Cloud Vision API Product Search

Color

Represents a color in the RGBA color space. This representation is designed for simplicity of conversion to/from color representations in various languages over compactness; for example, the fields of this representation can be trivially provided to the constructor of "java.awt.Color" in Java; it can also be trivially provided to UIColor's "+colorWithRed:green:blue:alpha" method in iOS; and, with just a little work, it can be easily formatted into a CSS "rgba()" string in JavaScript, as well.

Note: this proto does not carry information about the absolute color space that should be used to interpret the RGB value (e.g. sRGB, Adobe RGB, DCI-P3, BT.2020, etc.). By default, applications SHOULD assume the sRGB color space.

Example (Java):

```
import com.google.type.Color;

// ...
public static java.awt.Color fromProto(Color protocolor) {
  float alpha = protocolor.hasAlpha()
      ? protocolor.getAlpha().getValue()
      : 1.0;

  return new java.awt.Color(
      protocolor.getRed(),
      protocolor.getGreen(),
      protocolor.getBlue(),
      alpha);
}

public static Color toProto(java.awt.Color color) {
  float red = (float) color.getRed();
  float green = (float) color.getGreen();
  float blue = (float) color.getBlue();
  float denominator = 255.0;
  Color.Builder resultBuilder =
      Color
          .newBuilder()
          .setRed(red / denominator)
          .setGreen(green / denominator)
          .setBlue(blue / denominator);
```

```
  int alpha = color.getAlpha();
  if (alpha != 255) {
    result.setAlpha(
        FloatValue
            .newBuilder()
            .setValue(((float) alpha) / denominator)
            .build());
  }
  return resultBuilder.build();
}
// ...
```

Example (iOS / Obj-C):

```
// ...
static UIColor* fromProto(Color* protocolor) {
   float red = [protocolor red];
   float green = [protocolor green];
   float blue = [protocolor blue];
   FloatValue* alpha_wrapper = [protocolor alpha];
   float alpha = 1.0;
   if (alpha_wrapper != nil) {
     alpha = [alpha_wrapper value];
   }
   return [UIColor colorWithRed:red green:green blue:blue alpha:alpha];
}

static Color* toProto(UIColor* color) {
    CGFloat red, green, blue, alpha;
    if (![color getRed:&red green:&green blue:&blue alpha:&alpha]) {
      return nil;
    }
    Color* result = [[Color alloc] init];
    [result setRed:red];
    [result setGreen:green];
    [result setBlue:blue];
    if (alpha <= 0.9999) {
      [result setAlpha:floatWrapperWithValue(alpha)];
    }
    [result autorelease];
    return result;
}
// ...
```

Example (JavaScript):

```
// ...

var protoToCssColor = function(rgb_color) {
   var redFrac = rgb_color.red || 0.0;
   var greenFrac = rgb_color.green || 0.0;
   var blueFrac = rgb_color.blue || 0.0;
   var red = Math.floor(redFrac * 255);
   var green = Math.floor(greenFrac * 255);
   var blue = Math.floor(blueFrac * 255);

   if (!('alpha' in rgb_color)) {
      return rgbToCssColor_(red, green, blue);
   }

   var alphaFrac = rgb_color.alpha.value || 0.0;
   var rgbParams = [red, green, blue].join(',');
   return ['rgba(', rgbParams, ',', alphaFrac, ')'].join('');
};

var rgbToCssColor_ = function(red, green, blue) {
  var rgbNumber = new Number((red << 16) | (green << 8) | blue);
  var hexString = rgbNumber.toString(16);
  var missingZeros = 6 - hexString.length;
  var resultBuilder = ['#'];
  for (var i = 0; i < missingZeros; i++) {
     resultBuilder.push('0');
  }
  resultBuilder.push(hexString);
  return resultBuilder.join('');
};

// ...
```

JSON representation

```
{
  "red": number,
  "green": number,
  "blue": number,
  "alpha": number
}
```

Fields	
`red`	`number` The amount of red in the color as a value in the interval [0, 1].
`green`	`number` The amount of green in the color as a value in the interval [0, 1].
`blue`	`number` The amount of blue in the color as a value in the interval [0, 1].
`alpha`	`number` The fraction of this color that should be applied to the pixel. That is, the final pixel color is defined by the equation: pixel color = alpha * (this color) + (1.0 - alpha) * (background color) This means that a value of 1.0 corresponds to a solid color, whereas a value of 0.0 corresponds to a completely transparent color. This uses a wrapper message rather than a simple float scalar so that it is possible to distinguish between a default value and the value being unset. If omitted, this color object is to be rendered as a solid color (as if the alpha value had been explicitly given with a value of 1.0).