pydicom Documentation

Release 1.0a

Darcy Mason and pydicom contributors

January 27, 2017

Contents

Contents:

CHAPTER 1

Getting Started with pydicom

Brief overview of pydicom and how to install.

1.1 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:

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

1.2 License

pydicom has a license based on the MIT license.

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

1.3 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.

1.3.1 Prerequisites

- python 2.4 through 2.6 (or python 2.3 can be used for pydicom < 0.9.4)
- [NumPy (http://numpy.scipy.org/)] – optional, only needed if manipulating pixel data

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

1.3.2 Installing on Windows

On Windows, pydicom can be installed using the executable installer from the Downloads tab.

Alternatively, pydicom can be installed with easy_install, pip, or from source, as described in the sections below.

1.3.3 Installing using easy_install or pip (all platforms)

if you have setuptools installed, just use easy_install at the command line (you may need `sudo` on linux):

```
easy_install pydicom
```

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

pip is a newer install tool that works quite similarly to easy_install and can also be used.

1.3.4 Installing from source (all platforms)

- download the source code from the Downloads tab or checkout the mercurial repository source
- at a command line, change 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.
- for python < 2.6, you may get a syntax error message when the python files are "built" – this is due to some python 2.6 specific code in one unit test file. The installation seems to still be ok.

1.3.5 Installing on Mac

The instructions above for easy_install or installing from source will work on Mac OS. There is also a MacPorts portfile (py25-pydicom) available at http://trac.macports.org/browser/trunk/dports/python/py25-pydicom. This is maintained by other users and may not immediately be up to the latest release.

[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).

1.4 Using pydicom

Once installed, the package can be imported at a python command line or used in your own python program with `import dicom` (note the package name is `dicom`, not `pydicom` when used in code. See the examples directory for both kinds of uses. Also see the User Guide for more details of how to use the package.

1.5 Support

Please join the pydicom discussion group to ask questions, give feedback, post example code for others – in other words for any discussion about the pydicom code. New versions, major bug fixes, etc. will also be announced through the group.

1.6 Next Steps

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

CHAPTER 2

Pydicom User Guide

pydicom object model, description of classes, examples

2.1 Dataset

Dataset is the base object in pydicom's object model. The relationship between Dataset and other objects is:

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)

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 dicom
>>> ds = dicom.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 dicomtree.py.

You can access specific data elements by name or by DICOM tag number:

```
>>> ds.PatientsName
'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 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 name to a (group,element) number and returns the corresponding value for that key if it exists. The names are the descriptive text from the dictionary with spaces and apostrophes, etc. removed.

DICOM Sequences are turned into python `list` s. For these, the name is from the dictionary name with "sequence" removed, and the normal English plural added. So "Beam Sequence" becomes "Beams", "Referenced Film Box Sequence" becomes "ReferencedFilmBoxes". Items in the sequence are referenced by number, beginning at index 0 as per python convention.

```
>>> ds.Beams[0].BeamName
'Field 1'
>>> # 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')
```

Since you may not always remember the exact name, Dataset provides a handy *dir()* method, useful during interactive sessions at the python prompt:

```
>>> ds.dir("pat")
['PatientSetups', 'PatientsBirthDate', 'PatientsID', 'PatientsName', 'PatientsSex']
```

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

```
>>> "PatientsName" in ds
True
```

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

```
>>> del ds[0x10,0x1000]
>>> # OR
>>> tag = ds.data_element("OtherPatientIDs").tag
>>> del ds[tag]
```

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.

2.2 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.

2.3 Tag

The Tag class is derived from python's `long`, 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 a long or from a tuple containing the (group,element) separately:

```
>>> from dicom.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).

2.4 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.

CHAPTER 3

Transition to pydicom 1.x

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

3.1 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? Yes, this will cause some confusion for a while, and I apologize for this, but it will fade over time. 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...

3.2 For authors of packages requiring pydicom < 1.0

If you have authored code targeting the "old" dicom library, you have three options (at least):

> # update the code base to target pydicom >1.0 # update your dependencies to point to package `dicom` rather than `pydicom` # update dependencies to explictly target pydicom < 1, e.g. pydicom=0.9.9

The first option can be relatively simple for most projects. If you only use basic pydicom features, e.g. to read and write files, simply changing `import dicom` to `import pydicom` everywhere may be all that is needed.

The second and third options will get the same library installed. The second is preferred, as this will point to a repository explicit to the old `dicom` code. This makes it clear that your code has not been updated for pydicom 1, and allows people to install `pydicom` and `dicom` completely independently.

3.3 Error messages relating to the pydicom transition

This section is here in the hopes of people getting redirected 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.

CHAPTER 4

Working with Pixel Data

How to work with pixel data in pydicom.

4.1 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 dicom
>>> ds=dicom.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...

4.2 pixel_array

A property of Dataset called `pixel_array` provides more useful pixel data for uncompressed images. The NumPy numerical package must be installed on your system to use this property, because `pixel_array` returns a NumPy array:

```
>>> import dicom
>>> ds=dicom.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.

CHAPTER 5

Viewing Images

How to use other packages with pydicom to view DICOM images

5.1 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 Tkinter (comes standard with python)
- use pydicom with the Python Imaging Library (PIL)
- use pydicom with wxPython

5.2 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 dicom
>>> import pylab
>>> ds=dicom.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.

5.3 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.

5.4 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)

5.5 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.

CHAPTER 6

Pydicom Reference Guide

Common pydicom functions called by user code

6.1 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 dicom package is imported:

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

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

6.2 Dataset

CHAPTER 7

Indices and tables

- genindex
- modindex
- search