Octopus file format

Octopus saves all data in the simplest possible formats and all files can be opened without conversion in many different ways depending on what you are trying to do.

Metadata

All metadata (image width, height, times laser status, …) is saved in a simple text file ending in '.dth'. This text file can be opened with a word processor, BBEdit, Matlab, Mathematica, etc.

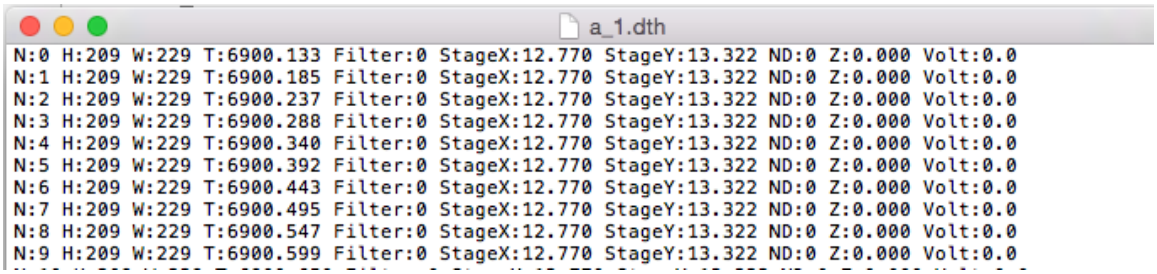


Figure 1 – the header file format (.dth)

Intensity (raw data)

All **intensity/pixel information** is stored in a binary format ending in the extension '.dat'. Binary files are needed for the computer to be able to keep up with the data streaming from the camera. The binary file can be read in hundreds of different ways, depending on your needs. I'll go through some of those use cases below.

Note that there is no need for any kind of conversion, and indeed, in general, you should never convert to e.g. .jpeg or .tiff, since this creates many problems.

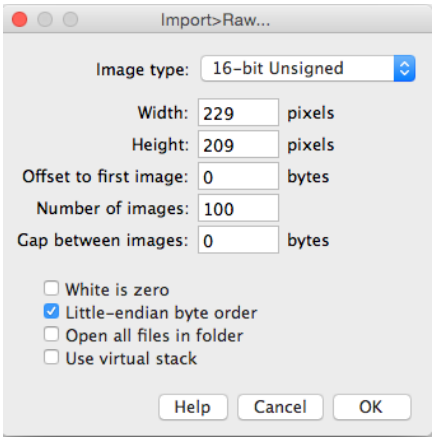
Once you convert to .jpeg, you loose any ability to quantitatively interpret the information, and since .jpeg is a lossy format, you loose information.

The **problem with tiff files** is that the camera information is 16 bit data, but most operating systems will only "show" tiffs with an 8 bit depth. So if you would like to look at what is inside the .dat file, and convert to tiff, you will either not be able to open that tiff file, or it will be rendered as completely black. Even if your operating system supports 16 bit tiffs (which Windows does not), the image will still be rendered as completely black, since the typical distribution of intensity values from the CCD is not scaled to cover the entire range from 0 – 65535.

Use cases

1. 'Quick and dirty' look at a .dat file. The easiest thing to do is to use ImageJ (or Fiji), which can read raw binary files.

File > **Import** > **Raw**…



Then select the .dat file, and then enter the information: Unsigned 16 bit integer, Little-endian byte order, and the Height and Width information, which depends on how you set your camera. A typical value will be 512 x 512, or, as in the case above (**Figure 1**), 209 x 229. Note that essentially all programs disagree about what is meant by the height and width of an image, so you may need to flip the values (giving 229 x 209) if your images look funny. This is true for all programs (Fiji, Matlab, Mathematica, …). You will also need to set the number of images to whatever value you set in Octopus, which defaults to storing 100 images per file, but this value is completely adjustable and up to you. At this point you will have the full dataset, as an image stack, in ImageJ (or Fiji). Note that the image may be completely white or black, depending on your experiment and how you set the gain etc. Then, go to

Image > **Adjust** > **Brightness/Contrast**…

to see where the data are in your experiment. Note that you cannot permanently apply any Brightness/Contrast values to the image stack, since it is in 16 bit format, and ImageJ/Fiji only allows you to do that with 8 bit images.

2. Quantitative analysis in Matlab. Right out the box, Matlab can read all octopus files directly.

```
fileID = fopen('a1.dat');
myImageArray = fread(fileID,'uint16');
```

At this point you have an array with all the data, and you can manipulate this in any way you want. Most people use 'reshape' to turn this into a simpler 3D array, e.g. [X Y T]. You can use 'imshow' on this array (or slices thereof) directly, to see/scale/process the images.

3. Quantitative analysis in Mathematica. Right out the box, Mathematica can read all octopus files directly.

```
file = "/Users/janliphardt/Desktop/a_1.dat";
bc = Import[file,"UnsignedInteger16"];
```

At this point you have all the data, and you can manipulate the data in any way you want. Most people use 'ArrayReshape[]' to turn everything into a simple 3D array, e.g. [X Y T] or whatever order they like most. You can manipulate the data as data, or you can can explicitly designate the data as raster images via Image[] if you prefer do your processing in the image space. Staying in 'data' space is typically easier. Compare finding the mean Intensity:

Data space (One command):

```
In[ 1]:= N[Mean[bc]]
Out[1]:= 841.535
```

Raster Image space (Many commands):

```
In[ 2]:=    means={};
            TXY = ArrayReshape[bc, {100, 209, 229}];
            For[i=1, i<=100, i++,
                  imi = Image[TXY[[i, All, All]]];
                  mv  = ImageMeasurements[imi, "MeanIntensity"];
                  AppendTo[means,mv];
            ]
            Mean[means]

Out[2]:= 841.535
```

Note that you can write this more compactly by (falsely) representing your XYT information as a 3D (XYZ) image:

```
In [3]:=
ImageMeasurements[Image3D[TXY],
"MeanIntensity"]

Out[3]:= 841.535
```

Which is also nice because that way you can visualize your single molecule data as a kymogram:

```
In [4]:=
ImageAdjust[Image3D[TXY], 0.1,
{1300, 10000}]

Out[4]:=
```