ITKPythonPackage Documentation

Jean-Christophe Fillion-Robin and Matt McCormick

Nov 23, 2020

Contents

This project provides a `setup.py` script to build ITK Python wheels and infrastructure to build ITK external module Python wheels.

ITK is an open-source, cross-platform system that provides developers with an extensive suite of software tools for image analysis.

To install the stable ITK Python package:

```
$ pip install itk
```

For more information on ITK's Python wrapping, see an introduction in the Book 1, Chapter 3 of the ITK Software Guide. There are also many downloadable examples documented in Sphinx.

CHAPTER 1

Quick start guide

1.1 Installation

To install the ITK Python package:

```
$ pip install itk
```

1.2 Usage

1.2.1 Basic examples

Here is a simple python script that reads an image, applies a median image filter (radius of 2 pixels), and writes the resulting image in a file.

```
#!/usr/bin/env python

import itk
import sys

input_filename = sys.argv[1]
output_filename = sys.argv[2]

image = itk.imread(input_filename)

median = itk.median_image_filter(image, radius=2)

itk.imwrite(median, output_filename)
```

There are two ways to instantiate filters with ITKPython:

- *Implicit (recommended)*: ITK type information is automatically detected from the data. Typed filter objects and images are implicitly created.

```
image = itk.imread(input_filename)
# Use ITK's functional, Pythonic interface. The filter type is implied by the
# type of the input image. The filter is eagerly executed, and the output image
# is directly returned.
smoothed = itk.median_image_filter(image)

# Alternatively, create filter objects. These filter objects can be connected in
# a pipeline to stream-process large datasets. To generate the output of the
# pipeline, .Update() must explicitly be called on the last filter of the
# pipeline.
#
# We can implicitly instantiate the filter object based on the type
# of the input image in multiple ways.

# Use itk.ImageFileReader instead of the wrapping function,
# itk.imread to illustrate this example.
reader = itk.ImageFileReader.New(FileName=input_filename)
# Here we specify the filter input explicitly
median = itk.MedianImageFilter.New(Input=reader.GetOutput())
# Same as above but shortened. Input does not have to be specified.
median = itk.MedianImageFilter.New(reader.GetOutput())
# Same as above. .GetOutput() does not have to be specified.
median = itk.MedianImageFilter.New(reader)

median.Update()
smoothed = median.GetOutput()
```

- *Explicit*: This can be useful if an appropriate type cannot be determined implicitly, e.g. with the *CastImageFilter*, and when a different filter type than the default is desired.

Explicit instantiation of a median image filter:

```
PixelType = itk.F
ImageType = itk.Image[PixelType,2]

# Functional interface
image = itk.imread(input_filename, PixelType)
smoothed = itk.median_image_filter(image, ttype=(ImageType, ImageType))

# Object-oriented interface
reader = itk.ImageFileReader[ImageType].New(FileName=input_filename)
median = itk.MedianImageFilter[ImageType, ImageType].New()
median.SetInput(reader.GetOutput())
median.Update()
smoothed = median.GetOutput()
```

Explicit instantiation of cast image filter:

```
image = itk.imread(input_filename)
InputType = type(image)
# Find input image dimension
input_dimension = image.GetImageDimension()
# Select float as output pixel type
OutputType = itk.Image[itk.UC, input_dimension]

# Functional interface
casted = itk.cast_image_filter(image, ttype=(InputType, OutputType))
```

(continues on next page)

(continued from previous page)

```
# Object-oriented interface
cast_filter = itk.CastImageFilter[InputType, OutputType].New()
cast_filter.SetInput(image)
cast_filter.Update()
casted = cast_filter.GetOutput()
```

1.2.2 ITK Python types

C++ type	Python type
float	itk.F
double	itk.D
unsigned char	itk.UC
std::complex<float>	itk.complex[itk.F]

This list is not exhaustive and is only presented to illustrate the type names. The complete list of types can be found in the ITK Software Guide.

Types can also be obtained from their name in the C programming language:

```
itk.F == itk.ctype('float') # True
```

1.2.3 Instantiate an ITK object

There are two types of ITK objects. Most ITK objects (images, filters, adapters, ...) are instantiated the following way:

```
InputType = itk.Image[itk.F,3]
OutputType = itk.Image[itk.F,3]
median = itk.MedianImageFilter[InputType, OutputType].New()
```

Some objects (matrix, vector, RGBPixel, ...) do not require the attribute *.New()* to be added to instantiate them:

```
pixel = itk.RGBPixel[itk.D]()
```

In case of doubt, look at the attributes of the object you are trying to instantiate.

1.2.4 Filter Parameters

ITK filter parameters can be specified in the following ways:

```
# Pythonic snake case keyword arguments:
#
#   number_of_iterations
#
smoothed = itk.anti_alias_binary_image_filter(image,
        number_of_iterations=3)

# CamelCase keyword arguments:
#
#   NumberOfIterations
```

(continues on next page)

(continued from previous page)

```
#
smoother = itk.AntiAliasBinaryImageFilter.New(image,
        NumberOfIterations=3)
smoother.Update()
smoothed = smoother.GetOutput()

# CamelCase Set method:
#
#   SetNumberOfIterations
#
smoother = itk.AntiAliasBinaryImageFilter.New(image)
smoother.SetNumberOfIterations(3)
smoother.Update()
smoothed = smoother.GetOutput()
```

1.3 Mixing ITK and NumPy

A common use case for using ITK in Python is to mingle NumPy and ITK operations on raster data. ITK provides a large number of I/O image formats and several sophisticated image processing algorithms not available in any other packages. The ability to intersperse that with numpy special purpose hacking provides a great tool for rapid prototyping.

The following script shows how to integrate NumPy and ITK:

```
import itk
import numpy as np

# Read input image
itk_image = itk.imread(input_filename)

# Run filters on itk.Image

# View only of itk.Image, data is not copied
np_view = itk.array_view_from_image(itk_image)

# Copy of itk.Image, data is copied
np_copy = itk.array_from_image(itk_image)


# Do NumPy stuff...


# Convert back to ITK, view only, data is not copied
itk_np_view = itk.image_view_from_array(np_copy)

# Convert back to ITK, data is copied
itk_np_copy = itk.image_from_array(np_copy)

# Save result
itk.imwrite(itk_np_view, output_filename)
```

Similar functions are available to work with *itk.Matrix*, VNL vectors and matrices:

```
# VNL matrix from array
arr = np.zeros([3,3], np.uint8)
matrix = itk.vnl_matrix_from_array(arr)

# Array from VNL matrix
arr = itk.array_from_vnl_matrix(matrix)

# VNL vector from array
vec = np.zeros([3], np.uint8)
vnl_vector = itk.vnl_vector_from_array(vec)

# Array from VNL vector
vec = itk.array_from_vnl_vector(vnl_vector)

# itk.Matrix from array
mat = itk.matrix_from_array(np.eye(3))

# array from itk.Matrix
arr = itk.array_from_matrix(mat)
```

1.4 Examples

Examples can be found in the ITKExamples project.

ITK is organized into *modules*. Modules for ITK can be developed outside the ITK source tree as *remote modules*. The *remote module* can be made available in ITK's CMake configuration by contributing it as a *remote module*. Python packages can also be generated for remote modules and uploaded to the Python Package Index (PyPI)

This section describes how to create, build, and upload ITK remote module Python packages to PyPI.

CHAPTER 2

Prerequisites

Building wheels requires:

- CMake
- Git
- C++ Compiler - Platform specific requirements are summarized in scikit-build documentation.
- Python

CHAPTER 3

Create the module

To create an ITK module with Python wrapping, first run cookiecutter:

```
python -m pip install cookiecutter
python -m cookiecutter gh:InsightSoftwareConsortium/ITKModuleTemplate
# Fill in the information requested at the prompts
```

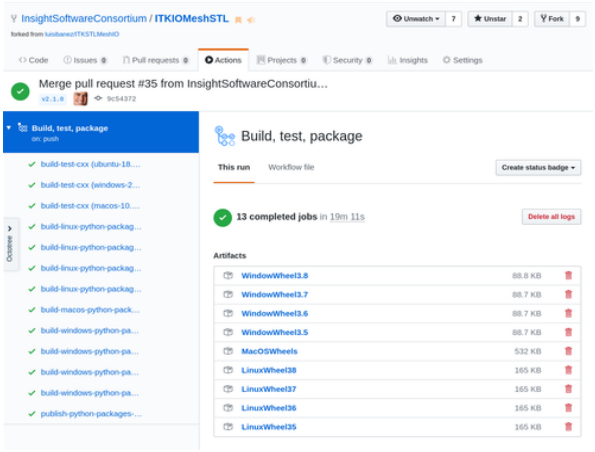
Then, add your classes. Reference documentation on how to populate the module can be found in the ITK Software Guide.

CHAPTER 4

GitHub automated CI package builds

Freely available GitHub Action continous integration (CI) build and test services for open source repositories are provided by GitHub. These services will build and test the C++ code for your module and also generate Linux, macOS, and Windows Python packages for your module.

For every pull request and push to the GitHub repository, a GitHub Action will run that builds and runs the repository's C++ tests and reports the results to the ITK CDash Dashboard. Python packages are also generated for every commit. Packages for a commit's build can be downloaded from the GitHub Action result page in the *Artifacts* Section.



4.1 Upload the packages to PyPI

First, register for an account on PyPI.

Next, create a *~/.pypirc* file with your login credentials:

```
[distutils]
index-servers =
```

(continues on next page)

(continued from previous page)

```
  pypi
  pypitest

[pypi]
username=<your-username>
password=<your-password>

[pypitest]
repository=https://test.pypi.org/legacy/
username=<your-username>
password=<your-password>
```

where <*your-username*> and <*your-password*> correspond to your PyPI account.

Then, upload wheels to the testing server. The wheels of dist/* are those that you have built locally or have downloaded from a recent build listed at *https://github.com/InsightSoftwareConsortium/<your-long-module-name>/actions*.

```
python -m pip install twine
python -m twine upload -r pypitest dist/*
```

Check out the packages on https://test.pypi.org/ the testing server.

Finally, upload the wheel packages to the production PyPI server:

```
python -m twine upload dist/*
```

Congratulations! Your packages can be installed with the commands:

```
python -m pip install --upgrade pip
python -m pip install itk-<your-short-module-name>
```

where *itk-*<*your-short-module-name*> is the short name for your module that is specified in your setup.py file.

4.2 Automate PyPI Package Uploads

Automated uploads of Python packages to the Python package index, PyPI will occur after adding a PyPI upload token to GitHub and creating a Git tag. Create a PyPI API token by logging in to https://pypi.org/manage/account/token/. Generally, for the token name use:

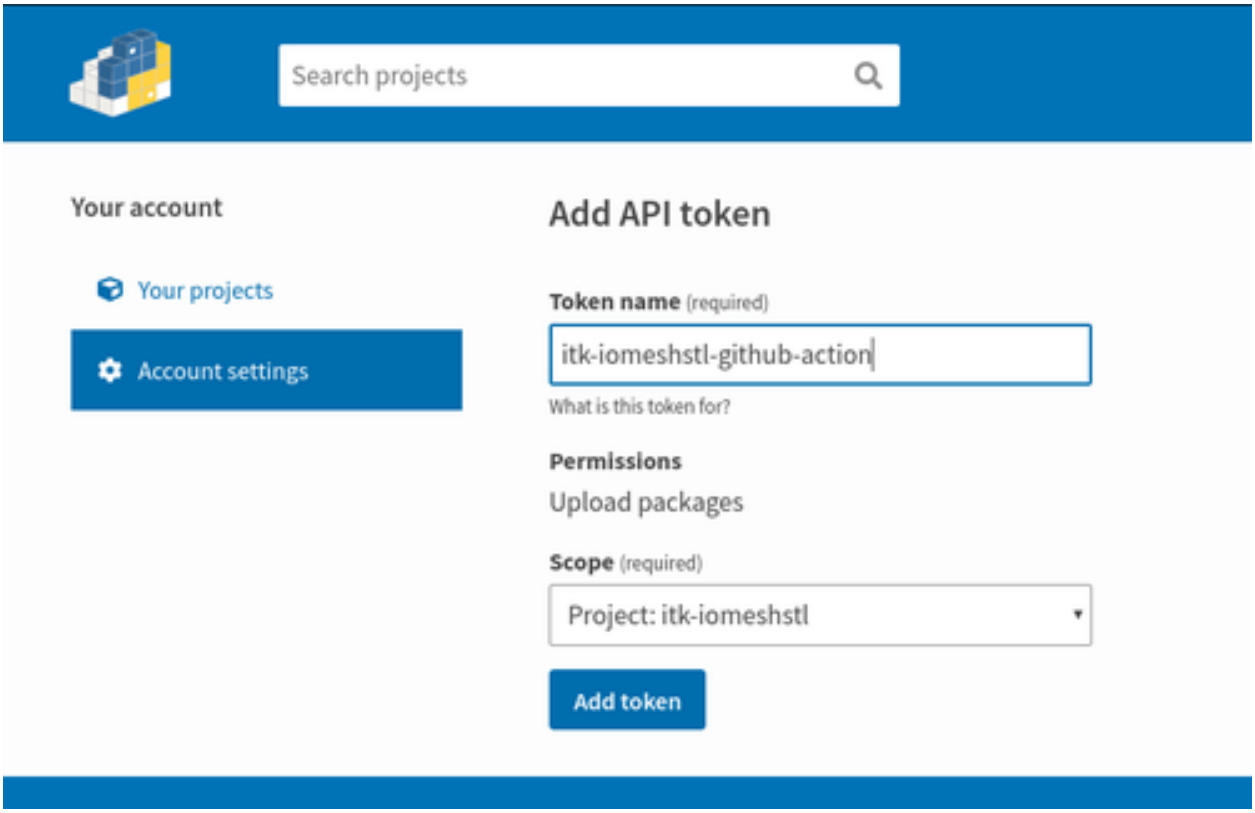
```
itk-<your-short-module-name>-github-action
```

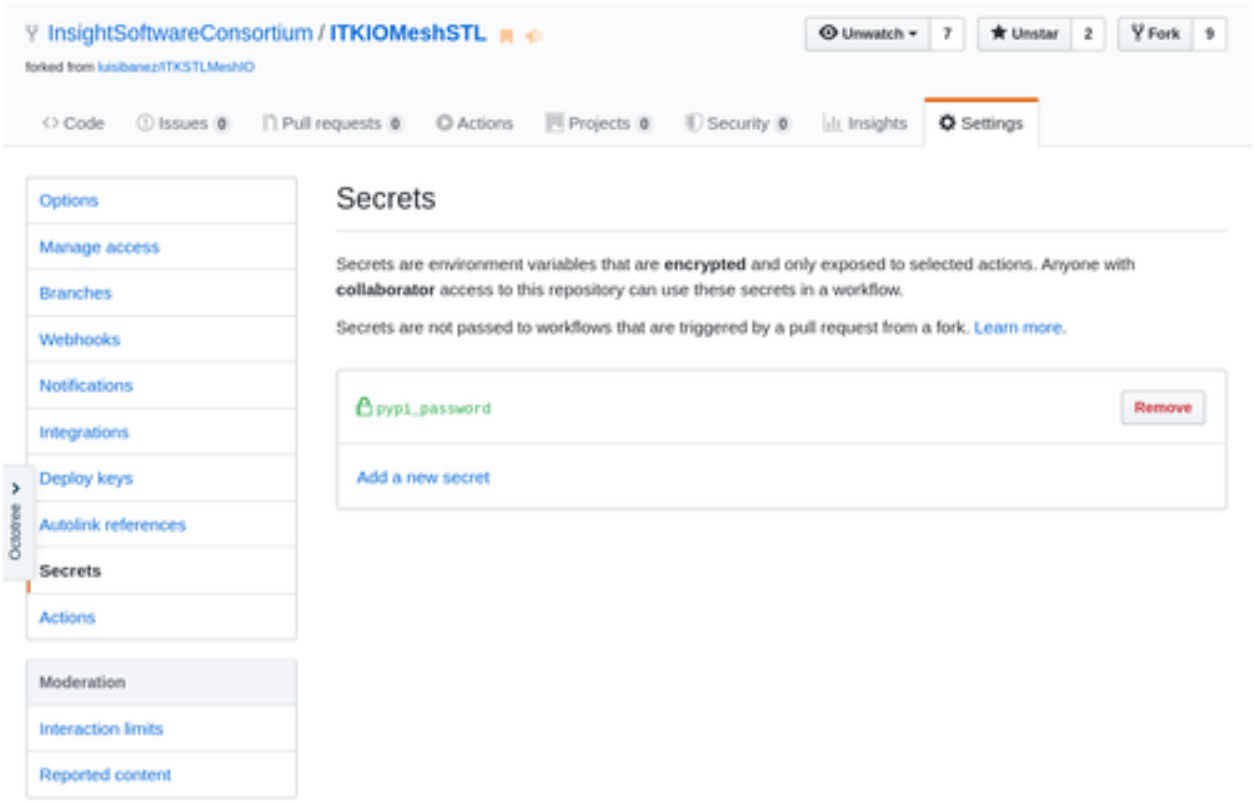
and for the scope use:

```
itk-<your-short-module-name>
```

where <*your-short-module-name*> is the short name for your module that is specified in your setup.py file. That scope will be available if you have already uploaded a first set of wheels via twine as described above; and that is the recommended approach. Otherwise, if you are creating the project at this time, choose an unlimited scope, but be careful with the created token.

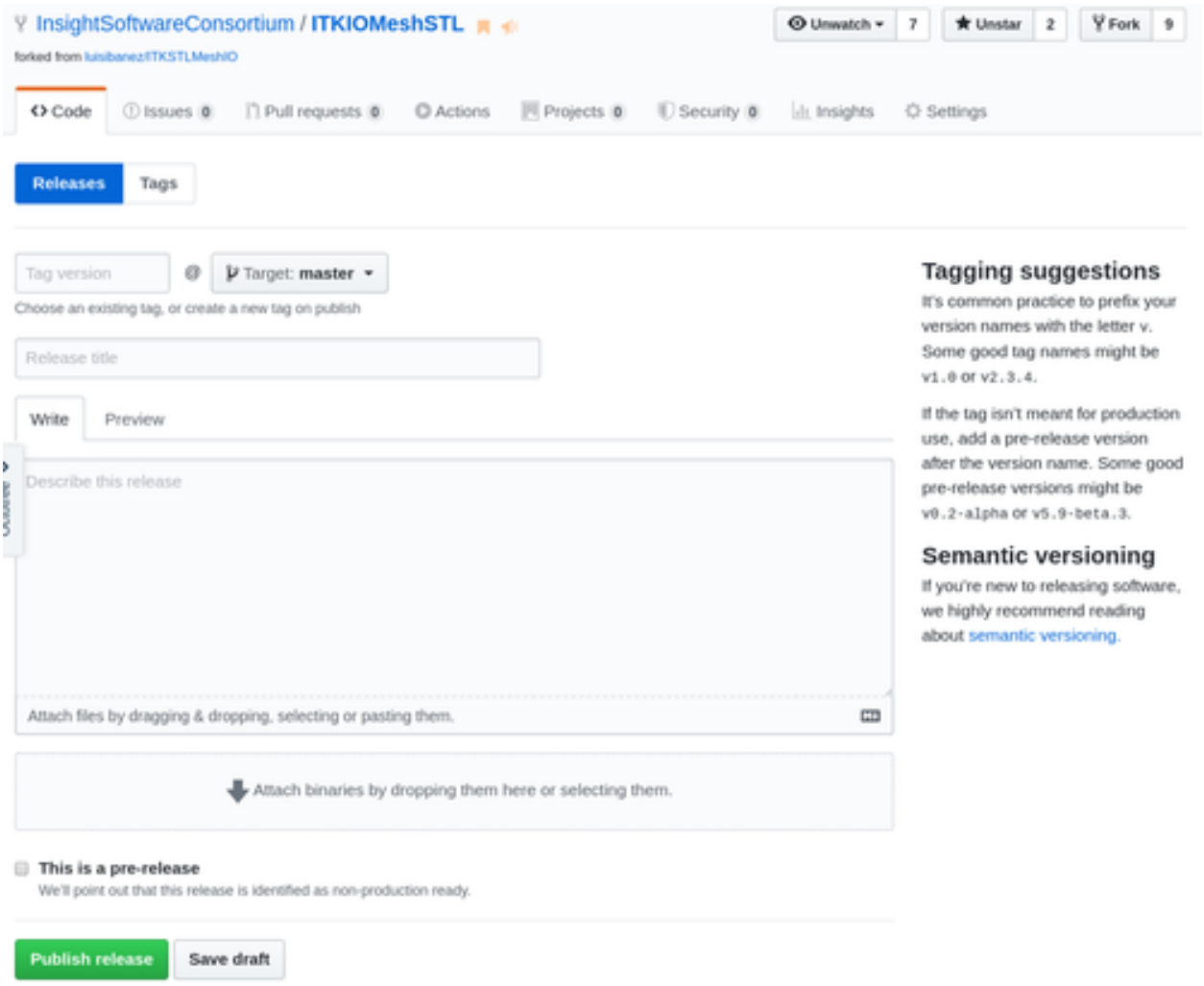
Then, add the API token to the GitHub repository *https://github.com/InsightSoftwareConsortium/<your-long-module-name>*. Choose the *Settings -> Secrets* page and add a key called *pypi_password*, setting the password to be the token string that begins with *pypi-*. Note that this will be a *token* instead of a password. Limit the scope of the token to the individual package as a best practice.


Search projects
Your account
Your projects
Account settings
Add API token
Token name (required)
itk-iomeshstl-github-action
What is this token for?
Permissions
Upload packages
Scope (required)
Project: itk-iomeshstl
Add token


InsightSoftwareConsortium / ITKIOMeshSTL
forked from luisibanez/ITKSTLMeshIO
Unwatch 7
Unstar 2
Fork 9
Code
Issues 0
Pull requests 0
Actions
Projects 0
Security 0
Insights
Settings
Options
Manage access
Branches
Webhooks
Notifications
Integrations
Deploy keys
Autolink references
Secrets
Actions
Moderation
Interaction limits
Reported content
Secrets
Secrets are environment variables that are encrypted and only exposed to selected actions. Anyone with collaborator access to this repository can use these secrets in a workflow.
Secrets are not passed to workflows that are triggered by a pull request from a fork. Learn more.
pypi_password
Remove
Add a new secret

To push packages to PyPI, first, make sure to update the *version* for your package in the *setup.py* file. The initial version might be *0.1.0* or *1.0.0*. Subsequent versions should follow semantic versioning.

Then, create a Git tag corresponding to the version. A Git tag can be created in the GitHub user interface via *Releases -> Draft a new release*.



CHAPTER 5

Automated platform scripts

Automated scripts are available in this repository to build Python packages that are binary compatible with the Python distributions provided by Python.org, Anaconda, and package managers like apt or Homebrew. The following sections outline how to use the associated scripts for Linux, macOS, and Windows.

Once the builds are complete, Python packages will be available in the *dist* directory.

5.1 Linux

To build portable Python packages on Linux, first install Docker.

For the first local build, clone the *ITKPythonPackage* repository inside your and download the required ITK binary builds:

```
cd ~/ITKMyModule
git clone https://github.com/InsightSoftwareConsortium/ITKPythonPackage
./ITKPythonPackage/scripts/dockcross-manylinux-download-cache-and-build-module-wheels.
↪sh
```

For subsequent builds, just call the build script:

```
./ITKPythonPackage/scripts/dockcross-manylinux-build-module-wheels.sh
```

5.2 macOS

First, install the Python.org macOS Python distributions. This step requires sudo:

```
cd ~/ITKMyModule
git clone https://github.com/InsightSoftwareConsortium/ITKPythonPackage
./ITKPythonPackage/scripts/macpython-install-python.sh
```

Then, build the wheels:

```
./ITKPythonPackage/scripts/macpython-build-wheels.sh
```

5.3 Windows

First, install Microsoft Visual Studio 2015, Git, and CMake, which should be added to the system PATH environmental variable.

Open a PowerShell terminal as Administrator, and install Python:

```
PS C:\> Set-ExecutionPolicy Unrestricted
PS C:\> $pythonArch = "64"
PS C:\> iex ((new-object net.webclient).DownloadString('https://raw.githubusercontent.
↪com/scikit-build/scikit-ci-addons/master/windows/install-python.ps1'))
```

In a PowerShell prompt, run the *windows-build-wheels.ps1* script:

```
PS C:\Windows> cd C:\ITKMyModule
PS C:\ITKMyModule> git clone https://github.com/InsightSoftwareConsortium/
↪ITKPythonPackage.git IPP
PS C:\ITKMyModule> .\ITKPythonPackage\scripts\windows-download-cache-and-build-module-
↪wheels.ps1
```

CHAPTER 6

Build ITK Python packages

This section describes how to builds ITK's Python packages. In most cases, the *pre-built ITK binary wheels can be used*.

ITK Python packages are built nightly on Kitware build systems and uploaded to the ITKPythonPackage GitHub releases page.

6.1 Prerequisites

Building wheels requires:

- CMake
- Git
- C++ Compiler - Platform specific requirements are summarized in scikit-build documentation.
- Python

6.2 Automated platform scripts

Steps required to build wheels on Linux, macOS and Windows have been automated. The following sections outline how to use the associated scripts.

6.2.1 Linux

On any linux distribution with docker and bash installed, running the script dockcross-manylinux-build-wheels.sh will create 64-bit wheels for both python 2.x and python 3.x in the dist directory.

For example:

```
$ git clone https://github.com/InsightSoftwareConsortium/ITKPythonPackage.git
[...]

$ ./scripts/dockcross-manylinux-build-wheels.sh
[...]

$ ls -1 dist/
itk-4.11.0.dev20170218-cp27-cp27m-manylinux1_x86_64.whl
itk-4.11.0.dev20170218-cp27-cp27mu-manylinux1_x86_64.whl
itk-4.11.0.dev20170218-cp34-cp34m-manylinux1_x86_64.whl
itk-4.11.0.dev20170218-cp35-cp35m-manylinux1_x86_64.whl
itk-4.11.0.dev20170218-cp36-cp36m-manylinux1_x86_64.whl
```

6.2.2 macOS

First, install the Python.org macOS Python distributions. This step requires sudo:

```
./scripts/macpython-install-python.sh
```

Then, build the wheels:

```
$ ./scripts/macpython-build-wheels.sh
[...]

$ ls -1 dist/
itk-4.11.0.dev20170213-cp27-cp27m-macosx_10_6_x86_64.whl
itk-4.11.0.dev20170213-cp34-cp34m-macosx_10_6_x86_64.whl
itk-4.11.0.dev20170213-cp35-cp35m-macosx_10_6_x86_64.whl
itk-4.11.0.dev20170213-cp36-cp36m-macosx_10_6_x86_64.whl
```

6.2.3 Windows

First, install Microsoft Visual Studio 2015, Git, and CMake, which should be added to the system PATH environmental variable.

Open a PowerShell terminal as Administrator, and install Python:

```
PS C:\> Set-ExecutionPolicy Unrestricted
PS C:\> $pythonArch = "64"
PS C:\> iex ((new-object net.webclient).DownloadString('https://raw.githubusercontent.
↪com/scikit-build/scikit-ci-addons/master/windows/install-python.ps1'))
```

In a PowerShell prompt:

```
PS C:\Windows> cd C:\
PS C:\> git clone https://github.com/InsightSoftwareConsortium/ITKPythonPackage.git␣
↪IPP
PS C:\> cd IPP
PS C:\IPP> .\scripts\windows-build-wheels.ps1
[...]

PS C:\IPP> ls dist
    Directory: C:\IPP\dist
```

(continues on next page)

(continued from previous page)

```
    Mode                 LastWriteTime         Length Name
    ----                 -------------         ------ ----
    -a----         4/9/2017  11:14 PM       63274441 itk-4.11.0.dev20170407-cp35-
↪cp35m-win_amd64.whl
    -a----         4/10/2017   2:08 AM       63257220 itk-4.11.0.dev20170407-cp36-
↪cp36m-win_amd64.whl
```

We need to work in a short directory to avoid path length limitations on Windows, so the repository is cloned into C:IPP.

Also, it is very important to disable antivirus checking on the C:IPP directory. Otherwise, the build system conflicts with the antivirus when many files are created and deleted quickly, which can result in Access Denied errors. Windows 10 ships with an antivirus application, Windows Defender, that is enabled by default.

6.2.4 sdist

To create source distributions, sdist's, that will be used by pip to compile a wheel for installation if a binary wheel is not available for the current Python version or platform:

```
$ python setup.py sdist --formats=gztar,zip
[...]

$ ls -1 dist/
itk-4.11.0.dev20170216.tar.gz
itk-4.11.0.dev20170216.zip
```

6.3 Manual builds

6.3.1 Building ITK Python wheels

Build the ITK Python wheel with the following command:

```
mkvirtualenv build-itk
pip install -r requirements-dev.txt
python setup.py bdist_wheel
```

6.3.2 Efficiently building wheels for different version of python

If on a given platform you would like to build wheels for different version of python, you can download and build the ITK components independent from python first and reuse them when building each wheel.

Here are the steps:

- Build ITKPythonPackage with ITKPythonPackage_BUILD_PYTHON set to OFF
- Build "flavor" of package using:

```
python setup.py bdist_wheel -- \
  -DITK_SOURCE_DIR:PATH=/path/to/ITKPythonPackage-core-build/ITK
```

CHAPTER 7

Miscellaneous

Written by Jean-Christophe Fillion-Robin and Matt McCormick from Kitware Inc.

For more information about ITK, visit https://itk.org

CHAPTER 8

Indices and tables

- genindex
- modindex
- search