ColorHandling

Color Handling

Introduction

This page shall discuss an approach to represent the various colors supported by upcoming SVG and XSL-FO specifications in Java.

Requirements

Both SVG and XSL-FO allow specification of color in various color spaces (defined by ICC color profiles, with 1, 3, 4 or n components). In both cases, sRGB is the default color space. sRGB also serves as fallback in both cases. With SVG 1.2 and XSL 2.0, additional color specifications will be introduced. Among these are:

- device-specific (uncalibrated) colors, mainly for specifying CMYK colors, but also for gray, RGB values and N-Component colorspaces (like HP's I ndiChrome 6-color printing).
- CIE Lab and CIE LCHab colors
- ICC Named Color (for spot colors)

Not all possible output formats for SVG and XSL-FO documents support all of the above color specifications. In all cases, fallback colors in sRGB color space are specified to achieve at least an approximation of the intended color. What's intended here is to find a way to represent these colors as primary color specifications for output formats that support them. In all other case, the sRGB fallback shall be used. There are a few more details in the SVG Color spec:

- For calibrated colors (ICC color, ICC named color or CIE Lab/LCHab), the spec says that the user agent shall use these and convert to other color spaces (for interpolation or compositing, for example) using the specific color values rather than the sRGB fallback.
- For device-specific colors, the color values are to be passed through to the output device but interpolation and compositing shall occur using the sRGB fallback.

Although not clearly specified, yet, it can be assumed that XSL-FO 2.0 will handle this in a similar way. XSL 1.1, at any rate, clearly says that the sRGB fallback is only used for ICC colors, if the associated color profile is unavailable.

So, we're presented with essentially two different way of color handling: Use the calibrated color if supported and the calibration info (color profile) is available. Device-specific colors are only used in the output format if it is supported and not involved in any color calculations. The sRGB fallback has a more important role in the latter case.

FOP-specific requirements

Apache FOP has two XML-based intermediate formats which makes it necessary to generate textual representations from the decoded original color specification. So there is need for a round-trip possibility. That means that we can't simply use the ColorSpace instance but also have to carry a name and a URI for the color profile/space.

Color in the Java class library

`java.awt.Color` is Java's way to specify color, primarily in the sRGB color space. But it also has constructors taking a [ColorSpace] parameter, so arbitrary colors with n components and color values between 0.0 and 1.0 can be specified. The class uses the `float[] [ColorSpace].toRGB(float[])` method to calculate the sRGB fallback value for such a color.

What the class doesn't support is the specification of named colors based on an ICC named color profile (or any other source for named colors). A possible work-around is thinkable: A named color profile is split up into its various named colors and a ColorSpace object is created for each of them with one component each for the tint (see notes below).

`java.awt.Color`'s `equals()` method has a little problem: it doesn't detect the difference between two different colors using the same sRGB fallback. Example: if there is a plain `Color(255, 204, 0)` and a Color descendant with additional color information but with the same sRGB values, they may be regarded as equal although they are not.

Current state (2010-06-15)

A `ColorExt` class was moved from FOP to XGC. It has its own set of RGB replacement values and fields for the name and URI of a color profile. Some code in there seems not to be used anywhere and other code appears to duplicate functionality of the superclass (`Color`). The `ColorExt` class is not sufficient to carry named colors, for example.

Apparently, the replacement values in `ColorExt` were introduced to have access to the original fallback values, since `Color` calculates the sRGB values using the `ColorSpace` which may not be the same value as the fallback. And the superclass doesn't allow write access to the sRGB values through the ICC-based constructor.

The member variable `colorValues` of `ColorExt` saves the same values as `fvalue` in the super class, so this seems to be superfluous.

Batik currently only supports sRGB colors and ICC colors with a backing (and loaded) ICC profile. No access is possible to the original ICC profile file (only the loaded data is available) since plain Java classes are used. However, Batik has a class ICCColorSpaceExt which can hold a rendering intent override (an ICC color profile has a rendering intent value, too) and calculates sRGB values by selected rendering intent. That could be useful to FOP, too, so the functionality is also a candidate to be ported to XGC. One current problem in Batik is that ICC colors are converted to sRGB on the input side, rather than the output side (information is lost even when a conversion to sRGB is not necessary or not desired). This will have to be changed so the specific color is preserved as long as possible. Fortunately, the `Color` class can always return the corresponding sRGB value when needed.

Current state (2010-07-02)

With the XGC 1.4 release, the ColorExt class has been moved back to FOP pending the color redesign discussed on this page.

Current state (2010-10-28)

The new code has been integrated in color branches for Batik and FOP and running in production since August 2010 and is working fine. A vote has been called to merge the color branch into Trunk.

Development branches

- https://svn.apache.org/repos/asf/xmlgraphics/commons/branches/Temp_Color
- https://svn.apache.org/repos/asf/xmlgraphics/batik/branches/svgcolor12
- https://svn.apache.org/repos/asf/xmlgraphics/fop/branches/Temp_Color

Ideas

It makes sense to create multiple subclasses of the `Color` class for the various color specifications, but:

Right now, SVG and XSL define an optional "real" color and a mandatory sRGB fallback. XSL uses single functions which include the sRGB fallback as part of the function. SVG, however, specifies the sRGB fallback in front and the color function after that. The latter has greater flexibility since it would allow to extend the paint specification so it can specify more than 2 colors as a prioritized list. The implementation could then choose the best color that the selected output format supports. Example: PDF supports to specify and use multiple calibrated color spaces. When generating a bitmap, only one color space may be used which means that colors not matching that color space have to be converted via CIE XYZ or CIE LAB color spaces. However, color conversions can be lossy. But this is might go too far. Still, implementations will have to deal with certain color conversions if they don't always want to fallback to the sRGB values since that has a smaller gamut than other color spaces and is using additive colors rather than subtractive ones better suited for printing.

We have to differentiate two cases:

1. For **calibrated colors**, the specific color could be used as the primary color (since the sRGB color is only to be used if a color profile is unavailable). Additional color specifications could be attached to the ordered list of alternative colors mentioned above.
2. For **device-specific colors**, the sRGB fallback is the primary color. The alternative is only used if the output format supports it. All color calculations are made off the sRGB color.

Since Java has the infrastructure for color management, we can always use the specific calibrated color if we can support it using the right `ColorSpace` subclasses. Such a color instance would be specified using that `ColorSpace` and be the primary color. Whenever an sRGB value is required as a fallback somewhere, it can always be calculated using its color space. The explicit sRGB fallback is essentially ignored in this case.

For device-specific colors, we work with the sRGB value as primary color and can attach the device-specific color to the primary color. Any output format supporting device-specific colors can choose the the attached color instead of the sRGB primary.

`ColorExt` from FOP should be retired in favor of a clearer and simpler `Color` hierarchy. A `ColorWithAlternatives` class can be used to attach device-specific colors.

FOP needs to regenerate the string color functions for its intermediate formats from the actual colors. For this purpose, `ColorWithAlternatives` is further subclassed to include the fallback sRGB color (not as part of the alternatives). Furthermore, the color profile name and URI formerly carried in `ColorExt` would be included with the color space, accessed by an interface called `ColorSpaceOrigin`. `ICC_ColorSpace` is subclassed for this purpose. The `ICCColorSpaceExt` class from Batik is used as base.

For device-specific colors we'd create a number of `ColorSpace` subclasses derived from an abstract base class indicating the device-specific nature of the color space. The color conversion routines would not be fully implemented in this case, because especially for N-Channel color spaces, we have to way to calculate an equivalent sRGB color. For CIE Lab colors, a color space class can be created that can operate fully in a color-managed way.

For each named color, we'd have to provide a new `ColorSpace` instance with just one component for the tint, so it will have to have type `ColorSpace.TYPE_GRAY` which is the only one with just one component. Maybe the alpha value could actually be used instead but that's not sure. The tricky thing here is probably to calculate the right XYZ coordinate for the various tint values given a specific white point if the color shall be used directly by the Java2D subsystem which is not necessary for just embedding the color spec in the output format. Once support for named color profiles is available, those special color spaces could be linked together with a common parent representing the actual named color profile.

A sketch given the above:

```
public class ColorWithAlternatives extends Color {

    private List<Color> alternativeColors;

[..]
} //in XGC

public class ColorWithFallback extends ColorWithAlternatives {

    private final Color fallback;

[..]
}

public class ICCColorSpaceExt extends ICC_ColorSpace implements ColorSpaceOrigin...
public class DeviceCMYKColorSpace extends AbstractDeviceSpecificColorSpace.... //4 components, TYPE_CMYK
public class DeviceRGBColorSpace extends AbstractDeviceSpecificColorSpace.... //3 components, TYPE_RGB
public class DeviceGrayColorSpace extends AbstractDeviceSpecificColorSpace.... //1 component, TYPE_GRAY
public class NamedColorSpace extends ColorSpace implements ColorSpaceOrigin.... //1 component, TYPE_GRAY
public class CIELabColorSpace extends ColorSpace .... //3 components, TYPE_LAB
public class CIELCHabColorSpace extends ColorSpace .... //3 components, TYPE_LAB
```

Various notes

ICC Named Color Profiles

The ICC specification provides a way to specify various named spot colors in a color profile. However, this facet is not very well supported by consumers as well as producers of ICC color profiles. So while the use of ICC named colors is not a bad idea per se, without the right tooling, it is rather difficult to handle. Furthermore, many print shops define named colors directly on the RIP/printer. PDF and PostScript allow to specify a "Separation" color space with an abstract named color which is only mapped to the final spot color on the RIP (raster image processor). In this case an ICC named color profile is not involved. This may make it necessary to define pseudo (or virtual) named color profiles which take their color specs from somewhere else (like a proprietary XML file). Furthermore, PDF and PostScript don't allow using an ICC named color profile.

Using named/spot colors

In offset printing spot colors are special inks that are premixed to achieve an exact color instead of having to mix the right color from 4 or more subtractive colors. That way it is, for example, possible to put a layer of gold "ink" on paper.

Often, named (or spot) colors are accompanied with a single tint value that specifies the amount of coverage with the spot color. 0.0 uses no ink while 1.0 specifies the maximum ink usage (complete coverage). For example, PDF's separation color space supports that. It is however interesting that at the moment the SVG and XSL working drafts don't provide means to specify tints on named colors.

White Points

Most color conversions involve a white point (and sometimes a black point). Currently, it is not clear from the SVG and XSL-FO WDs which white points shall be used (especially with CIE Lab colors). We currently assume D65 as white point on the input side where no color profile tells us otherwise. The current opinion of the SVG WG is to use D65 in SVG Color 1.2.

Color.equals() method

The problem of the `Color.equals()` method outlined above cannot be solved by simply providing a better equals() on a `Color` subclass because that violates the contract established in the runtime Javadocs. We have several possibilities:

- Create a method in `ColorUtil` that compares two colors.
- Establish a new method (ex. `isSameColor(Color)`) which has an extended contract to compared two colors.
- Always compare two colors in both directions (currently implemented like this). This solution might be sub-optimal concerning performance and still essentially violates the basic contract of `Color.equals()`.

Spec references

- SVG 1.1: http://www.w3.org/TR/SVG11/painting.html#SpecifyingPaint
- SVG Color 1.2 (draft): http://www.w3.org/TR/SVGColor12/#Color_syntax
- XSL 1.1: http://www.w3.org/TR/xsl11/#expr-color-functions
- XSL 2.0 (draft): http://www.w3.org/TR/2010/WD-xslfo20-20100204/#d2e3930
- ICC 4.2: http://color.org (by International Color Consortium)

Color-related information sources

- CIE XYZ color space: http://en.wikipedia.org/wiki/Lab_color_space
- Lab color space: http://en.wikipedia.org/wiki/Lab_color_space
- D65 standard illuminant: http://en.wikipedia.org/wiki/D65
- Good article on color space conversions: http://www.cambridgeincolour.com/tutorials/color-space-conversion.htm
- Color Wiki: http://www.colorwiki.com
- Color Swatch Formats: http://www.selapa.net/couleurs/fileformats.php
- Color Swatch Tool (SwatchBooker): http://www.selapa.net/swatchbooker/
- Color Converter by Bruce Lindbloom: http://www.brucelindbloom.com/index.html?ColorCalculator.html
- Another color converter: http://www.colorpro.com/info/tools/convert.htm
- Named color thread on www-svg: http://lists.w3.org/Archives/Public/www-svg/2010Jun/0109.html
- Lab color thread on www-svg: http://lists.w3.org/Archives/Public/www-svg/2010Jun/0168.html