A color converter class

```
//http://calcsharp.codeplex.com/license
//Microsoft Public License (Ms-PL)
using System;
using System.Drawing;
using System.Text;
using System.Drawing.Imaging;

namespace CalcSharp.Core
{
///

/// A color converter class
///
public class Int24
{
private byte r;
private byte g;
private byte b;

private static void TestArrayLength(byte[] array, int l)
{
if (array.Length != l) throw new Exception(“Not enough data”);
}

public static int RGB2Int(byte R, byte G, byte B)
{
int temp = R < 255) this.r = 255;
else this.r = (byte)R;

if (G < 0) this.g = 0; else if (G > 255) this.g = 255;
else this.g = (byte)R;

if (B < 0) this.b = 0; else if (B > 255) this.b = 255;
else this.b = (byte)R;
}
```

```
public Int24(Color c)
{
this.FromColor(c);
}
#endregion

#region Operators & Overrides

public static implicit operator Int24(int input)
{
return new Int24(input);
}

public static implicit operator int(Int24 input)
{
return input.ToInt32();
}

public static Int24 operator +(Int24 i1, Int24 i2)
{
Int24 ret = new Int24();
int r, g, b;
r = i1.R + i2.R;
g = i1.G + i2.G;
b = i1.B + i2.B;
if (r > 255) r = 255;
if (r < 0) r = 0; if (g > 255) g = 255;
if (g < 0) g = 0; if (b > 255) b = 255;
if (b < 0) b = 0; ret.R = (byte)r; ret.G = (byte)g; ret.B =
(byte)b; return ret; } public static Int24 operator +(Int24 i,
int n) { Int24 ret = new Int24(); int r, g, b; r = i.R + n; g
= i.G + n; b = i.B + n; if (r > 255) r = 255;
if (r < 0) r = 0; if (g > 255) g = 255;
if (g < 0) g = 0; if (b > 255) b = 255;
if (b < 0) b = 0; ret.R = (byte)r; ret.G = (byte)g; ret.B =
(byte)b; return ret; } public static Int24 operator -(Int24
i1, Int24 i2) { Int24 ret = new Int24(); int r, g, b; r = i1.R
- i2.R; g = i1.G - i2.G; b = i1.B - i2.B; if (r > 255) r =
```

```
255;
if (r < 0) r = 0; if (g > 255) g = 255;
if (g < 0) g = 0; if (b > 255) b = 255;
if (b < 0) b = 0; ret.R = (byte)r; ret.G = (byte)g; ret.B =
(byte)b; return ret; } public static Int24 operator -(Int24 i,
int n) { Int24 ret = new Int24(); int r, g, b; r = i.R - n; g
= i.G - n; b = i.B - n; if (r > 255) r = 255;
if (r < 0) r = 0; if (g > 255) g = 255;
if (g < 0) g = 0; if (b > 255) b = 255;
if (b < 0) b = 0; ret.R = (byte)r; ret.G = (byte)g; ret.B =
(byte)b; return ret; } public static Int24 operator *(Int24
i1, Int24 i2) { Int24 ret = new Int24(); int r, g, b; r = i1.R
* i2.R; g = i1.G * i2.G; b = i1.B * i2.B; if (r > 255) r =
255;
if (r < 0) r = 0; if (g > 255) g = 255;
if (g < 0) g = 0; if (b > 255) b = 255;
if (b < 0) b = 0; ret.R = (byte)r; ret.G = (byte)g; ret.B =
(byte)b; return ret; } public static Int24 operator *(Int24 i,
int n) { Int24 ret = new Int24(); int r, g, b; r = i.R * n; g
= i.G * n; b = i.B * n; if (r > 255) r = 255;
if (r < 0) r = 0; if (g > 255) g = 255;
if (g < 0) g = 0; if (b > 255) b = 255;
if (b < 0) b = 0; ret.R = (byte)r; ret.G = (byte)g; ret.B =
(byte)b; return ret; } public static Int24 operator *(Int24 i,
short n) { Int24 ret = new Int24(); int r, g, b; r = i.R * n;
g = i.G * n; b = i.B * n; if (r > 255) r = 255;
if (r < 0) r = 0; if (g > 255) g = 255;
if (g < 0) g = 0; if (b > 255) b = 255;
if (b < 0) b = 0; ret.R = (byte)r; ret.G = (byte)g; ret.B =
(byte)b; return ret; } public static Int24 operator /(Int24
i1, Int24 i2) { Int24 ret = new Int24(); int r, g, b; r = i1.R
/ i2.R; g = i1.G / i2.G; b = i1.B / i2.B; if (r > 255) r =
255;
if (r < 0) r = 0; if (g > 255) g = 255;
if (g < 0) g = 0; if (b > 255) b = 255;
if (b < 0) b = 0; ret.R = (byte)r; ret.G = (byte)g; ret.B =
(byte)b; return ret; } public static Int24 operator /(Int24 i,
```

```
int n) { Int24 ret = new Int24(); int r, g, b; r = i.R / n; g
= i.G / n; b = i.B / n; if (r > 255) r = 255;
if (r < 0) r = 0; if (g > 255) g = 255;
if (g < 0) g = 0; if (b > 255) b = 255;
if (b < 0) b = 0; ret.R = (byte)r; ret.G = (byte)g; ret.B =
(byte)b; return ret; } public static bool operator ==(Int24 a,
Int24 b) { return (a.R == b.R && a.G == b.G && a.B == b.B); }
public static bool operator !=(Int24 a, Int24 b) { return (a.R
!= b.R || a.G != b.G || a.B != b.B); } public override bool
Equals(object obj) { if (obj == null || GetType() !=
obj.GetType()) return false; return (this == (Int24)obj); }
public override int GetHashCode() { return R.GetHashCode() ^
G.GetHashCode() ^ B.GetHashCode(); } #endregion #region From
public void FromInt32(int num) { int w = num; int temp; if (w
< 0) w = Math.Abs(w); if (w > 0xFFFFFF) w = 0xFFFFFF;

temp = w & 0xFF0000;
temp = temp >> 16;
this.r = Convert.ToByte(temp);

temp = w & 0x00FF00;
temp = temp >> 8;
this.g = Convert.ToByte(temp);

temp = w & 0x0000FF;
this.b = Convert.ToByte(temp);
}

public void FromColor(Color c)
{
this.r = c.R;
this.b = c.B;
this.g = c.G;
}

public void FromByteArray(byte[] array, PixelFormat f)
{
ushort data;
```

```
switch (f)
{
case PixelFormat.Format16bppArgb1555:
case PixelFormat.Format16bppRgb555:
TestArrayLength(array, 2);
data = BitConverter.ToUInt16(array, 0);
this.r = (byte)((data & 0x7C00) >> 10);
this.g = (byte)((data & 0x3E0) >> 5);
this.b = (byte)((data & 0x1F));
break;
case PixelFormat.Format16bppRgb565:
TestArrayLength(array, 2);
data = BitConverter.ToUInt16(array, 0);
this.r = (byte)((data & 0xF800) >> 11);
this.g = (byte)((data & 0x7E0) >> 6);
this.b = (byte)(data & 0x1F);
break;
case PixelFormat.Format24bppRgb:
TestArrayLength(array, 3);
this.r = array[0];
this.g = array[1];
this.b = array[2];
break;
case PixelFormat.Format32bppArgb:
case PixelFormat.Format32bppRgb:
TestArrayLength(array, 4);
this.r = array[1];
this.g = array[2];
this.b = array[3];
break;
case PixelFormat.Format32bppPArgb:
TestArrayLength(array, 4);
byte alpha = array[0];
this.r = (byte)(array[1] / alpha);
this.g = (byte)(array[2] / alpha);
this.b = (byte)(array[3] / alpha);
break;
```

```
default:
throw new Exception(“Unsuported pixel format”);
}
}

public void FromByteArray(byte[] array)
{
byte backup;
Array.Reverse(array);
ushort b2, b;
switch (array.Length)
{
case 1:
backup = (byte)(array[0] & 192);
this.r = (byte)(backup >> 5);
backup = (byte)(array[0] & 28);
this.g = (byte)(backup >> 2);
this.b = (byte)(array[0] & 3);
break;
case 2:
//5650 format
b = BitConverter.ToUInt16(array, 0);
b2 = (ushort)(b & 0xf800);
this.r = (byte)(b2 >> 11);
b2 = (ushort)(b & 0x07e0);
this.g = (byte)(b2 >> 5);
this.b = (byte)(b & 0x001f);
break;
case 3:
this.r = array[0];
this.g = array[1];
this.b = array[2];
break;
case 4:
this.r = array[1];
this.g = array[2];
this.b = array[3];
```

```
break;
}
}

public void FromString(string val)
{
string[] raw = val.Split(';');
if (raw.Length != 3) throw new Exception(“Can't parse as HSL
value: ” + val);
raw[0] = raw[0].Replace(“R: “, “”);
raw[1] = raw[1].Replace(” G: “, “”);
raw[2] = raw[2].Replace(” B: “, “”);
this.R = Convert.ToByte(raw[0]);
this.G = Convert.ToByte(raw[1]);
this.B = Convert.ToByte(raw[2]);
}
#endregion

#region To
public int ToInt32()
{
int temp = this.r <
```