How do ColorModels and WritableRasters work in java BufferedImages?

Asked 6 years, 6 months ago Active 6 years, 5 months ago Viewed 1k times

1

2

When working with the `BufferedImage` class in Java, I usually use the constructor with parameters `int width, int height, int type`. For a certain application, though, I wanted an image which would store the color data using bytes in the order ARGB, which can't be done in that way (it has only `TYPE_4BYTE_ABGR`).

I found the following solution, which worked fine:

```
    WritableRaster raster = Raster.createInterleavedRaster(DataBuffer.TYPE_BYTE, width,
height, 4, null);
    ColorModel colorModel = new
ComponentColorModel(ColorSpace.getInstance(ColorSpace.CS_sRGB), new int[]{8,8,8,8},
true, false, ColorModel.TRANSLUCENT, DataBuffer.TYPE_BYTE);
    img = new BufferedImage(colorModel, raster, false, new Hashtable<>());
```

I don't understand why this works?

Though - I understand that the `WritableRaster` is the data structure that holds the pixel data of the picture, but past that I am lost. Which of these two objects - the Raster, or the ColorModel - determines that the pixel data is in the order RGBA? And how could I simulate any of the types in BufferedImage's `(int, int, int)` constructor using the `(ColorModel, WritableRaster, boolean, HashTable)` constructor?

java image bufferedimage

edited Jul 9 '14 at 11:00
gprathour
12.6k 4 55 80

asked Jul 8 '14 at 19:16
Carmeister
198 1 7

1 Answer

Active Oldest Votes

4

It's the method

```
Raster.createInterleavedRaster(DataBuffer.TYPE_BYTE, width, height, 4, null);
```

...that specifies the byte order. It does so implicitly, by assuming for 4 bands, you want the band offsets to be `0, 1, 2, 3` (which corresponds to RGBA; see the source for details). For RGB color space, band 0 = Red, 1 = Green, 2 = Blue and 3 = Alpha.

If you wanted a different order, you could have used a different factory method, for instance to create a raster with ARGB order:

```
Raster.createInterleavedRaster(DataBuffer.TYPE_BYTE, width, height,
                               width * 4, 4, new int[] {3, 0, 1, 2}, null);
```

Both of these methods will create an instance of `PixelInterleavedSampleModel` for you, and it's this `SampleModel` that really controls the sample order.

For how the `BufferedImage(int, int, int)` constructor works, and how you could do similar things, I think the best would be to just look at the source code for yourself. It's basically one big `switch` statement, where for each constant `TYPE_*` it creates a `WritableRaster` and a `ColorModel` similar to how you do it above.

For example:

```
ColorModel colorModel = ColorModel.getRGBdefault();
WritableRaster raster = colorModel.createCompatibleWritableRaster(width, height);

new BufferedImage(colorModel, raster, colorModel.isAlpahPremultiplied(), null);
```

...will create an image with type `TYPE_INT_ARGB` (the way this reverse lookup actually works is somewhat nasty, but it works... :-)). If no corresponding type exists in `BufferedImage` the type will be `TYPE_CUSTOM` (`0`).

edited Jul 9 '14 at 13:41

answered Jul 9 '14 at 9:31
haraldK
23.2k 6 46 94

Does this exists for JavaFX? – Displee Feb 17 '17 at 14:26

@Displee While you can use the above with the `SwingFXUtils` class to convert between a JavaFX `WriteableImage` and Java2D `BufferedImage` , JavaFX uses a completely different API. See WriteableImage and the PixelReader class. Oh, and.. Please don't ask new, unrelated questions in comments. :-) – haraldK Feb 17 '17 at 14:32