The gradients are currently stored in the closed source for the SmartSDR for Windows client.  However, I'm happy to share the code snippets for the relevant pieces provided that the code's intent is for use on FlexRadio hardware.  Let me know if you have any questions.  Here are the snippets:

private void GenerateAllWaterfallGradients()

        {

            // the default gradient

            waterfallGradientBasic.colorList.Clear();

            waterfallGradientBasic.colorList.Add(new GradientColorStop(Colors.Black, 0.0));

            waterfallGradientBasic.colorList.Add(new GradientColorStop(Colors.Blue, 0.15)); // 0.15

            waterfallGradientBasic.colorList.Add(new GradientColorStop(Colors.Cyan, 0.25)); // 0.10

            waterfallGradientBasic.colorList.Add(new GradientColorStop(Color.FromRgb(0, 255, 0), 0.35)); // 0.10

            waterfallGradientBasic.colorList.Add(new GradientColorStop(Colors.Yellow, 0.55)); // 0.20

            waterfallGradientBasic.colorList.Add(new GradientColorStop(Colors.Red, 0.90)); // 0.35

            waterfallGradientBasic.colorList.Add(new GradientColorStop(Colors.White, 1.0)); // 0.10

            // Abed's choice of colors

            waterfallGradientPurple.colorList.Clear();

            waterfallGradientPurple.colorList.Add(new GradientColorStop(Colors.Black, 0.0));

            waterfallGradientPurple.colorList.Add(new GradientColorStop(Colors.Blue, 0.15));

            waterfallGradientPurple.colorList.Add(new GradientColorStop(Color.FromRgb(0, 255, 0), 0.30));

            waterfallGradientPurple.colorList.Add(new GradientColorStop(Colors.Yellow, 0.45));

            waterfallGradientPurple.colorList.Add(new GradientColorStop(Colors.Red, 0.60));

            waterfallGradientPurple.colorList.Add(new GradientColorStop(Colors.Purple, 0.75));

            waterfallGradientPurple.colorList.Add(new GradientColorStop(Colors.White, 1.0));

            // Dark

            waterfallGradientDark.colorList.Clear();

            waterfallGradientDark.colorList.Add(new GradientColorStop(Colors.Black, 0.0));

            waterfallGradientDark.colorList.Add(new GradientColorStop(Colors.Blue, 0.65));

            waterfallGradientDark.colorList.Add(new GradientColorStop(Color.FromRgb(0, 255, 0), 0.90));

            waterfallGradientDark.colorList.Add(new GradientColorStop(Colors.Red, 0.95));

            waterfallGradientDark.colorList.Add(new GradientColorStop(Colors.LightPink, 1.0));

            waterfallGradientDark.colorList.Add(new GradientColorStop(Colors.LightPink, 1.0));

            waterfallGradientDark.colorList.Add(new GradientColorStop(Colors.LightPink, 1.0));

            // No colors allowed

            waterfallGradientGrayscale.colorList.Clear();

            waterfallGradientGrayscale.colorList.Add(new GradientColorStop(Colors.Black, 0.0));

            waterfallGradientGrayscale.colorList.Add(new GradientColorStop(Colors.White, 1.00));

            waterfallGradientGrayscale.colorList.Add(new GradientColorStop(Colors.White, 1.00));

            waterfallGradientGrayscale.colorList.Add(new GradientColorStop(Colors.White, 1.00));

            waterfallGradientGrayscale.colorList.Add(new GradientColorStop(Colors.White, 1.00));

            waterfallGradientGrayscale.colorList.Add(new GradientColorStop(Colors.White, 1.00));

            waterfallGradientGrayscale.colorList.Add(new GradientColorStop(Colors.White, 1.00));

            // Deuteranopia

            waterfallGradientDeuteranopia.colorList.Clear();

            waterfallGradientDeuteranopia.colorList.Add(new GradientColorStop(Colors.Black, 0.0));                  // Black

            waterfallGradientDeuteranopia.colorList.Add(new GradientColorStop(Color.FromRgb(8, 60, 107), 0.15));     // Dark Blue

            waterfallGradientDeuteranopia.colorList.Add(new GradientColorStop(Color.FromRgb(132, 162, 214), 0.50)); // Light Blue

            waterfallGradientDeuteranopia.colorList.Add(new GradientColorStop(Color.FromRgb(165, 150, 115), 0.65));  // Dark Yellow

            //waterfallGradientDeuteranopia.colorList.Add(new GradientColorStop(Color.FromRgb(255, 219, 49), 0.80));  // Less Dark Yellow

            waterfallGradientDeuteranopia.colorList.Add(new GradientColorStop(Colors.Yellow, 0.75)); // Light Yellow

            waterfallGradientDeuteranopia.colorList.Add(new GradientColorStop(Colors.Yellow, 0.95));                 // White

            waterfallGradientDeuteranopia.colorList.Add(new GradientColorStop(Colors.White, 1.00));                 // White

            // Tritanopia

            waterfallGradientTritanopia.colorList.Clear();

            waterfallGradientTritanopia.colorList.Add(new GradientColorStop(Colors.Black, 0.0));

            waterfallGradientTritanopia.colorList.Add(new GradientColorStop(Color.FromRgb(0, 69, 82), 0.15)); // dark teal

            waterfallGradientTritanopia.colorList.Add(new GradientColorStop(Color.FromRgb(107, 186, 214), 0.45));   // light blue  rgb(107, 186, 214)

            waterfallGradientTritanopia.colorList.Add(new GradientColorStop(Color.FromRgb(74, 8, 24), 0.46));  // dark red rgb(74, 8, 24)

            waterfallGradientTritanopia.colorList.Add(new GradientColorStop(Colors.Red, 0.90));   // light red

            waterfallGradientTritanopia.colorList.Add(new GradientColorStop(Color.FromRgb(214, 121, 132), 0.99));   // light red rgb(214, 121, 132)

            waterfallGradientTritanopia.colorList.Add(new GradientColorStop(Colors.White, 1.00));   //white

=============================

private void GetWaterfallColor(ushort input, out byte red, out byte green, out byte blue)

        {

            ushort low, high;

            if (\_autoBlackLevelEnable)

            {

                low = \_fallAutoLowThreshold;

                high = \_fallAutoHighThreshold;

            }

            else

            {

                low = \_fallLowThreshold;

                high = \_fallHighThreshold;

            }

            if (\_updateWaterfallColors && \_waterfallGradientSelected != null)

            {

                // choose a specific color gradient to use

                // this needs to be called whenever the selected gradient has changed

                GenerateColorGradientArray(\_waterfallGradientSelected);

                \_updateWaterfallColors = false;

            }

            // figure out where input is relative to color dynamic range

            // is the input below the low threshold?

            if (input <= low)

            {

                // yes -- just use the low color value

                red = \_fallLowColor.R;

                green = \_fallLowColor.G;

                blue = \_fallLowColor.B;

            }

            // no -- is the input above the high threshold?

            else if (input >= high) // input is above range

            {

                // yes -- just use the high color value

                red = \_fallHighColor.R;

                green = \_fallHighColor.G;

                blue = \_fallHighColor.B;

            }

            else // no --  input is in the middle of the range

            {

                // We need to figure out where the input is as a percentage of the color space.

                // This the low and high thresholds into account

                float percent = (input - low) / (float)(high - low);

                // percentage to 0-65535

                ushort colorIndex = (ushort)((float)ushort.MaxValue \* percent);

                red = waterfallColorArray[colorIndex].R;

                green = waterfallColorArray[colorIndex].G;

                blue = waterfallColorArray[colorIndex].B;

            }

        }

====================================

private void UpdateAutoColorDynamicRange()

        {

            \_fallAutoHighThreshold = CalculateHighThreshold(\_fallAutoLowThreshold);

            // force a redraw since we have changed color parameters

            KickRenderWaterfall();

        }

==================================

private ushort CalculateHighThreshold(ushort low\_threshold)

        {

            ushort ret\_val = 0;

            // adjust high boundary from low + margin to max in X^3 pattern

            // move from 0-100 space into [1,cuberoot(2^16) space]

            double temp = (100 - \_fallColorGain) / 100.0 \* Math.Pow(ushort.MaxValue - low\_threshold, 1 / 3.0); // need a buffer between low and high??

            // now scale the value using the new value

            ret\_val = (ushort)(low\_threshold + Math.Pow(temp, 3.0));

            // Make sure that the \_fallHighThreshold is not lower than \_fallLowThreshold

            // Lets give a minimum allowed separation of 100

            if (ret\_val < low\_threshold + 100)

                ret\_val = (ushort)(low\_threshold + 100);

            return ret\_val;

        }

===================================

        private ushort CalculateLowThresholdFromBlackLevelSlider()

        {

            double val = 1.0 - (double)\_fallBlackLevel / 100.0; // map the 0-100 black level slider to an inverted 1.0-0.0

            double val2 = Math.Pow(val, 8); // remap the value to give extra dynamic range on the low end of the slider -- note that this leaves the values from 75-100 with no change

            return (ushort)(val2 \* (ushort.MaxValue - 10000)); // bring back into the 2^16 space (less the typical noise floor value)

        }