Introducing Python as a Supported Language for Accelerator Controls at CERN

P. Elson, C. Baldi, I. Sinkarenko, CERN, Geneva, Switzerland

ICALEPCS 2021

Introduction

In 2019, Python was adopted as an officially supported language for interacting with CERN's accelerator controls. Highlighted here is some of the key infrastructure put in place to facilitate a user friendly, idiomatic and stable Python platform fit for operational high-level accelerator controls.

Acc-Py base distribution

- Lightweight, easy to install, used on local disk, containers and network drives
- Designed to be installed side-by-side with other versions of Acc-Py base
- Each invocation logged
- Not a pre-prepared fully-fledged distribution such as Anaconda or LCG

=> Designed to be extended by the user

Virtual environment

- A user can create an unlimited number of virtual environments
- No special privileges required
- Python interpreter and standard library linked from the base distribution

```
venv
├── bin
│   ├── python (symlink to base)
│   └── ...
├── lib
│   └── pythonX.Y
│       └── site-packages
│           └── <venv-packages>
├── include
├── share
└── pyvenv.cfg
```

pip install

- User can choose which packages to install
- Pre-configured to use the Acc-Py package index
- CERN-specific packages combined with those from the Python Package Index (PyPI)
- Tools built on top of pip (e.g. poetry) can be used

Python well-adapted for:

- Data processing and analysis
- Machine learning & deep learning
- System automation
- Rapid prototyping
- GUI development
- Mathematical modelling
- Hardware interfacing
- ...

Examples at CERN:

- Machine development (MD) studies
- Linac4 Source autopilot (WEPV018)
- Timing sequencer GUI (THPV015)
- Expert GUI applications
- Physics simulations
- Numerical optimisation for operations
- Rapid hardware interfacing
- ...

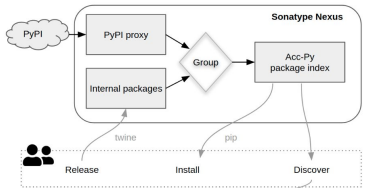
Python invocation logs

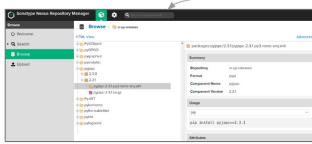
Each Acc-Py Python invocation is logged, along with data including the time, machine, username and a list of all packages installed in the environment.

This data has been invaluable in helping inform both technical and strategic decisions, and it provides a snapshot of usage of Python for accelerator controls.

In numbers, there are around 10,000 Python invocations per day and over 400 unique active users (September 2021).

Acc-Py package index





Motivation

- Access to PyPI from within technical network
- Ability to install internal and public packages using pip
- Filters to prevent unwanted or malicious packages being installed
- Archive of packages in case of PyPI outage or package removal

Development and deployment tools

Step 1: Create a project

```
$ acc-py init my-project
=> Writing setup.py
=> Writing README.md
=> Configuring my_project's directory structure
./my-project/
├── my_project/
│   ├── __init__.py
│   ├── tests/
│   │   └── __init__.py
├── README.md
└── setup.py
```

- Create a new Python package following a simple structure
- Tests are a first-class component of the project
- Possibility to generate configuration for API Documentation (sphinx) and Continuous integration (GitLab-CI)

Step 2: Freeze dependencies

```
$ acc-py app lock ./my-project/
./my-project/deployment/app
├── config.json
├── requirements-jar.json
└── requirements.txt
```

- Resolve floating dependency versions
- Including Java dependencies
- Guarantees that the dependencies in production are the same as those tested in development

Step 3: Deploy to central location

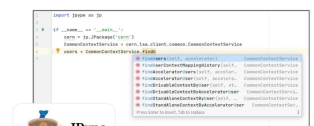
```
$ acc-py app deploy ./my-project/
=> Building wheel for my-project
=> Creating app in deployment location
=> my-project version 1.2.3 was deployed
/deployment/location/my-project
├── 1.2.3
│   ├── exec
│   ├── ...
│   └── venv
├── ...
├── app-config.yml
└── audit.log
```

- Consistent & repeatable deployment
- Based on a virtual environment
- Auditable history of application changes
- Common location for deployments

Step 4: Run in production

```
$ acc-py app run my-project
```

- Always runs in "isolated" mode
  => Consistent behaviour no matter the executing user or current directory
- Runs as a Python module, no additional scripting required



JPype

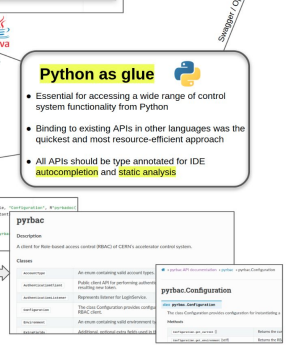
- Manages in-process JVM
- Zero-copy arrays between Python & Java
- Type annotations with `stubgenj`
- JARs automatically installed into the virtual environment

Python as glue

- Essential for accessing a wide range of control system functionality from Python
- Binding to existing APIs in other languages was the quickest and most resource-efficient approach
- All APIs should be type annotated for IDE autocompletion and static analysis

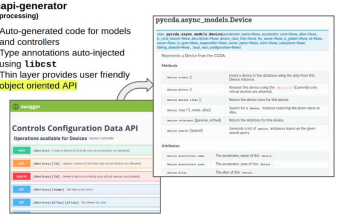
pybind11

- C++ based binding definition
- Easy to craft Pythonic bindings
- Creates shared-object module
- Released as pre-compiled `manylinux2014` Python wheels



openapi-generator
(+post-processing)

- Auto-generated code for models and controllers
- Type annotations auto-injected using `libcst`
- Thin layer provides user friendly object oriented API



Conclusion

There has been significant growth in the use of Python in the accelerator sector at CERN, with support for high-level accelerator controls having been introduced in 2019. To facilitate this, key infrastructure has been put in place, including development tools for a simple yet Pythonic experience, solutions for consistent application deployment and execution, and a suite of libraries for interacting with CERN's accelerator control system.

When considered together, the infrastructure presented here has resulted in a stable and effective platform from which Python users can efficiently build operational-quality libraries, applications and services for CERN's next accelerator run and beyond. Furthermore, such infrastructure is general in purpose, and would be a good fit for a diverse range of organisations wishing to adopt a robust and maintainable operational Python environment.

Introduction

In 2019, Python was adopted as an officially supported language for interacting with CERN's accelerator controls. Highlighted here is some of the key infrastructure put in place to facilitate a user friendly, idiomatic and stable Python platform fit for operational high-level accelerator controls.

Acc-Py base distribution

- Lightweight, easy to install, used on local disk, containers and network drives
- Designed to be installed side-by-side with other versions of Acc-Py base
- Each invocation logged
- Not a pre-prepared fully-fledged distribution such as Anaconda or LCG

=> Designed to be extended by the user

Virtual environment

- A user can create an unlimited number of virtual environments
- No special privileges required
- Python interpreter and standard library linked from the base distribution

```
venv
├── bin
│   ├── python (symlink to base)
│   └── …
├── lib
│   └── pythonX.Y
│       └── site-packages
│           └── <venv-packages>
├── include
├── share
└── pyvenv.cfg
```

pip install

- User can choose which packages to install
- Pre-configured to use the Acc-Py package index
- CERN-specific packages combined with those from the Python Package Index (PyPI)
- Tools built on top of pip (e.g. poetry) can be used

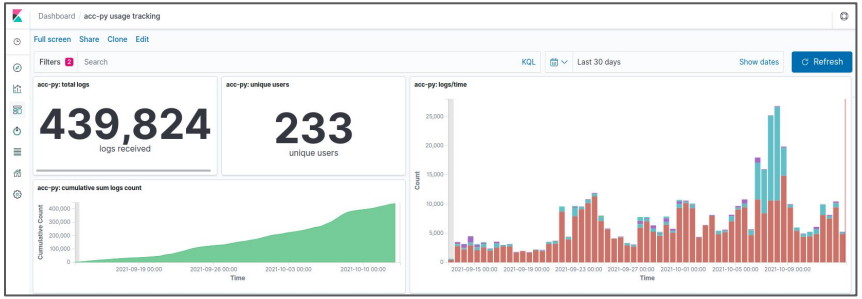
Python well-adapted for:

- Data processing and analysis
- Machine learning & deep learning
- System automation
- Rapid prototyping
- GUI development
- Mathematical modelling
- Hardware interfacing
- ...

Examples at CERN:

- Machine development (MD) studies
- Linac4 Source autopilot (WEPV018)
- Timing sequencer GUI (THPV015)
- Expert GUI applications
- Physics simulations
- Numerical optimisation for operations
- Rapid hardware interfacing
- ...

Python invocation logs

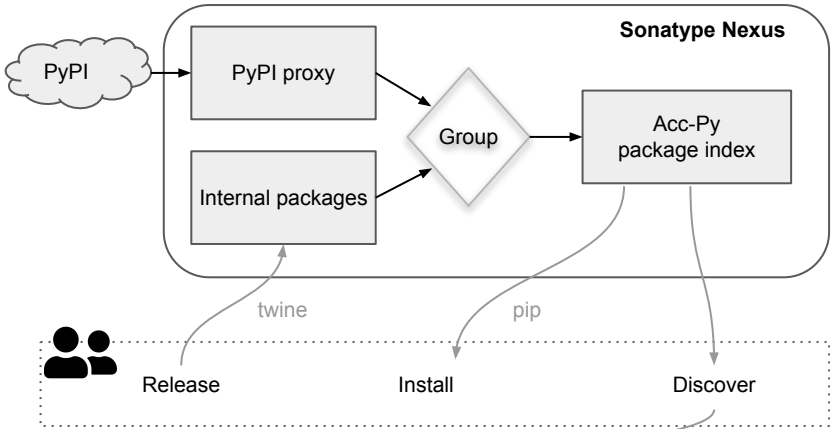


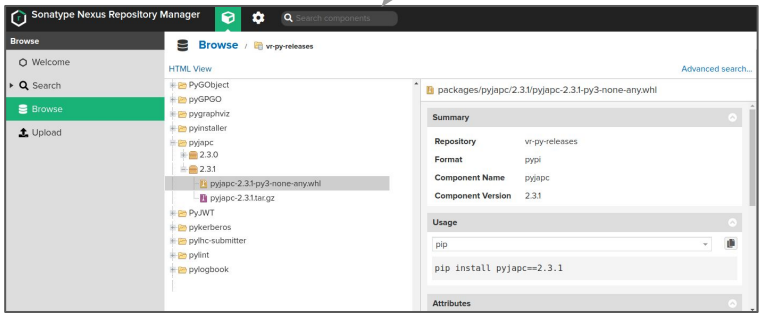
Each Acc-Py Python invocation is logged, along with data including the time, machine, username and a list of all packages installed in the environment.

This data has been invaluable in helping inform both technical and strategic decisions, and it provides a snapshot of usage of Python for accelerator controls.

In numbers, there are around 10,000 Python invocations per day and over 200 unique active users (September 2021).

Acc-Py package index





Motivation

- Access to PyPI from within technical network
- Ability to install internal and public packages using pip
- Filters to prevent unwanted or malicious packages being installed
- Archive of packages in case of PyPI outage or package removal

Development and deployment tools

Step 1: Create a project

```
$ acc-py init my-project
=> Writing setup.py
=> Writing README.md
=> Configuring my_project's directory structure
```

```
./my-project/
├── my_project/
│   ├── __init__.py
│   └── tests/
│       └── __init__.py
├── README.md
└── setup.py
```

- Create a new Python package following a simple structure
- Tests are a first-class component of the project
- Possibility to generate configuration for API Documentation (sphinx) and Continuous integration (GitLab-CI)

Step 2: Freeze dependencies

```
$ acc-py app lock ./my-project/
```

```
./my-project/deployment/app
├── config.json
├── requirements-jar.json
└── requirements.txt
```

- Resolve floating dependency versions
- Including Java dependencies
- Guarantees that the dependencies in production are the same as those tested in development

Step 3: Deploy to central location

```
$ acc-py app deploy ./my-project/
=> Building wheel for my-project
=> Creating app in deployment location
=> my-project version 1.2.3 was deployed
```

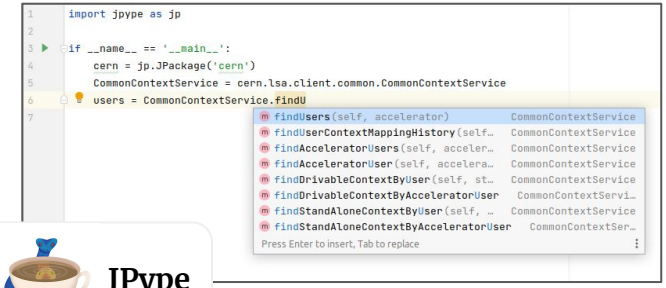
```
/deployment/location/my-project
├── 1.2.3
│   ├── exec
│   │   ├── …
│   └── venv
│   ├── …
├── app-config.yml
└── audit.log
```

- Consistent & repeatable deployment
- Based on a virtual environment
- Auditable history of application changes
- Common location for deployments

Step 4: Run in production

```
$ acc-py app run my-project
```

- Always runs in “isolated” mode
  => Consistent behaviour no matter the executing user or current directory
- Runs as a Python module, no additional scripting required



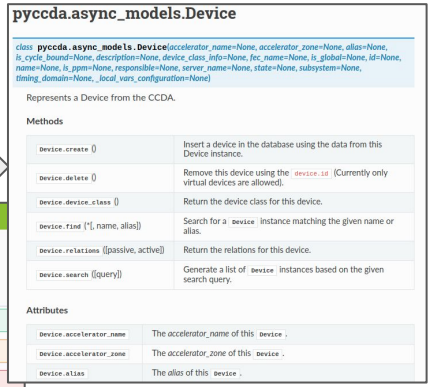
JPype

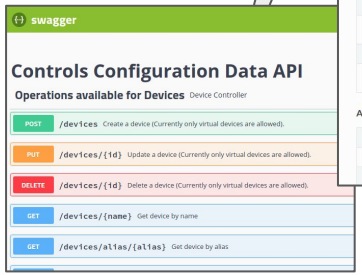
- Manages in-process JVM
- Zero-copy arrays between Python & Java
- Type annotations with **`stubgenj`**
- JARs automatically installed into the virtual environment

openapi-generator
(+post-processing)

- Auto-generated code for models and controllers
- Type annotations auto-injected using **`libcst`**
- Thin layer provides user friendly object oriented API

Swagger / OpenAPI

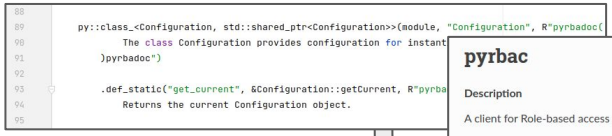


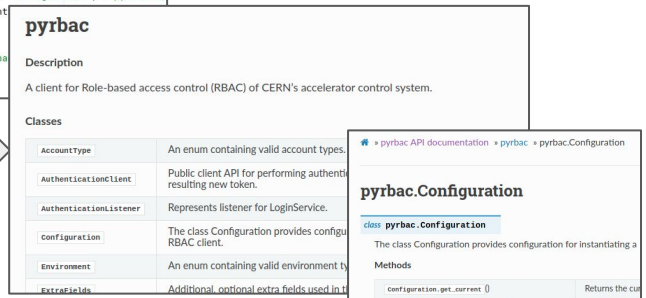


Python as glue

- Essential for accessing a wide range of control system functionality from Python
- Binding to existing APIs in other languages was the quickest and most resource-efficient approach
- All APIs should be type annotated for IDE autocompletion and static analysis

pybind11





- C++ based binding definition
- Easy to craft Pythonic bindings
- Creates shared-object module
- Released as pre-compiled **`manylinux2014`** Python wheels

Conclusion

There has been significant growth in the use of Python in the accelerator sector at CERN, with support for high-level accelerator controls having been introduced in 2019. To facilitate this, key infrastructure has been put in place, including development tools for a simple yet Pythonic experience, solutions for consistent application deployment and execution, and a suite of libraries for interacting with CERN's accelerator control system.

When considered together, the infrastructure presented here has resulted in a stable and effective platform from which Python users can efficiently build operational-quality libraries, applications and services for CERN's next accelerator run and beyond. Furthermore, such infrastructure is general in purpose, and would be a good fit for a diverse range of organisations wishing to adopt a robust and maintainable operational Python environment.