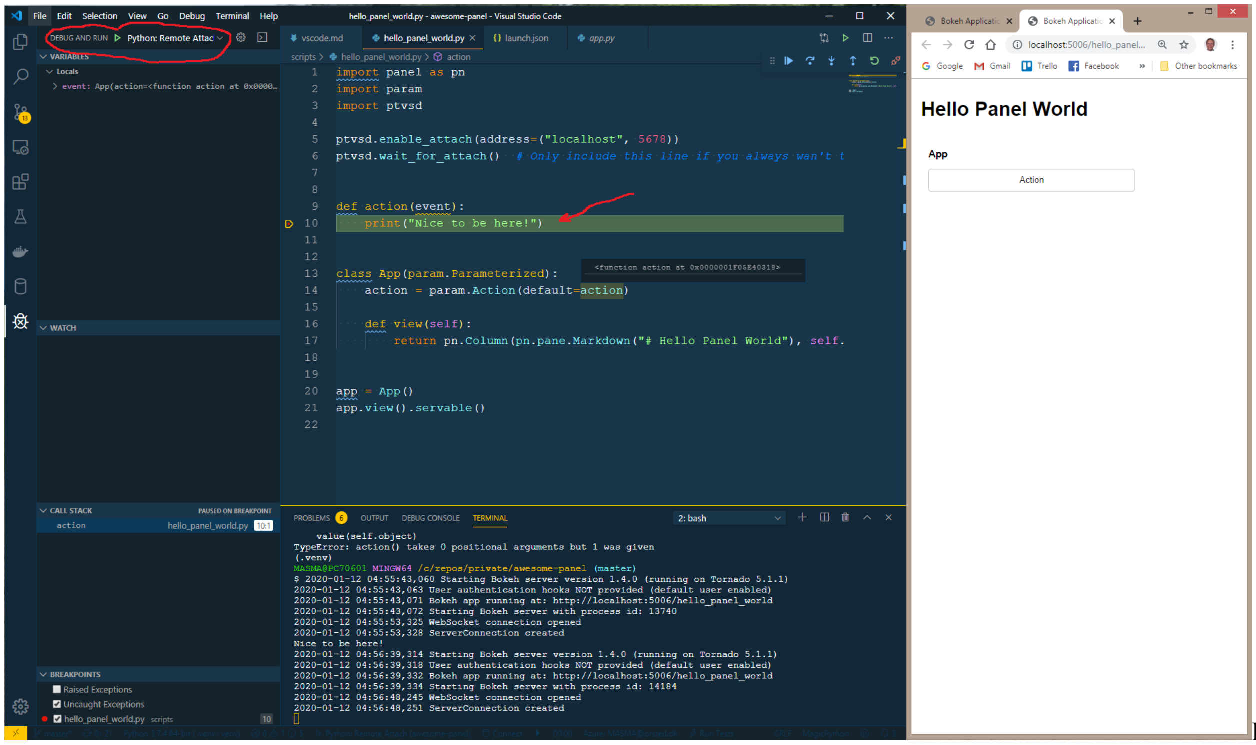
```
{
    "name": "Python: Remote Attach",
    "type": "python",
    "request": "attach",
    "port": 5678,
    "host": "localhost",
    "justMyCode": true,
    "redirectOutput": true,
    "pathMappings": [
        {
            "localRoot": "${workspaceFolder}",
            "remoteRoot": "."
        }
    ]
}
```

Please note that by default you will be debugging your own code only. If you want to debug into for example the panel code, then you can change the `justMyCode` setting from `true` to `false`.

Then you can start your Panel app

```
panel serve <your-app_name>.py
```

Finally you can attach the debugger by clicking the debugger play button



```python
import panel as pn
import param
import ptvsd

ptvsd.enable_attach(address=("localhost", 5678))
ptvsd.wait_for_attach()  # Only include this line if you always wan't t


def action(event):
    print("Nice to be here!")


class App(param.Parameterized):
    action = param.Action(default=action)

    def view(self):
        return pn.Column(pn.pane.Markdown("# Hello Panel World"), self.


app = App()
app.view().servable()
```

```
    value(self.object)
TypeError: action() takes 0 positional arguments but 1 was given
(.venv)
MASMA@PC70601 MINGW64 /c/repos/private/awesome-panel (master)
$ 2020-01-12 04:55:43,060 Starting Bokeh server version 1.4.0 (running on Tornado 5.1.1)
2020-01-12 04:55:43,063 User authentication hooks NOT provided (default user enabled)
2020-01-12 04:55:43,071 Bokeh app running at: http://localhost:5006/hello_panel_world
2020-01-12 04:55:43,072 Starting Bokeh server with process id: 13740
2020-01-12 04:55:53,325 WebSocket connection opened
2020-01-12 04:55:53,328 ServerConnection created
Nice to be here!
2020-01-12 04:56:39,314 Starting Bokeh server version 1.4.0 (running on Tornado 5.1.1)
2020-01-12 04:56:39,318 User authentication hooks NOT provided (default user enabled)
2020-01-12 04:56:39,332 Bokeh app running at: http://localhost:5006/hello_panel_world
2020-01-12 04:56:39,334 Starting Bokeh server with process id: 14184
2020-01-12 04:56:48,245 WebSocket connection opened
2020-01-12 04:56:48,251 ServerConnection created
```

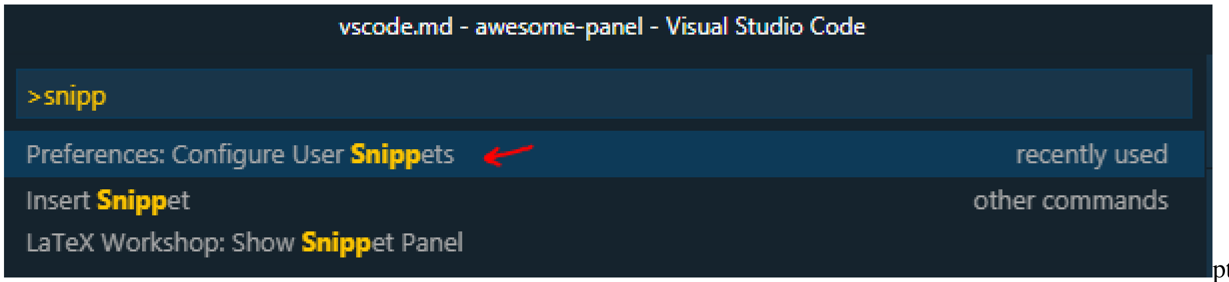
Python remote attach

and you can debug away.

Please **note**, that you need to re-attach the debugger when the server reloads.

Using a ptvsd snippet

You can create a snippet in your `python.json` snippet configuration to insert the `import ptvsd...` code.



vscode.md - awesome-panel - Visual Studio Code

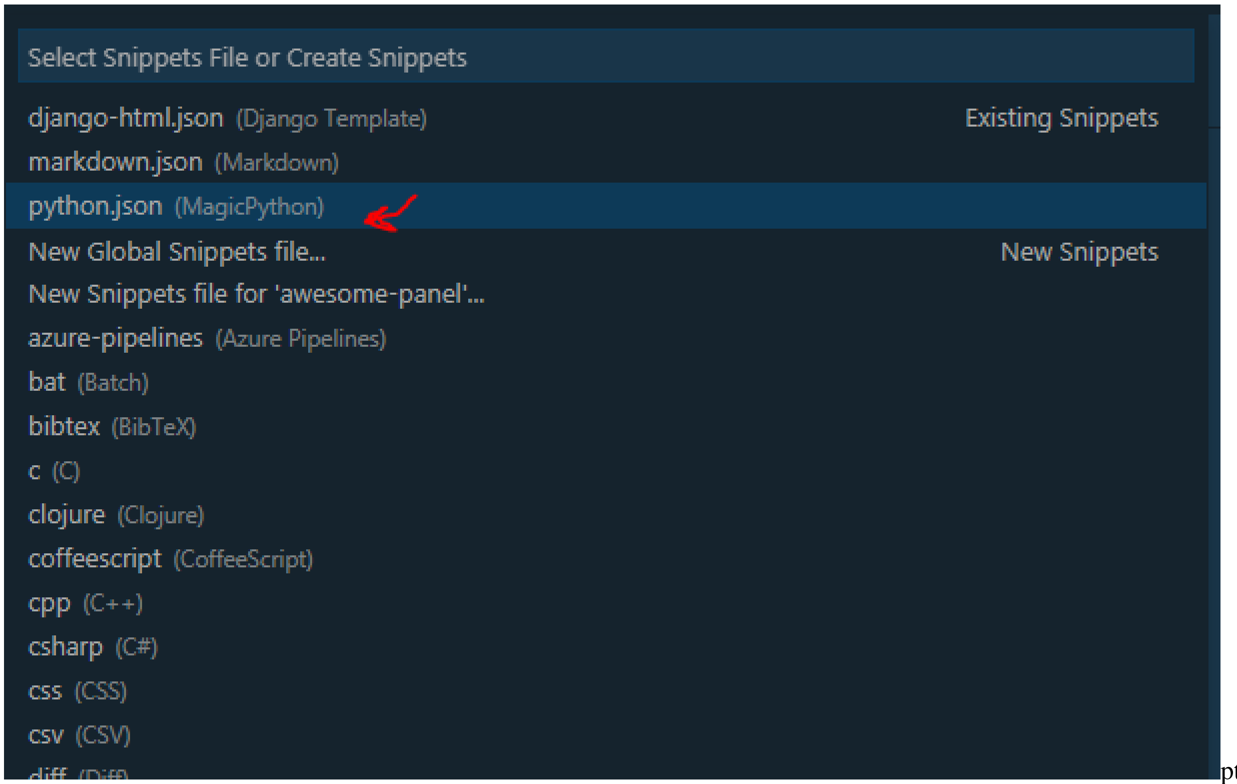
>snipp

Preferences: Configure User Snippets — recently used

Insert Snippet — other commands

LaTeX Workshop: Show Snippet Panel

ptvsd code snippet



Select Snippets File or Create Snippets

django-html.json (Django Template) — Existing Snippets

markdown.json (Markdown)

python.json (MagicPython)

New Global Snippets file... — New Snippets

New Snippets file for 'awesome-panel'...

azure-pipelines (Azure Pipelines)

bat (Batch)

bibtex (BibTeX)

c (C)

clojure (Clojure)

coffeescript (CoffeeScript)

cpp (C++)

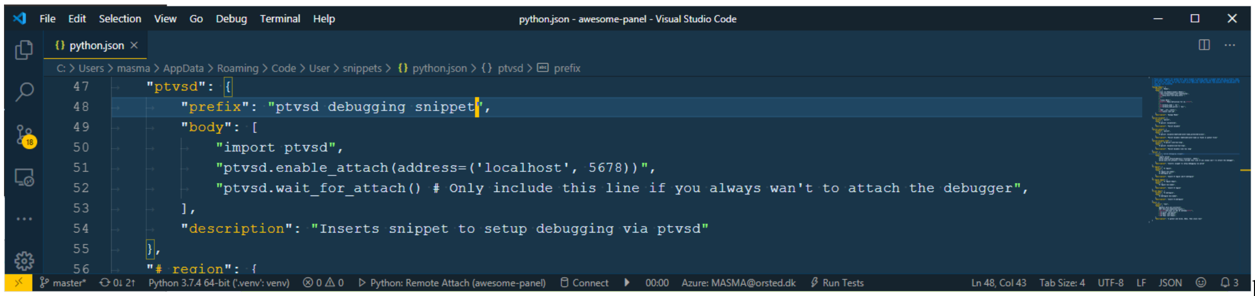
csharp (C#)

css (CSS)

csv (CSV)

diff (Diff)

ptvsd code snippet



python.json - awesome-panel - Visual Studio Code

C: > Users > masma > AppData > Roaming > Code > User > snippets > python.json > ptvsd > prefix

```
    "ptvsd": {
        "prefix": "ptvsd debugging snippet",
        "body": [
            "import ptvsd",
            "ptvsd.enable_attach(address=('localhost', 5678))",
            "ptvsd.wait_for_attach() # Only include this line if you always wan't to attach the debugger",
        ],
        "description": "Inserts snippet to setup debugging via ptvsd"
    },
    "# region": {
```

master* 0↓ 2↑ Python 3.7.4 64-bit ('.venv': venv) ⊗ 0 ⚠ 0 Python: Remote Attach (awesome-panel) Connect 00:00 Azure: MASMA@orsted.dk Run Tests Ln 48, Col 43 Tab Size: 4 UTF-8 LF JSON 3

ptvsd code snippet

```
"ptvsd": {
    "prefix": "ptvsd debugging snippet",
```

(continues on next page)

(continued from previous page)

```
    "body": [
        "import ptvsd",
        "ptvsd.enable_attach(address=('localhost', 5678))",
        "print('Ready to attach the VS Code debugger')",
        "ptvsd.wait_for_attach() # Only include this line if you always want to␣
↪attach the debugger",
    ],
    "description": "Inserts snippet to setup debugging via ptvsd"
},
```

Using a dedicated app_debug_vscode.py file for debugging

Adding and removing the *ptvsd* code above can be cumbersome. So a usefull trick is to setup a dedicated *app_debug_vscode.py* file for debugging.

Assuming your app.py file has a `def main():` function, then your *app_debug_vscode.py* file could look as follows

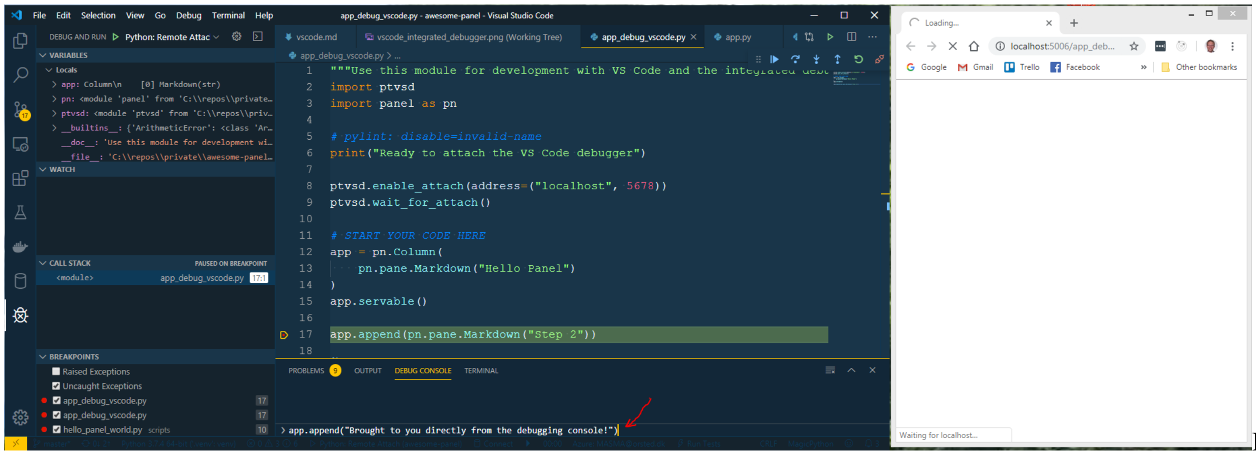
```
"""Use this module for development with VS Code and the integrated debugger"""
import ptvsd
import panel as pn

print("Ready to attach the VS Code debugger")

ptvsd.enable_attach(address=("localhost", 5678))
ptvsd.wait_for_attach()

# START YOUR CODE HERE
import app

app.main().servable()
```

then you run `panel serve app_debug_vscode.py` instead of `panel serve app.py` and attach the debugger.

For a use case see my app.py and app_dev_vscode.py files.

Using the integrated Debugging Console

When you are running your integrated debugging in VS Code, you can use the *Debugging Console* with Panel. Then you can write dataframes and charts to the browser window and take a better look at your data, than you can in VS Code. You can also add or remove a Viewable to and from any layout.

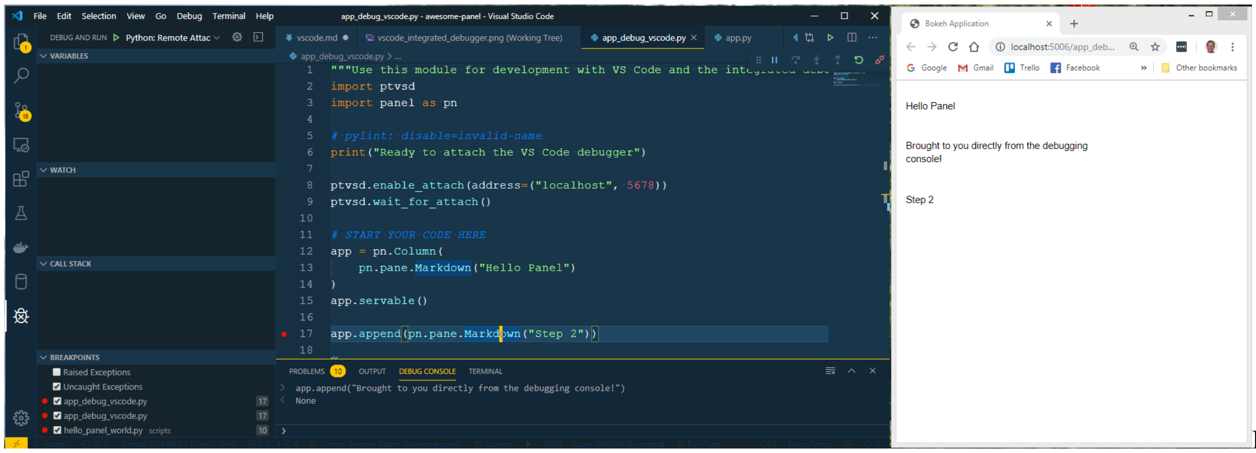


```
"""Use this module for development with VS Code and the integrated debu
import ptvsd
import panel as pn

# pylint: disable=invalid-name
print("Ready to attach the VS Code debugger")

ptvsd.enable_attach(address=("localhost", 5678))
ptvsd.wait_for_attach()

# START YOUR CODE HERE
app = pn.Column(
    pn.pane.Markdown("Hello Panel")
)
app.servable()

app.append(pn.pane.Markdown("Step 2"))
```

```
> app.append("Brought to you directly from the debugging console!")
```

```
VARIABLES
  Locals
  > app: Column\n    [0] Markdown(str)
  > pn: <module 'panel' from 'C:\\repos\\private…
  > ptvsd: <module 'ptvsd' from 'C:\\repos\\priv…
  > __builtins__: {'ArithmeticError': <class 'Ar…
    __doc__: 'Use this module for development wi…
    __file__: 'C:\\repos\\private\\awesome-panel…
CALL STACK                         PAUSED ON BREAKPOINT
  <module>               app_debug_vscode.py 17:1
BREAKPOINTS
  Raised Exceptions
  Uncaught Exceptions
  app_debug_vscode.py                          17
  app_debug_vscode.py                          17
  hello_panel_world.py scripts                 10
```

Import Panel



```
"""Use this module for development with VS Code and the integrated debu
import ptvsd
import panel as pn

# pylint: disable=invalid-name
print("Ready to attach the VS Code debugger")

ptvsd.enable_attach(address=("localhost", 5678))
ptvsd.wait_for_attach()

# START YOUR CODE HERE
app = pn.Column(
    pn.pane.Markdown("Hello Panel")
)
app.servable()

app.append(pn.pane.Markdown("Step 2"))
```

```
> app.append("Brought to you directly from the debugging console!")
< None
>
```

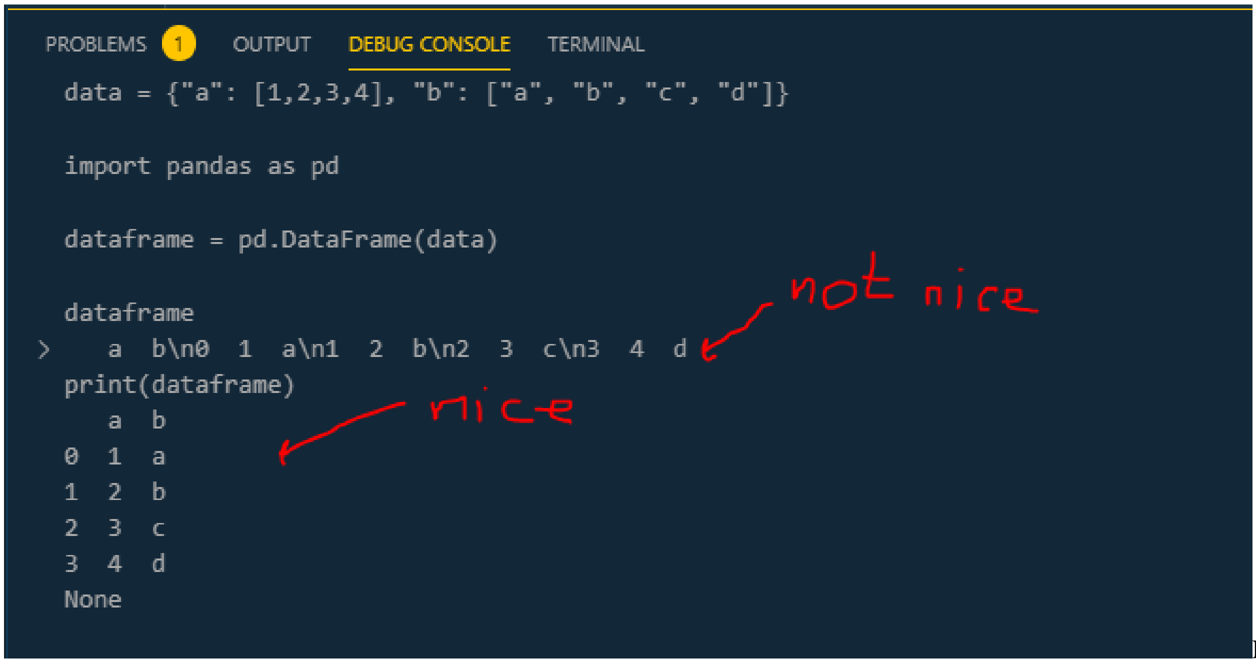
Hello Panel

Brought to you directly from the debugging console!

Step 2

Import Panel

You should also remember to *print* your dataframes to the debugger console to get a useable formatting.



```
data = {"a": [1,2,3,4], "b": ["a", "b", "c", "d"]}

import pandas as pd

dataframe = pd.DataFrame(data)

dataframe
>    a  b\n0  1  a\n1  2  b\n2  3  c\n3  4  d
print(dataframe)
   a  b
0  1  a
1  2  b
2  3  c
3  4  d
None
```

Nice

Print of DataFrame

7.2.3 Increasing the log level

You can add `--log-level=debug` to your `panel serve` command to increase the level of logging from panel it self.

7.2.4 Additional Resources

The Debugging section been developed on top of many great sources

- Bokeh - VS Code Debugging
- VS Code Debugging
- PyCharm Debugging

CHAPTER 8

Tips and Tricks + Best Practices for creating performant panel apps

Slow Panel apps can have different causes

- The Bokeh Layout Engine can slow your application down. See Issue 9515.

My tips+tricks to mitigate this is.

- Use the Template system whenever you can.
- Don't do lots of nested Columns and Rows in Panel. Use pn.Param if you have a model with a lot of parameters.
- Don't configure layout settings like width, height, margin etc. of Panel Columns and Rows via Css. Use Column and Row attributes in Panel for that.
- Use FireFox instead of Chrome. It's much faster.
- Swap out all content of a layout at once (`col[:]=[objects]`) instead of multiple times (`for obj in objects:  col.append(obj)`)

CHAPTER 9

Prebuilt Bokeh Extensions

In this document I will describe how I got **prebuilt bokeh model extensions** setup as a part of the awesome-panel package. I needed it temporarily while waiting for the `WebComponent` PR to be reviewed and released by Panel.

Setting up prebuilt extensions using `Bokeh init --interactive` is briefly described in the Bokeh Docs. See Bokeh Pre-built extensions.

I hope this description can help others who would like to create prebuilt custom bokeh models for Bokeh or Panel.

9.1 Steps

I navigated to the root of the awesome-panel package

```
cd awesome-panel/package
```

ran `bokeh init --interactive`

```
$ bokeh init --interactive
Working directory: C:\repos\private\awesome-panel\package\awesome_panel
Wrote C:\repos\private\awesome-panel\package\awesome_panel\bokeh.ext.json
Create package.json? This will allow you to specify external dependencies. [y/n] y
  What's the extension's name? [awesome_panel]
  What's the extension's version? [0.0.1]
  What's the extension's description? []
Wrote C:\repos\private\awesome-panel\package\awesome_panel\package.json
Create tsconfig.json? This will allow for customized configuration and improved IDE␣
↪experience. [y/n] y
Wrote C:\repos\private\awesome-panel\package\awesome_panel\tsconfig.json
Created empty index.ts. This is the entry point of your extension.
You can build your extension with bokeh build
All done.
```

In the `package.json` I had to replace

```
"dependencies": {
    "bokehjs": "^2.0.2"
  },
```

with

```
"dependencies": {
    "@bokeh/bokehjs": "^2.0.2"
  },
```

See bokeh init issue.

I also replaced the `tsconfig.json` contents with

```
{
  "compilerOptions": {
    "noImplicitAny": true,
    "noImplicitThis": true,
    "noImplicitReturns": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "strictNullChecks": true,
    "strictBindCallApply": false,
    "strictFunctionTypes": false,
    "strictPropertyInitialization": false,
    "alwaysStrict": true,
    "noErrorTruncation": true,
    "noEmitOnError": false,
    "declaration": true,
    "sourceMap": true,
    "importHelpers": false,
    "experimentalDecorators": true,
    "module": "esnext",
    "moduleResolution": "node",
    "esModuleInterop": true,
    "resolveJsonModule": true,
    "skipLibCheck": true,
    "target": "ES2017",
    "lib": ["es2017", "dom", "dom.iterable"],
    "baseUrl": ".",
    "outDir": "./dist/lib",
    "paths": {
      "@bokehjs/*": [
        "./node_modules/@bokeh/bokehjs/build/js/lib/*",
        "./node_modules/@bokeh/bokehjs/build/js/types/*"
      ]
    }
  },
  "include": ["./**/*.ts"]
}
```

At least including the `path` section is needed to be able to `import { div, label } from "@bokehjs/core/dom"` like @philippjfr does in Panel.

In the `index.ts` file I imported my models

```
import * as AwesomePanel from "./express/models/"
export {AwesomePanel}
```

(continues on next page)

(continued from previous page)

```
import {register_models} from "@bokehjs/base"
register_models(AwesomePanel as any)
```

In the `express/models/index.ts` file I exported the `WebComponent`.

```
export {WebComponent} from "./web_component"
```

Then I could `build` my extension

```
$ panel build
Working directory: C:\repos\private\awesome-panel\package\awesome_panel
Using C:\repos\private\awesome-panel\package\awesome_panel\tsconfig.json
Compiling TypeScript (3 files)
Linking modules
Output written to C:\repos\private\awesome-panel\package\awesome_panel\dist
All done.
```

The result is in the `dist` folder.

I discovered I did not even have to `serve` the `awesome_panel.js` file.

I could `panel serve` something like

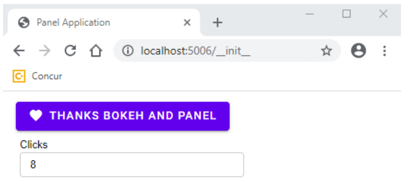
```
import param
import panel as pn
from awesome_panel.express.pane.web_component import WebComponent

MWC_ICONS = [
    None,
    "accessibility",
    "code",
    "favorite",
]  # For more icons see https://material.io/resources/icons/?style=baseline

MATERIAL = "https://cdn.jsdelivr.net/gh/marcskovmadsen/awesome-
↪panel@be59521090b7c9d9ba5eb16e936034e412e2c86b/assets/js/mwc.bundled.js"
pn.config.js_files["material"]=MATERIAL
font_pane = pn.pane.HTML(
    """
<link href="https://fonts.googleapis.com/css?family=Roboto:300,400,500" rel=
↪"stylesheet">
<link href="https://fonts.googleapis.com/css?family=Material+Icons&display=block" rel=
↪"stylesheet">
    """, width=0, height=0, margin=0,
)

class MWCButton(WebComponent):
    html = param.String("<mwc-button></mwc-button")
    attributes_to_watch = param.Dict({"label": "name", "icon": "icon", "raised":
↪"raised"})
    events_to_watch = param.Dict({"click": "clicks"})

    raised = param.Boolean(default=True)
    icon = param.ObjectSelector(default="favorite", objects=MWC_ICONS, allow_
↪None=True)
    clicks = param.Integer()
```

(continues on next page)

(continued from previous page)

```
    height = param.Integer(default=30)

button = MWCButton(name="Thanks Bokeh and Panel")

pn.Column(font_pane, button, pn.Param(button.param.clicks)).servable()
```

and get



Custom Bokeh Model

CHAPTER 10

Indices and tables

- genindex
- modindex
- search