ITKPythonPackage Documentation

Jean-Christophe Fillion-Robin and Matt McCormick

Sep 06, 2023

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.

Quick start guide

1.1 Installation

To install the ITK Python package:

```
$ pip install itk
```

1.2 Usage

1.2.1 Basic example

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.

```python
#!/usr/bin/env python3

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)
```

1.2.2 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 the SciPy ecosystem provides a great tool for rapid prototyping.

The following script shows how to integrate NumPy and *itk.Image*:

```python
import itk
import numpy as np

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

# Run filters on itk.Image

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

# Copy of itk.Image, pixel data is copied
array_copy = itk.array_from_image(itk_image)
# Equivalent
array_copy = np.asarray(itk_image)

# Image metadata
# Sequences, e.g. spacing, are in zyx (NumPy) indexing order
metadata = dict(itk_image)

# Pixel array and image metadata together
# in standard Python data types + NumPy array
# Sequences, e.g. spacing, are in xyz (ITK) indexing order
image_dict = itk.dict_from_image(itk_image)


# Do interesting things...


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

# Convert back to ITK, data is copied
itk_image_copy = itk.image_from_array(array_copy)

# Add the metadata
for k, v in metadata.items():
    itk_image_view[k] = v

# Save result
itk.imwrite(itk_image_view, output_image_filename)

# Convert back to itk image data structure
itk_image = itk.image_from_dict(image_dict)
```

NumPy and *itk.Mesh*:

```python
# Read input mesh
itk_mesh = itk.meshread(input_mesh_filename)
```

```python
# Convert to standard Python data types + NumPy arrays
mesh_dict = itk.dict_from_mesh(itk_mesh)


# Do interesting things...


# Convert back to itk mesh data structure
itk_mesh = itk.mesh_from_dict(mesh_dict)

# Save result
itk.meshwrite(itk_mesh, output_mesh_filename)
```

NumPy and *itk.Transform*:

```python
# Read input transforms
#
# This is a Python list
#
# When there is more than one transformation
# the list defines a transformation chain
itk_transforms = itk.transformread(input_transform_filename)

# Convert to standard Python data types + NumPy arrays
transform_dicts = [itk.dict_from_transform(t) for t in itk_transforms]


# Do interesting things...


# Convert back to itk transform instance
itk_transforms = [itk.transform_from_dict(t) for t in transform_dicts]

# Save result
itk.transformwrite(itk_transforms, output_transform_filename)
```

NumPy and *itk.Matrix*, VNL vectors, and VNL matrices:

```python
# VNL matrix from np.ndarray
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 np.ndarray
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 np.ndarray
mat = itk.matrix_from_array(np.eye(3))

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

```
# Equivalent
arr = np.asarray(mat)
```

1.2.3 ITK and Xarray

An *itk.Image* can be converted to and from an xarray.DataArray while preserving metadata:

```
da = itk.xarray_from_image(image)

image = itk.image_from_xarray(da)
```

1.2.4 ITK and VTK

An *itk.Image* can be converted to and from a vtk.vtkImageData while preserving metadata:

```
vtk_image = itk.vtk_image_from_image(image)

image = itk.image_from_vtk_image(vtk_image)
```

1.2.5 ITK and napari

An *itk.Image* can be converted to and from a napari.layers.Image while preserving metadata with the itk-napari-conversion package.

1.2.6 ITK Python types

C++ type	Python type	NumPy dtype
float	itk.F	np.float32
double	itk.D	np.float64
unsigned char	itk.UC	np.uint8
std::complex<float>	itk.complex[itk.F]	np.complex64

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
```

To cast the pixel type of an image, use *.astype*:

```
image = itk.imread(input_filename)

# Cast to an unsigned char pixel type
cast_image = image.astype(itk.UC)

# Equivalent
cast_image = image.astype(np.uint8)
```

```
itk.imwrite(cast_image, output_filename)
```

1.2.7 Metadata dictionary

An *itk.Image* has a metadata dict of *key: value* pairs.

The metadata dictionary can be retrieved with:

```
meta_dict = dict(image)
```

For example:

```
In [3]: dict(image)
Out[3]:
{'0008|0005': 'ISO IR 100',
 '0008|0008': 'ORIGINAL\\PRIMARY\\AXIAL',
 '0008|0016': '1.2.840.10008.5.1.4.1.1.2',
 '0008|0018': '1.3.12.2.1107.5.8.99.484849.834848.79844848.2001082217554549',
 '0008|0020': '20010822',
```

Individual dictionary items can be accessed or assigned:

```
print(image['0008|0008'])

image['origin'] = [4.0, 2.0, 2.0]
```

In the Python dictionary interface to image metadata, keys for the spatial metadata, the *'origin'*, *'spacing'*, and *'direction'*, are reversed in order from *image.GetOrigin()*, *image.GetSpacing()*, *image.GetDirection()* to be consistent with the NumPy array index order resulting from pixel buffer array views on the image.

1.2.8 Access pixel data with NumPy indexing

Array views of an *itk.Image* provide a way to set and get pixel values with NumPy indexing syntax, e.g.:

```
In [6]: image[0,:2,4] = [5,5]

In [7]: image[0,:4,4:6]
Out[7]:
NDArrayITKBase([[    5,  -997],
                [    5, -1003],
                [ -993,  -999],
                [ -996,  -994]], dtype=int16)
```

1.2.9 Input/Output (IO)

Convenient functions are provided read and write from ITK's many supported file formats:

```
image = itk.imread('image.tif')

# Read in with a specific pixel type.
image = itk.imread('image.tif', itk.F)
```

```python
# Read in an image series.
# Pass a sorted list of files.
image = itk.imread(['image1.png', 'image2.png', 'image3.png'])

# Read in a volume from a DICOM series.
# Pass a directory.
# Only a single series, sorted spatially, will be returned.
image = itk.imread('/a/dicom/directory/')

# Write an image.
itk.imwrite(image, 'image.tif')


# Read a mesh.
mesh = itk.meshread('mesh.vtk')

# Write a mesh.
itk.meshwrite(mesh, 'mesh.vtk')


# Read a spatial transform.
transform = itk.transformread('transform.h5')

# Write a spatial transform.
itk.transformwrite(transform, 'transform.h5')
```

1.2.10 Image filters and Image-like inputs and outputs

All *itk* functional image filters operate on an *itk.Image* but also:

- xarray.DataArray *

- numpy.ndarray

- dask.array.Array

* Preserves image metadata

1.2.11 Filter parameters

ITK filter parameters can be specified in the following ways:

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

# CamelCase keyword arguments:
#
#   NumberOfIterations
#
smoother = itk.AntiAliasBinaryImageFilter.New(image, NumberOfIterations=3)
smoother.Update()
```

```python
smoothed = smoother.GetOutput()

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

1.2.12 Filter types

In *itk*, filters are optimized at compile time for each image pixel type and image dimension. There are two ways to instantiate these filters with the *itk* Python wrapping:

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

```python
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.
ImageType = itk.Image[itk.UC, 2]
reader = itk.ImageFileReader[ImageType].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 or when a different filter type than the default is desired.

To specify the type of the filter, use the *ttype* keyword argument. Explicit instantiation of a median image filter:

```python
# An apriori ImageType
PixelType = itk.F
```

```
ImageType = itk.Image[PixelType,2]
image = itk.imread(input_filename, PixelType)

# An image type dynamically determined from the type on disk
image = itk.imread(input_filename)
ImageType = type(image)

# Functional interface
# The `ttype` keyword argument specifies the filter type.
smoothed = itk.median_image_filter(image, ttype=(ImageType, ImageType))

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

1.2.13 Instantiate an ITK object

There are two types of ITK objects. Most ITK objects, such as images, filters, or 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, like a Matrix, Vector, or RGBPixel, do not require the attribute *.New()* to be added to instantiate them:

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

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

1.3 Examples

Examples can be found in the ITKSphinxExamples 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

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.

GitHub automated CI package builds

Freely available GitHub Actions 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.
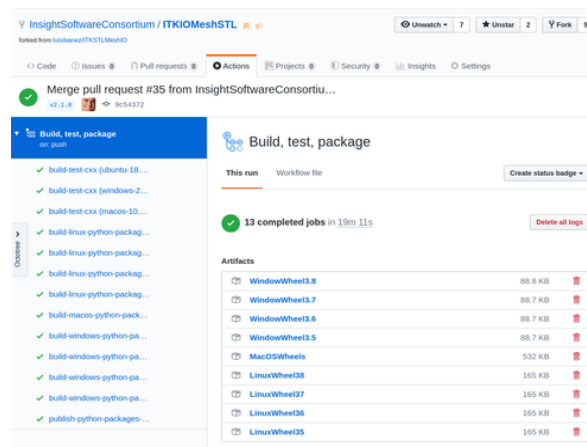
Reusable workflows available in [ITKRemoteModuleBuildTestPackageAction](https://github.com/InsightSoftwareConsortium/ITKRemoteModuleBuildTestPackageAction) can be used to handle the build-test-package process for a majority of ITK external modules with minimal extra development.

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 =
  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.

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*.

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 2022, 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
```

5.4 Other Notes

ITK modules sometimes depend on third-party libraries. To include third-party libraries in development wheels for distribution, first add the library path to *LD_LIBRARY_PATH* on Linux, *DYLD_LIBRARY_PATH* on MacOS, or *PATH* on Windows. Then, run the platform build script.

ITK modules sometimes depend on other ITK modules. For instance, to build [ITKBSplineGradient](https://github.com/InsightSoftwareConsortium/ITKBSplineGradient) the user must first build ITK and then [ITKMeshToPoly-Data](https://github.com/InsightSoftwareConsortium/ITKmeshtopolydata). ITKPythonPackage scripts support iterative prerequisite ITK module dependencies with the *ITK_MODULE_PREQ* environment variable.

For Python build scripts, the ordered list of ITK module dependencies must be formatted as follows:

` ITK_MODULE_PREQ=<module_org>/<module_name>@<module_tag>:<module_org>/<module_name>@<module_tag>:... `

Where - *module_org* is the name of a Github organization to use to fetch the module, i.e. "InsightSoftwareConsortium"; - *module_name* is the name of the module, i.e. "ITKMeshToPolyData"; - *module_tag* is the git tag or commit hash to use to fetch the module, i.e. "v1.0.0"

Module names must be provided in order of dependencies for the build to succeed.

For more information see the [build scripts directory](https://github.com/InsightSoftwareConsortium/ITKPythonPackage/tree/master/scripts).

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-manylinux2014_x86_64.whl
itk-4.11.0.dev20170218-cp27-cp27mu-manylinux2014_x86_64.whl
itk-4.11.0.dev20170218-cp34-cp34m-manylinux2014_x86_64.whl
itk-4.11.0.dev20170218-cp35-cp35m-manylinux2014_x86_64.whl
itk-4.11.0.dev20170218-cp36-cp36m-manylinux2014_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)

```
    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:

```
python3 -m venv build-itk
./build-itk/bin/pip install --upgrade pip
./build-itk/bin/pip install -r requirements-dev.txt numpy
./build-itk/bin/python setup.py bdist_wheel
```

6.3.2 Build a wheel for a custom version of ITK

To build a wheel for a custom version of ITK, point to your ITK git repository with the *ITK_SOURCE_DIR* CMake variable:

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

Other CMake variables can also be passed with *-D* after the double dash.

Miscellaneous

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

It is covered by the Apache License, Version 2.0:

http://www.apache.org/licenses/LICENSE-2.0

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

CHAPTER 8

Indices and tables

- genindex
- modindex
- search