pydicom Documentation

Release 1.0a

Darcy Mason and pydicom contributors

May 30, 2017

Contents

Contents:

Getting Started with pydicom

Brief overview of pydicom and how to install.

Introduction

pydicom is a pure python package for working with DICOM files such as medical images, reports, and radiotherapy objects.

pydicom makes it easy to read these complex files into natural pythonic structures for easy manipulation. Modified datasets can be written again to DICOM format files.

Here is a simple example of using pydicom in an interactive session, to read a radiotherapy plan file, change the patient setup from head-first-supine to head-first-prone, and save to a new file:

```
>>> from pydicom import dicomio
>>> ds = dicomio.read_file("rtplan.dcm")   # plan dataset
>>> ds.PatientName
'Last^First^mid^pre'
>>> ds.dir("setup")     # get a list of tags with "setup" somewhere in the name
['PatientSetupSequence']
>>> ds.PatientSetupSequence[0]
(0018, 5100) Patient Position                    CS: 'HFS'
(300a, 0182) Patient Setup Number                IS: '1'
(300a, 01b2) Setup Technique Description         ST: ''
>>> ds.PatientSetupSequence[0].PatientPosition = "HFP"
>>> ds.save_as("rtplan2.dcm")
```

pydicom is not a DICOM server[1], and is not primarily about viewing images. It is designed to let you manipulate data elements in DICOM files with python code.

pydicom is easy to install and use, and because it is a pure python package, it should run anywhere python runs.

One limitation of pydicom: compressed pixel data (e.g. JPEG) cannot be altered in an intelligent way as it can be for uncompressed pixels. Files can always be read and saved, but compressed pixel data cannot easily be modified.

---

[1] For DICOM network capabilities, see the pynetdicom project.

License

pydicom has an MIT-based license.

Installing

As a pure python package, pydicom is easy to install and has no requirements other than python itself (the NumPy library is recommended, but is only required if manipulating pixel data).

Note: In addition to the instructions below, pydicom can also be installed through the Python(x,y) distribution, which can install python and a number of packages[2] (including pydicom) at once.

Prerequisites

- python 2.6, 2.7, 3.3 or later

- NumPy – optional, only needed if manipulating pixel data

Note: To run unit tests when using python 2.6, Unittest2 is required.

Python installers can be found at the python web site (http://python.org/download/). On Windows, the Activepython distributions are also quite good.

Installing using pip (all platforms)

The easiest way to install pydicom is using pip:

```
pip install pydicom
```

Depending on your python version, there may be some warning messages, but the install should still be ok.

Note: Pip comes pre-installed with Python 3.x.

Installing from source (all platforms)

- Download the source code directly, or clone the repo with Github's desktop application.

- In a command terminal, move to the directory with the setup.py file

- With admin privileges, run `python setup.py install`

  - With some linux variants, for example, use `sudo python setup.py install`

  - With other linux variants you may have to `su` before running the command.

---

[2] If using python(x,y), other packages you might be interested in include IPython (an indispensable interactive shell with auto-completion, history etc), Numpy (optionally used by pydicom for pixel data), and ITK/VTK or PIL (image processing and visualization).

Installing on Mac

Using pip as described above is recommended. However, there was previously a MacPorts portfile. This is maintained by other users and may not immediately be up to the latest release.

Using pydicom

Once installed, the package can be imported at a python command line or used in your own python program with `import pydicom`. See the examples directory for both kinds of uses. Also see the *User Guide* for more details of how to use the package.

Support

Please join the pydicom discussion group to ask questions or give feedback. Bugs can be submitted through the issue tracker. Besides the example directory, cookbook recipes are encouraged to be posted on the wiki page

New versions, major bug fixes, etc. will also be announced through the group.

Next Steps

To start learning how to use pydicom, see the *Pydicom User Guide*.

Footnotes::

Pydicom User Guide

pydicom object model, description of classes, examples

Dataset

`Dataset` is the main object you will work with directly. Dataset is derived from python's `dict`, so it inherits (and overrides some of) the methods of `dict`. In other words it is a collection of key:value pairs, where the key value is the DICOM (group,element) tag (as a Tag object, described below), and the value is a DataElement instance (also described below).

A dataset could be created directly, but you will usually get one by reading an existing DICOM file:

```
>>> import pydicom
>>> ds = pydicom.read_file("rtplan.dcm") # (rtplan.dcm is in the testfiles directory)
```

You can display the entire dataset by simply printing its string (str or repr) value:

```
>>> ds
(0008, 0012) Instance Creation Date          DA: '20030903'
(0008, 0013) Instance Creation Time          TM: '150031'
(0008, 0016) SOP Class UID                   UI: RT Plan Storage
(0008, 0018) SOP Instance UID                UI: 1.2.777.777.77.7.7777.7777.
↪20030903150023
(0008, 0020) Study Date                      DA: '20030716'
(0008, 0030) Study Time                      TM: '153557'
(0008, 0050) Accession Number                SH: ''
(0008, 0060) Modality                        CS: 'RTPLAN'
```

Note: You can also view DICOM files in a collapsible tree using the example program dcm_qt_tree.py.

You can access specific data elements by name (DICOM 'keyword') or by DICOM tag number:

```
>>> ds.PatientName
'Last^First^mid^pre'
>>> ds[0x10,0x10].value
'Last^First^mid^pre'
```

In the latter case (using the tag number directly) a DataElement instance is returned, so the `.value` must be used to get the value.

You can also set values by name (DICOM keyword) or tag number:

```
>>> ds.PatientID = "12345"
>>> ds.SeriesNumber = 5
>>> ds[0x10,0x10].value = 'Test'
```

The use of names is possible because pydicom intercepts requests for member variables, and checks if they are in the DICOM dictionary. It translates the keyword to a (group,element) number and returns the corresponding value for that key if it exists.

To understand using Sequences in pydicom, please refer to this object model:

> **Dataset** (derived from python's *dict*)
>
> > **—> contains DataElement instances**
> >
> > > **–> the value of the data element can be one of:**
> > >
> > > > - a regular value like a number, string, etc.
> > > >
> > > > - a list of regular values (e.g. a 3-D coordinate)
> > > >
> > > > - **a Sequence instance** –> a Sequence is a list of Datasets (and so we come full circle)

DICOM Sequences are turned into python `list` s. Items in the sequence are referenced by number, beginning at index 0 as per python convention.:

```
>>> ds.BeamSequence[0].BeamName
'Field 1'
```

Using DICOM keywords is the recommended way to access data elements, but you can also use the tag numbers directly, such as:

```
>>> # Same thing with tag numbers:
>>> ds[0x300a,0xb0][0][0x300a,0xc2].value
'Field 1'
>>> # yet another way, using another variable
>>> beam1=ds[0x300a,0xb0][0]
>>> beam1.BeamName, beam1[0x300a,0xc2].value
('Field 1', 'Field 1')
```

If you don't remember or know the exact tag name, Dataset provides a handy *dir()* method, useful during interactive sessions at the python prompt:

```
>>> ds.dir("pat")
['PatientBirthDate', 'PatientID', 'PatientName', 'PatientSetupSequence', 'PatientSex']
```

`dir` will return any DICOM tag names in the dataset that have the specified string anywhere in the name (case insensitive).

Note: Calling *dir* with no string will list all tag names available in the dataset.

You can also see all the names that pydicom knows about by viewing the *_dicom_dict.py* file. You could modify that file to add tags that pydicom doesn't already know about.

Under the hood, Dataset stores a DataElement object for each item, but when accessed by name (e.g. `ds.PatientName`) only the *value* of that DataElement is returned. If you need the whole DataElement (see the DataElement class discussion), you can use Dataset's data_element() method or access the item using the tag number:

```
>>> data_element = ds.data_element("PatientsName")  # or data_element = ds[0x10,0x10]
>>> data_element.VR, data_element.value
('PN', 'Last^First^mid^pre')
```

To check for the existence of a particular tag before using it, use the *in* keyword:

```
>>> "PatientName" in ds
True
```

To remove a data element from the dataset, use `del`:

```
>>> del ds.SoftwareVersions   # or del ds[0x0018, 0x1020]
```

To work with pixel data, the raw bytes are available through the usual tag:

```
>>> pixel_bytes = ds.PixelData
```

but to work with them in a more intelligent way, use `pixel_array` (requires the [NumPy library](#)):

```
>>> pix = ds.pixel_array
```

For more details, see *Working with Pixel Data*.

DataElement

The DataElement class is not usually used directly in user code, but is used extensively by Dataset. DataElement is a simple object which stores the following things:

- tag – a DICOM tag (as a Tag object)
- VR – DICOM value representation – various number and string formats, etc
- VM – value multiplicity. This is 1 for most DICOM tags, but can be multiple, e.g. for coordinates. You do not have to specify this, the DataElement class keeps track of it based on value.
- value – the actual value. A regular value like a number or string (or list of them), or a Sequence.

Tag

The Tag class is derived from python's `int`, so in effect, it is just a number with some extra behaviour:

- Tag enforces that the DICOM tag fits in the expected 4-byte (group,element)
- A Tag instance can be created from an int or from a tuple containing the (group,element) separately:

```
>>> from pydicom.tag import Tag
>>> t1=Tag(0x00100010) # all of these are equivalent
>>> t2=Tag(0x10,0x10)
>>> t3=Tag((0x10, 0x10))
```

```
>>> t1
(0010, 0010)
>>> t1==t2, t1==t3
(True, True)
```

- Tag has properties group and element (or elem) to return the group and element portions

- The `is_private` property checks whether the tag represents a private tag (i.e. if group number is odd).

Sequence

Sequence is derived from python's `list`. The only added functionality is to make string representations prettier. Otherwise all the usual methods of `list` like item selection, append, etc. are available.

Transition to pydicom 1.x

Important information on differences in pydicom post 1.0 vs pre-1.0

Introduction

As is often the case for major software version number changes, pydicom 1.0 breaks with the previous release of pydicom (0.9.9) in several ways. These require changes to user code to target the pydicom >= 1.0 package, or to check and deal with the differences between the versions.

Backwards-compatible changes post 1.0

- the library is no longer `dicom` but is `pydicom`, to match the package name

- short-form names such as `Beams` are no longer allowed; use the full keyword e.g. `BeamSequence`

- some less-used modules within pydicom have been renamed, e.g. `dicom.UID` is now `pydicom.uid`

Why was the package name changed? There are several reasons for this change:

- it is standard python practice for the package and the installed library to have the same name

- first-time users expect to be able to type `import pydicom` rather than `import dicom`, which has caused confusion

- it makes sense for search engines - with the correct name it is much easier to find relevant questions and example code online

The decision wasn't taken lightly, but with a great deal of discussion on the github issues list. Having made the leap, the rest of this guide should help smooth the way...

For authors of packages requiring pydicom < 1.0

The old pydicom releases have been split off into their own package, called `dicom`, which is now hosted on PyPI. This allows the old library `dicom` to co-exist alongside the new library `pydicom`.

The main things to do, to ensure your old pydicom code will remain functional, are:

1. you should `pip uninstall pydicom` and `pip install dicom` in your existing pydicom installs

2. If you have `requirements.txt` files, change the pydicom line from "pydicom" to "dicom"

   e.g. `pydicom==0.9.9`

   becomes `dicom==0.9.9`

3. Change your instructions to users to `pip install dicom` rather than `pip install pydicom`

Error messages relating to the pydicom transition

This section is here in the hopes of people getting directed to this page on searches. If that's you, then welcome! Hopefully the information here can get things going quickly for you.

For those with pydicom < 1.0 installed, on trying to import pydicom, they will get an ImportError message:

```
>>> import pydicom
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named pydicom
>>>
```

Your choice then is to update to pydicom >=1.0 (see Installing pydicom section), or to instead use `import dicom` and follow old-style pydicom syntax.

Conversely, if pydicom >= 1.0 is installed, the error message for `import dicom` will look like:

```
>>> import dicom
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named dicom
>>>
```

In this case you likely have installed pydicom >= 1.0, and so `dicom` library does not exist. You can simply `import pydicom` instead, and continue with the new pydicom, or, if you really need the old pydicom, then you should:

> pip install dicom

and you should be good to go.

Working with Pixel Data

How to work with pixel data in pydicom.

Introduction

pydicom tends to be "lazy" in interpreting DICOM data. For example, by default it doesn't do anything with pixel data except read in the raw bytes:

```
>>> import pydicom
>>> ds=pydicom.read_file("MR_small.dcm")
>>> ds.PixelData
'\x89\x03\xfb\x03\xcb\x04\xeb\x04\xf9\x02\x94\x01\x7f ...
...
```

`PixelData` contains the raw bytes exactly as found in the file. If the image is JPEG compressed, these bytes will be the compressed pixel data, not the expanded, uncompressed image. Whether the image is e.g. 16-bit or 8-bit, multiple frames or not, `PixelData` contains the same raw bytes. But there is a function that can shape the pixels more sensibly if you need to work with them...

pixel_array

> **Warning:** To work with the pixel_array property [NumPy](#) must be installed on your system.

A property of Dataset called `pixel_array` provides more useful pixel data for uncompressed images. The `pixel_array` property returns a NumPy array:

```
>>> import dicom
>>> ds=pydicom.read_file("MR_small.dcm")
>>> ds.pixel_array
```

```
array([[ 905, 1019, 1227, ...,  302,  304,  328],
       [ 628,  770,  907, ...,  298,  331,  355],
       [ 498,  566,  706, ...,  280,  285,  320],
       ...,
       [ 334,  400,  431, ..., 1094, 1068, 1083],
       [ 339,  377,  413, ..., 1318, 1346, 1336],
       [ 378,  374,  422, ..., 1369, 1129,  862]], dtype=int16)
>>> ds.pixel_array.shape
(64, 64)
```

NumPy can be used to modify the pixels, but if the changes are to be saved, they must be written back to the
`PixelData` attribute:

```
>>> for n,val in enumerate(ds.pixel_array.flat): # example: zero anything < 300
...     if val < 300:
...         ds.pixel_array.flat[n]=0
>>> ds.PixelData = ds.pixel_array.tostring()
>>> ds.save_as("newfilename.dcm")
```

Some changes may require other DICOM tags to be modified. For example, if the pixel data is reduced (e.g. a 512x512
image is collapsed to 256x256) then `ds.Rows` and `ds.Columns` should be set appropriately. You must explicitly
set these yourself; pydicom does not do so automatically.

`pixel_array` can also be used to pass image data to graphics libraries for viewing. See *Viewing Images* for details.

Viewing Images

How to use other packages with pydicom to view DICOM images

Introduction

pydicom is mainly concerned with getting at the DICOM data elements in files, but it is often desirable to view pixel data as an image. There are several options:

- Use any of the many DICOM viewer programs available
- use pydicom with matplotlib
- use pydicom with python's stdlib Tkinter module.
- use pydicom with the Python Imaging Library (PIL)
- use pydicom with wxPython

Using pydicom with matplotlib

matplotlib is available at http://matplotlib.sourceforge.net/. It can take 2-d image information from `Dataset.pixel_array` and display it. Here is an example:

```
>>> import pydicom
>>> import pylab
>>> ds=pydicom.read_file("CT_small.dcm")
>>> pylab.imshow(ds.pixel_array, cmap=pylab.cm.bone)
<matplotlib.image.AxesImage object at 0x0162A530>
>>> pylab.show()
>>>
```

Thanks to Roy Keyes for pointing out how to do this.

Using pydicom with Tkinter

The program pydicom_Tkinter.py in the `contrib` folder demonstrates how to show an image using the Tkinter graphics system, which comes standard with most python installs. It creates a Tkinter PhotoImage in a Label widget or a user-supplied widget.

Using pydicom with Python Imaging Library (PIL)

The module pydicom_PIL.py in the `contrib` folder uses PIL's `Image.show()` method after creating an Image instance from the pixel data and some basic information about it (bit depth, LUTs, etc)

Using pydicom with wxPython

The module imViewer-Simple.py in the `contrib` folder uses wxPython (also PIL, but it notes that it may not be strictly necessary) to display an image from a pydicom dataset.

Pydicom Reference Guide

Common pydicom functions called by user code

File Reading/Parsing

The main function to read and parse DICOM files using pydicom is `read_file`. It is coded in the module dicom.filereader, but is also imported when the pydicom package is imported:

```
>>> import pydicom
>>> dataset = pydicom.read_file(...)
```

If you need fine control over the reading, you can either call `read_partial` or use `open_dicom`. All are documented below:

File Writing

DICOM files can also be written using pydicom. There are two ways to do this. The first is to use `write_file` with a prexisting FileDataset (derived from Dataset) instance. The second is to use the `save_as` method on an Dataset instance.

Dataset

Indices and tables

- genindex
- modindex
- search