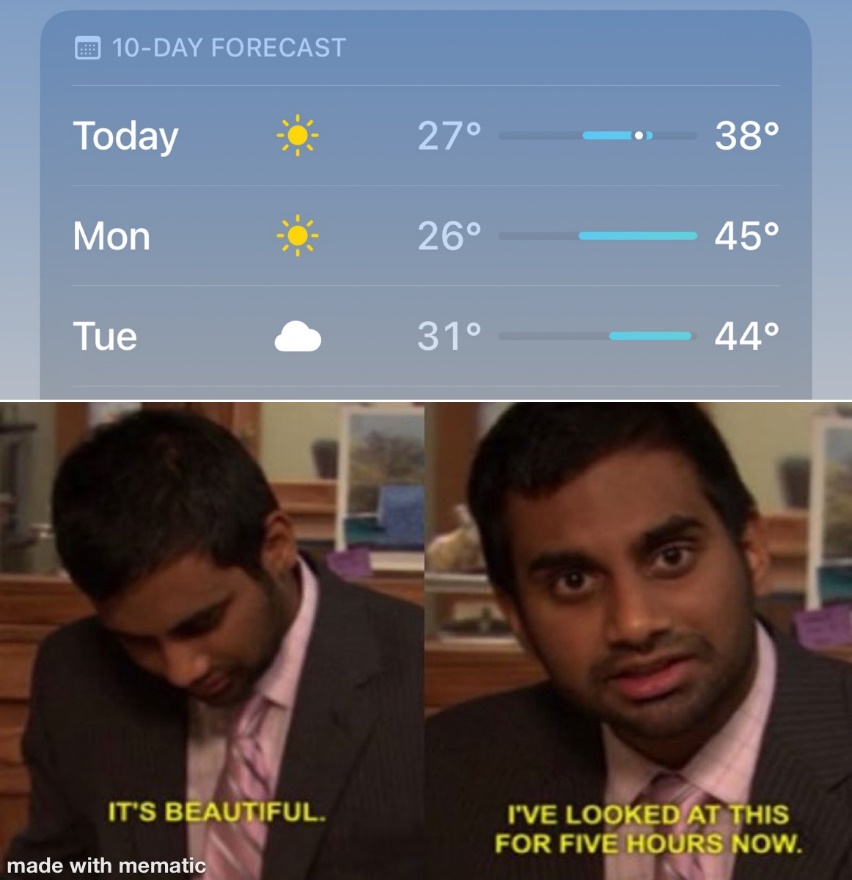
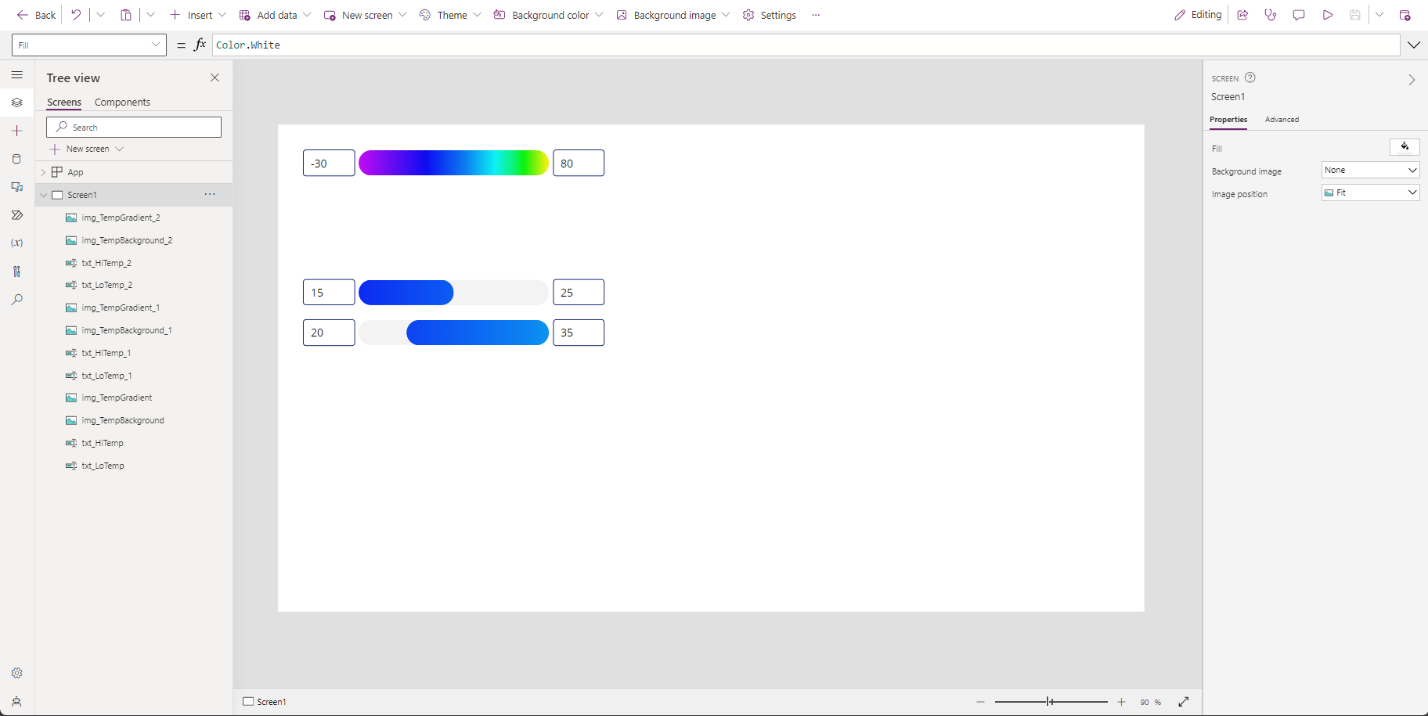
# Creating an SVG-based color band to represent temperatures in a Power App

One of my all-time favorite apps is the iOS weather app. The way that it communicates the current and upcoming weather conditions is nothing short of beautiful. While the “live weather” animations at the top are awesome, what I really like are the temperature gradient bars for the 10-day forecast.

A screenshot of a weather forecast

Description automatically generated

At this point, you are likely thinking (sarcastically), “Wow wyotim, with a favorite app like that you must be great fun at parties.” And you are not wrong; I’m not much fun at parties…but I am pretty okay-ish at creating a color band to represent temperatures, which is what this post will be about. 😁  
  
First, let’s look at what we are going to make.



We’ll start by looking at the SVG that makes it possible, then dive in to how we can create a single temperature band, and finish with how to create two temperature bands that scale relative to the two temperature ranges. Pretty exciting stuff, right? I bet you are rethinking that whole “fun at parties” thing now, right? No? That’s okay, you are still right. Anyway, let’s get started!

## The temperature band SVG

Let’s look at the SVG that will make all this possible.



To be fair, that is a raster image of the SVG, but I digress. What this represents is a temperature range of (from left to right) -100F to 200F, or for the more civilized types, -73.33C to 93.33C. This temperature range is on purpose as it gives a wide enough range to cover all the temperatures that a person will experience (hopefully! 😬), and it gave an even range of 300 to help me sort out how to make this thing work. The changes in colors are strategic as well, representing relative comfort ranges. I made this a while ago and didn’t take good notes on the source that I used for these ranges. I was unable to find it at the time of writing, but if I can track it down, I will cite it here later.

Here is the code for the SVG itself, in which you can see where the color breaks happen.

<svg width='600' height='100' viewBox='0 0 600 100' version='1.1' xmlns='http://www.w3.org/2000/svg'>

<defs>

<linearGradient id='TempGradient' x1='0' x2='1' y1='0' y2='0'>

<stop offset='0%' stop-color='hsl(300,90%,80%)'/>

<stop offset='20%' stop-color='hsl(300,90%,50%)'/>

<stop offset='28%' stop-color='hsl(270,90%,50%)'/>

<stop offset='36.33%' stop-color='hsl(240,90%,50%)'/>

<stop offset='44%' stop-color='hsl(210,90%,50%)'/>

<stop offset='49.67%' stop-color='hsl(180,90%,50%)'/>

<stop offset='53%' stop-color='hsl(150,90%,50%)'/>

<stop offset='55.22%' stop-color='hsl(120,90%,50%)'/>

<stop offset='57.44%' stop-color='hsl(90,90%,50%)'/>

<stop offset='59.67%' stop-color='hsl(60,90%,50%)'/>

<stop offset='63%' stop-color='hsl(30,90%,50%)'/>

<stop offset='66.67%' stop-color='hsl(0,90%,50%)'/>

<stop offset='71.33%' stop-color='hsl(0,90%,35%)'/>

<stop offset='100%' stop-color='hsl(0,90%,20%)'/>

</linearGradient>

</defs>

<rect x='0' y='0' width='600' height='100' fill='url(#TempGradient)' />

</svg>

Here are the temperature ranges that the color changes represent, in the same order that they appear in the SVG code:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Gradient Offset** | **Color** | **Low Temp, °F** | **High Temp, °F** | **Low Temp, °C** | **High Temp, °C** |
| 0% | Light pink | -100 | -41 | -73.33 | -41 |
| 20% | Pink | -40 | -17 | -40 | -28 |
| 28% | Purple | -16 | 8 | -27 | -14 |
| 36.33% | Blue | 9 | 31 | -13 | -1 |
| 44% | Light blue | 32 | 48 | 0 | 8 |
| 49.67% | Cyan | 49 | 58 | 9 | 14 |
| 53% | Seafoam green | 59 | 65 | 15 | 18 |
| 55.22% | Green | 66 | 71 | 19 | 21 |
| 57.44% | Yellow green | 72 | 78 | 22 | 25 |
| 59.67% | Yellow | 79 | 88 | 26 | 31 |
| 63% | Orange | 89 | 99 | 32 | 37 |
| 66.67% | Red | 100 | 113 | 38 | 45 |
| 71.33% | Dark red | 114 | 199 | 46 | 93.32 |
| 100% | Darker red | 200 | 200 | 93.33 | 93.33 |

One item of note: the color space I used is HSL, which stands for Hue, Saturation, and Lightness. This choice was, again, intentional as it allows three useful functions.

First, setting the colors (i.e. the Hues) in the gradient is super simple as it is based on degrees around the color wheel with red being at 0.

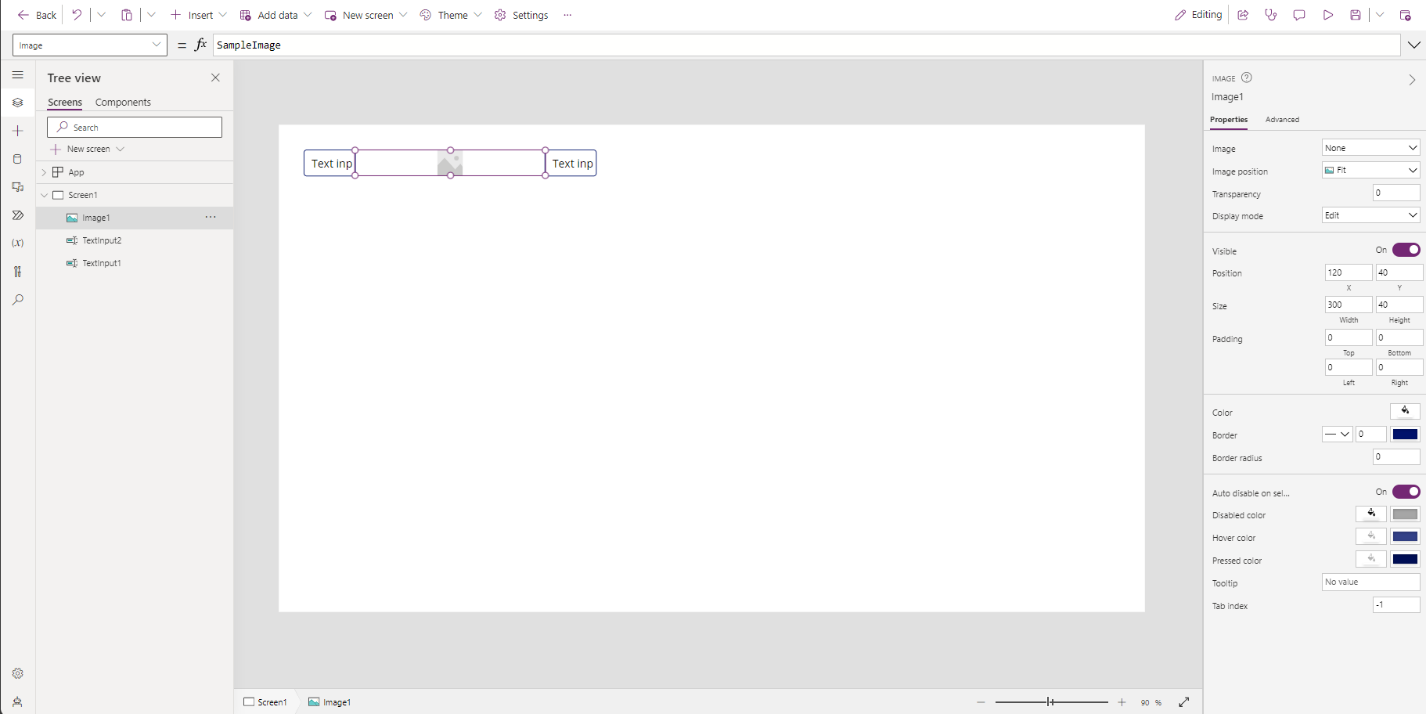
Second, it allows an easy way to add lighter or darker colors (i.e. the Lightness), which is done at the extremes of the color band. A pure color is at 50% in HSL, so you can set higher or lower values to darken or lighten the hue, respectively. A color space like HSB wouldn’t allow that as you really can only darken colors in the same hue due to the pure color being at 100% of the Brightness, so more complex adjustments would need to come in to play.

Third, it is possible to fit the color band to the existing color scheme of an app by adjusting the Saturation value to the same (or a similar value) as a color said scheme, like a primary color for instance. This means that the color band can have a defined structure that can make visual sense in the app it is being used in. I won’t cover this in this post, but I’m happy to write up how that would work in a later post if anyone were interested.

Now that we have a way to show off a range of temperatures, how do we make it show the temperature range we currently want to see? To do this, we need to define a space that the range will exist in. For a single temperature range, this is a trivial step: the space is just the size that you want the temperature range to exist in, such as the space between the high and low temperatures in the example picture. With two or more ranges, this gets a bit more complex as we probably want the temperature bands to line up so that we can compare the differences in the temperature extremes between the ranges and still have the colors line up properly so that the temperature color on one corresponds with the same temperature on the other(s). We’ll start with the single range.

## Single range gradient

We’ll begin by creating a new, blank Power App (either tablet or phone will work but I will be using tablet) and adding three controls: two text input controls and one image control. Size and arrange them like the image below.



Rename the left-most text input to txt\_LowTemp, the right-most text input to txt\_HiTemp, and the image control to img\_TempBackground. The exact size and spacing doesn’t matter too much; it’s the gradient that we are after. The sizes I used are 80 wide and 40 high for the text inputs and 300 wide and 40 high for the image.

Next, set the Format properties for both text input controls to Number. We will be using these to simulate the temperature inputs that we would get from a weather data source, such as the MSN Weather connector which is a standard connector and free to use in any Power App. If you want to enter some default numbers just to have them ready, feel free to do so. Just make sure that the number on the left side is smaller than the number on the right side; our temperature gradient goes from cold to hot in a left to right fashion. I’ll enter 15 on the left and 25 on the right.

For the image control, set each of the corner radius properties (RadiusBottomLeft, RadiusBottomRight, RadiusTopLeft, RadiusTopRight) to the following:

Self.Height / 2

This is the formula for circularly rounded corners.

We will also remove the SampleImage text from the Image property and leave it blank for now, then set the Fill property to:

RGBA(0,0,0,0.05)

We are going to use this image control as a background as it allows us to easily create a shape with rounded corners. We don’t need it for the single range version, but now we will have it to copy and paste for the two-range version we will create later.

Having done these steps, we should have something that looks like this:

A screenshot of a computer

Description automatically generated

Now copy and paste the image control, setting the Fill property to

RGBA(0,0,0,0)

and set the position properties (Height, Width, X, and Y) to match the first image control. Rename it to img\_TempGradient.

In the Image property, copy and paste the following:

"data:image/svg+xml," & EncodeUrl(

    "<svg

        width='" & Self.Width & "'

        height='" & Self.Height & "'

        viewBox='0 0 " & Self.Width & " " & Self.Height & "'

        version='1.1' xmlns='http://www.w3.org/2000/svg'

    >

        <defs>

            <linearGradient id='TempGradient' x1='0' x2='1' y1='0' y2='0'>

                <stop offset='0%' stop-color='hsl(300,90%,80%)'/>

                <stop offset='20%' stop-color='hsl(300,90%,50%)'/>

                <stop offset='28%' stop-color='hsl(270,90%,50%)'/>

                <stop offset='36.33%' stop-color='hsl(240,90%,50%)'/>

                <stop offset='44%' stop-color='hsl(210,90%,50%)'/>

                <stop offset='49.67%' stop-color='hsl(180,90%,50%)'/>

                <stop offset='53%' stop-color='hsl(150,90%,50%)'/>

                <stop offset='55.22%' stop-color='hsl(120,90%,50%)'/>

                <stop offset='57.44%' stop-color='hsl(90,90%,50%)'/>

                <stop offset='59.67%' stop-color='hsl(60,90%,50%)'/>

                <stop offset='63%' stop-color='hsl(30,90%,50%)'/>

                <stop offset='66.67%' stop-color='hsl(0,90%,50%)'/>

                <stop offset='71.33%' stop-color='hsl(0,90%,35%)'/>

                <stop offset='100%' stop-color='hsl(0,90%,20%)'/>

            </linearGradient>

        </defs>

        <rect

            x='0'

            y='0'

            width='" & Self.Width & "'

            height='" & Self.Height & "'

            fill='url(#TempGradient)'

        />

    </svg>"

)

This is our previous SVG code with some changes. Essentially, we have set the width and height properties of the SVG itself, the SVG viewBox, and the rectangle that houses the color gradient to the same as the image control it resides in. After doing this, we should see the following:

A screenshot of a computer

Description automatically generated

Now comes the hard part. We want the color gradient to only show the range set by the low temperature on the left and the high temperature on the right. In addition to that, we want it to fit exactly in the image control so that the width of the gradient range we are using fits the full width and height. To do this, we need to scale the height and width of the SVG and rectangle object inside the SVG, as well as adjust the viewBox of the SVG. Just to give some information about the viewBox, it represents a viewable area within the SVG. Most of the time, we want to see the whole SVG so it often has the same dimensions as the SVG itself. Not so in our case, as we want to only see the section that represents the given temperature range. I’m going to dump the code to put in the Image property and then explain what is happening.

With(

{

// For imperial units, use -100 for min, 200 for max

// For metric units, use -73.33 for min, 93.33 for max

wth\_MinTemp: -100,

wth\_MaxTemp: 200

},

With(

{

wth\_ScaledWidth: (wth\_MaxTemp – wth\_MinTemp) \* img\_TempBackground.Width / (txt\_HiTemp.Text - txt\_LoTemp.Text)

},

With(

{

wth\_ScaledHeight: wth\_ScaledWidth \* Self.Height / Max(Self.Width, 1),

wth\_StartingPosition: (txt\_LoTemp.Text – wth\_MinTemp) \* (wth\_ScaledWidth / (wth\_MaxTemp – wth\_MinTemp))

},

"data:image/svg+xml," & EncodeUrl(

"<svg

width='" & wth\_ScaledWidth & "'

height='" & wth\_ScaledHeight & "'

viewBox='" & wth\_StartingPosition & " 0 " & Self.Width & " " & Self.Height & "'

version='1.1' xmlns='http://www.w3.org/2000/svg'

>

<defs>

<linearGradient id='TempGradient' x1='0' x2='1' y1='0' y2='0'>

<stop offset='0%' stop-color='hsl(300,90%,80%)'/>

<stop offset='20%' stop-color='hsl(300,90%,50%)'/>

<stop offset='28%' stop-color='hsl(270,90%,50%)'/>

<stop offset='36.33%' stop-color='hsl(240,90%,50%)'/>

<stop offset='44%' stop-color='hsl(210,90%,50%)'/>

<stop offset='49.67%' stop-color='hsl(180,90%,50%)'/>

<stop offset='53%' stop-color='hsl(150,90%,50%)'/>

<stop offset='55.22%' stop-color='hsl(120,90%,50%)'/>

<stop offset='57.44%' stop-color='hsl(90,90%,50%)'/>

<stop offset='59.67%' stop-color='hsl(60,90%,50%)'/>

<stop offset='63%' stop-color='hsl(30,90%,50%)'/>

<stop offset='66.67%' stop-color='hsl(0,90%,50%)'/>

<stop offset='71.33%' stop-color='hsl(0,90%,35%)'/>

<stop offset='100%' stop-color='hsl(0,90%,20%)'/>

</linearGradient>

</defs>

<rect

x='0'

y='0'

width='" & wth\_ScaledWidth & "'

height='" & wth\_ScaledHeight & "'

fill='url(#TempGradient)'

/>

</svg>"

)

)

)

)

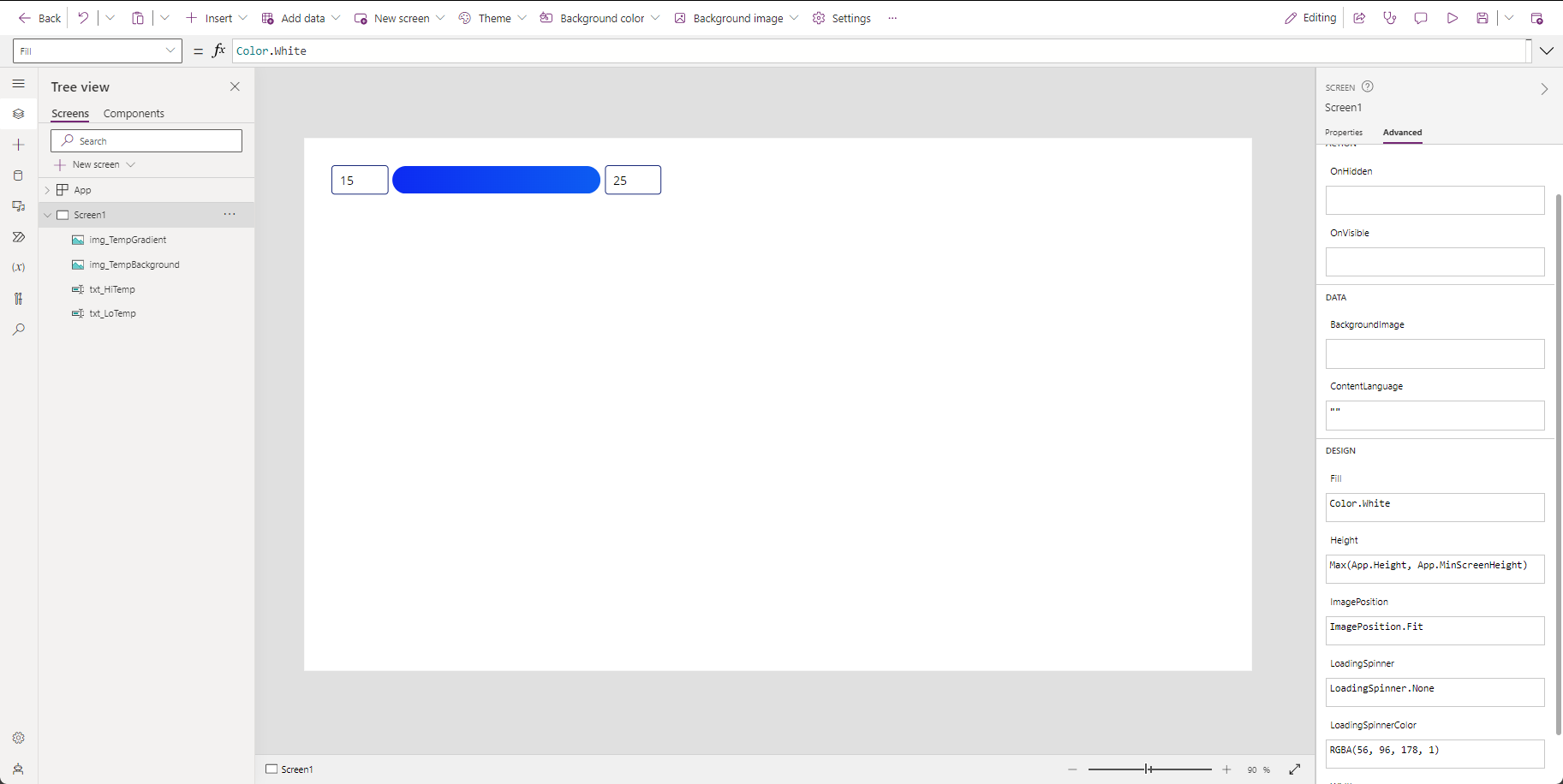
First off, the top With function is setting the endpoints of our temperature gradient range based on whether we are using Imperial (°F) or Metric (°C) units. I am using Imperial units, so I set wth\_MinTemp to -100 and wth\_MaxTemp to 200.

The next With function is setting the scaled width for the SVG and rectangle in the SVG by using the width of img\_TempBackground and the given temperature values to create a ratio. One note on this: we could replace img\_TempBackground.Width with Self.Width but, again, setting it up this way will help us in the two-range version.

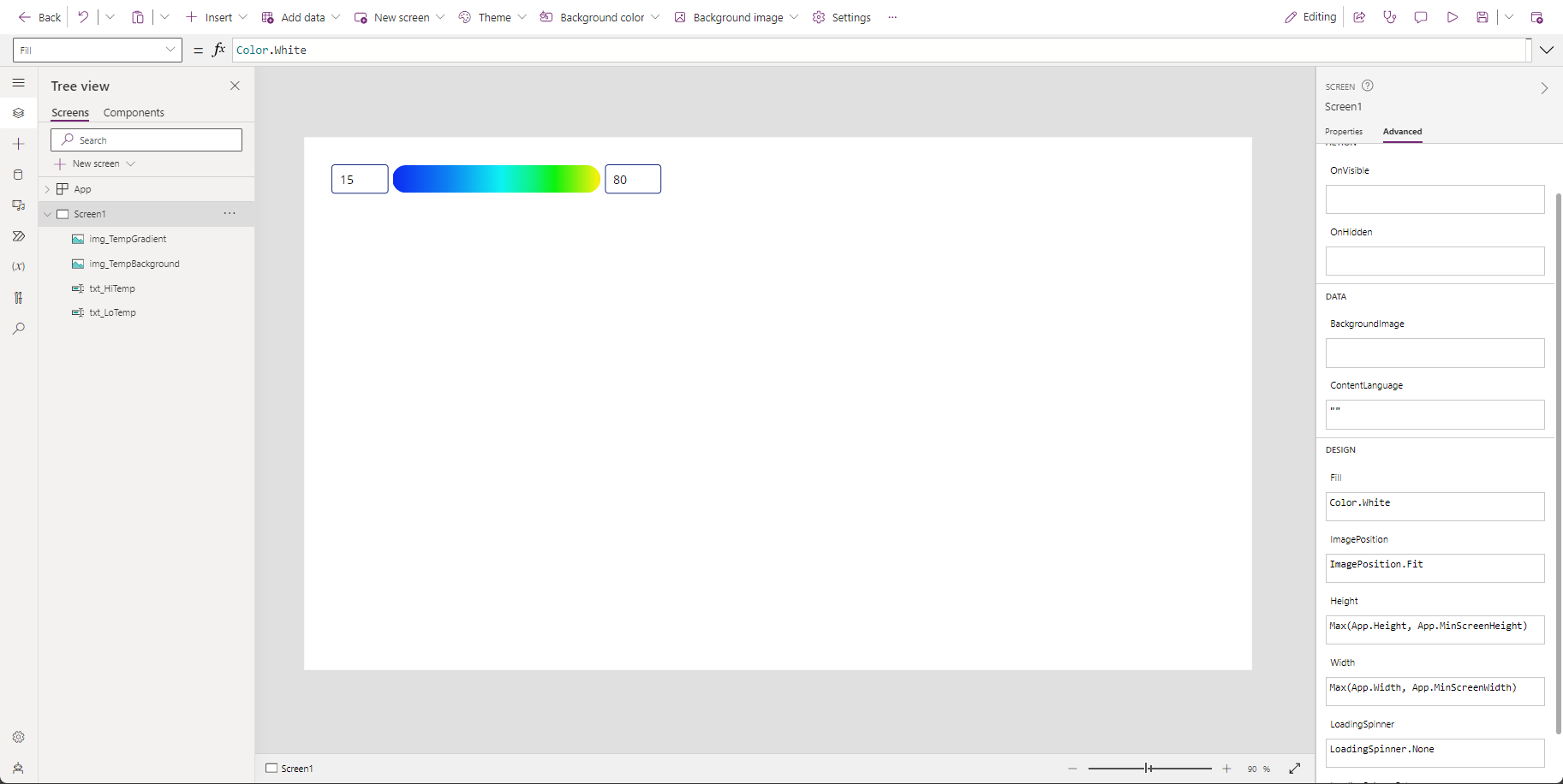
The third With function is doing two things: scaling the height relative to the width so that the resulting shape fits the Image control and setting the starting point for the viewBox so that we see just the area of the SVG that fits the temperature range.

To summarize these changes, we made the SVG big enough that our chosen viewBox is exactly the width and height of the Image control it is represented by and then made it so the viewBox starts at the low temperature and ends at the high temperature.

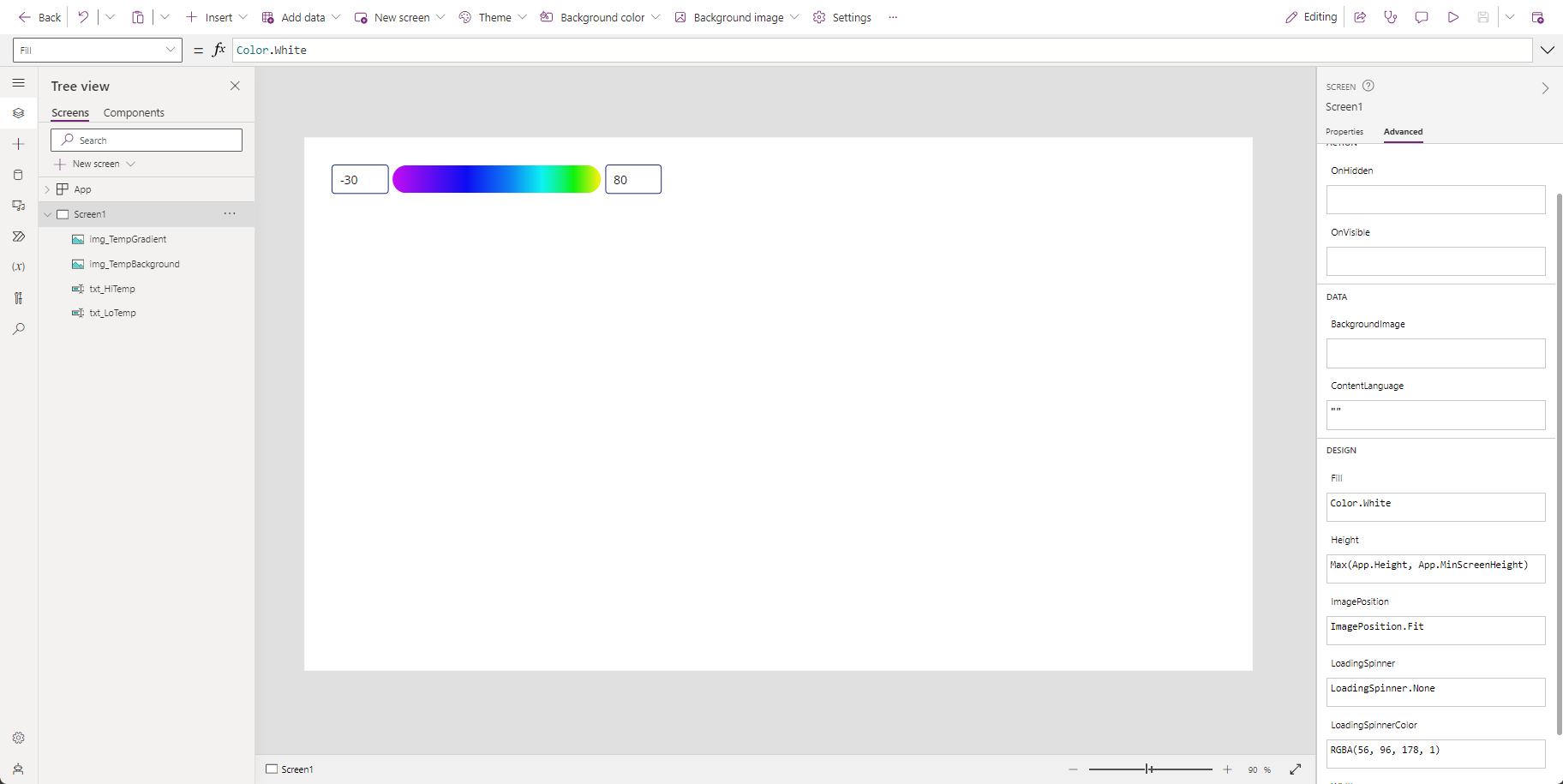
It should look thusly:



Pretty cool right? How about we test to make sure that it works though. Let’s enter 80 for the high temp. We should see this:



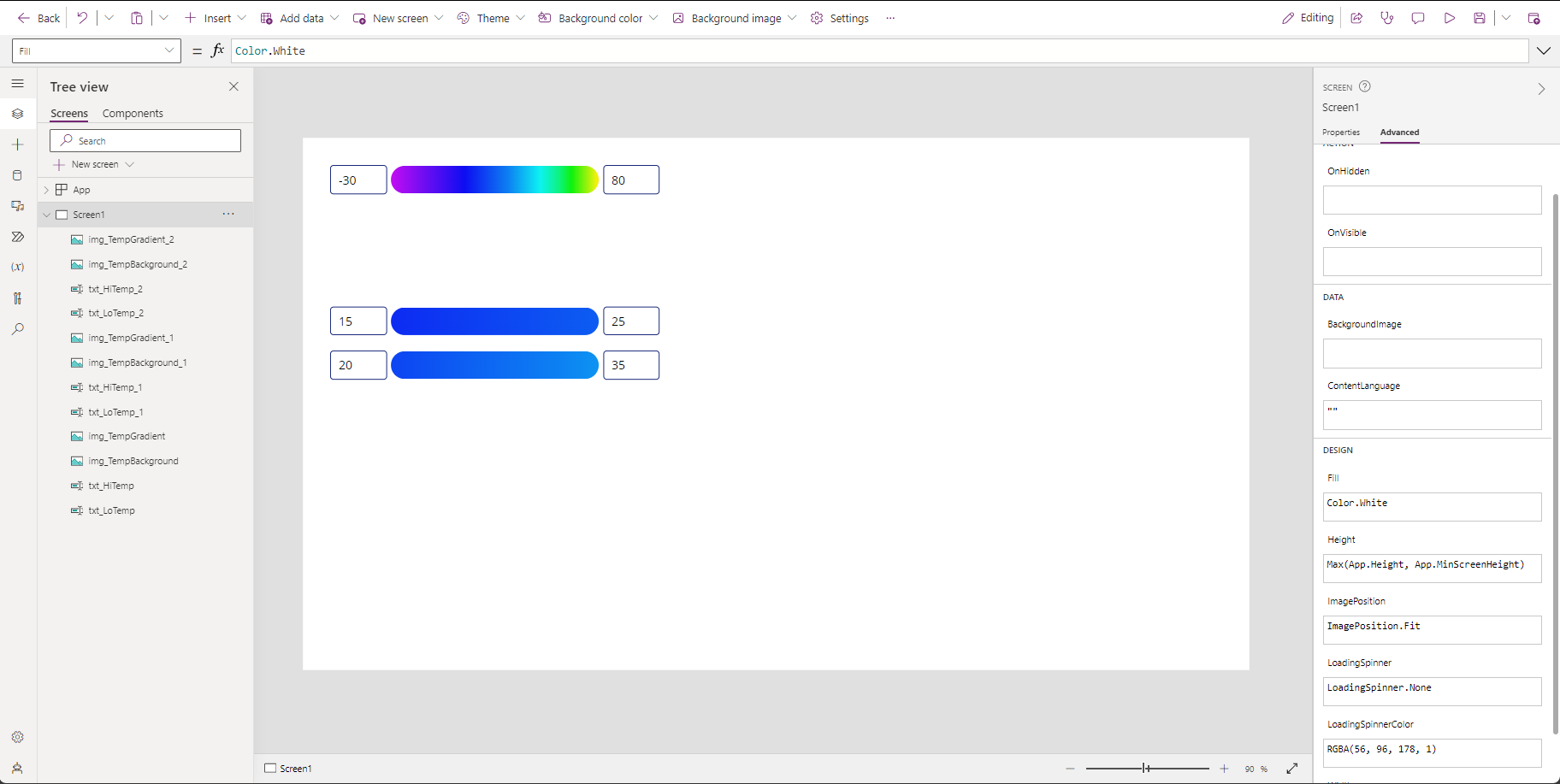
And just to make doubly sure, let’s set the low temp to -30. We should then see:



Success! Now, let’s make it work for two temperature ranges.

## Two Range Gradients

We’re going to start by copying and pasting all four existing controls twice and positioning them like shown below. We’ll also set the low and high temperatures for the bottom set of controls (txt\_LoTemp\_2 and txt\_HiTemp\_2) to 20 and 35, respectively.



We will replace the Image property for the upper temperature gradient image (txt\_TempGradient\_1) with:

With(

    {

        // For imperial units, use -100 for min, 200 for max

        // For metric units, use -73.33 for min, 93.33 for max

        wth\_MinTemp: -100,

        wth\_MaxTemp: 200

    },

    With(

        {

            wth\_ScaledWidth: (wth\_MaxTemp - wth\_MinTemp) \* *img\_TempBackground\_1*.Width / (Max(*txt\_HiTemp\_1*.Text, *txt\_HiTemp\_2*.Text) - Min(*txt\_LoTemp\_1*.Text, *txt\_LoTemp\_2*.Text))

        },

        With(

            {

                wth\_ScaledHeight: wth\_ScaledWidth \* *Self*.Height / Max(*Self*.Width, 1),

                wth\_StartingPosition: (*txt\_LoTemp\_1*.Text - wth\_MinTemp) \* (wth\_ScaledWidth / (wth\_MaxTemp - wth\_MinTemp))

            },

            "data:image/svg+xml," & EncodeUrl(

                "<svg

                    width='" & wth\_ScaledWidth & "'

                    height='" & wth\_ScaledHeight & "'

                    viewBox='" & wth\_StartingPosition & " 0 " & Self.Width & " " & Self.Height & "'

                    version='1.1' xmlns='http://www.w3.org/2000/svg'

                >

                    <defs>

                        <linearGradient id='TempGradient' x1='0' x2='1' y1='0' y2='0'>

                            <stop offset='0%' stop-color='hsl(300,90%,80%)'/>

                            <stop offset='20%' stop-color='hsl(300,90%,50%)'/>

                            <stop offset='28%' stop-color='hsl(270,90%,50%)'/>

                            <stop offset='36.33%' stop-color='hsl(240,90%,50%)'/>

                            <stop offset='44%' stop-color='hsl(210,90%,50%)'/>

                            <stop offset='49.67%' stop-color='hsl(180,90%,50%)'/>

                            <stop offset='53%' stop-color='hsl(150,90%,50%)'/>

                            <stop offset='55.22%' stop-color='hsl(120,90%,50%)'/>

                            <stop offset='57.44%' stop-color='hsl(90,90%,50%)'/>

                            <stop offset='59.67%' stop-color='hsl(60,90%,50%)'/>

                            <stop offset='63%' stop-color='hsl(30,90%,50%)'/>

                            <stop offset='66.67%' stop-color='hsl(0,90%,50%)'/>

                            <stop offset='71.33%' stop-color='hsl(0,90%,35%)'/>

                            <stop offset='100%' stop-color='hsl(0,90%,20%)'/>

                        </linearGradient>

                    </defs>

                    <rect

                        x='0'

                        y='0'

                        width='" & wth\_ScaledWidth & "'

                        height='" & wth\_ScaledHeight & "'

                        fill='url(#TempGradient)'

                    />

                </svg>"

            )

        )

    )

)

Basically what is happening here is that we are now scaling our SVG with respect to the maximum high temperature and the minimum low temperature.

We will do the same in the Image property of img\_TempGradient\_2, which will be replaced with:

With(

{

// For imperial units, use -100 for min, 200 for max

// For metric units, use -73.33 for min, 93.33 for max

wth\_MinTemp: -100,

wth\_MaxTemp: 200

},

With(

{

wth\_ScaledWidth: (wth\_MaxTemp - wth\_MinTemp) \* img\_TempBackground\_2.Width / (Max(txt\_HiTemp\_1.Text, txt\_HiTemp\_2.Text) - Min(txt\_LoTemp\_1.Text, txt\_LoTemp\_2.Text))

},

With(

{

wth\_ScaledHeight: wth\_ScaledWidth \* Self.Height / Max(Self.Width, 1),

wth\_StartingPosition: (txt\_LoTemp\_2.Text - wth\_MinTemp) \* (wth\_ScaledWidth / (wth\_MaxTemp - wth\_MinTemp))

},

"data:image/svg+xml," & EncodeUrl(

"<svg

width='" & wth\_ScaledWidth & "'

height='" & wth\_ScaledHeight & "'

viewBox='" & wth\_StartingPosition & " 0 " & Self.Width & " " & Self.Height & "'

version='1.1' xmlns='http://www.w3.org/2000/svg'

>

<defs>

<linearGradient id='TempGradient' x1='0' x2='1' y1='0' y2='0'>

<stop offset='0%' stop-color='hsl(300,90%,80%)'/>

<stop offset='20%' stop-color='hsl(300,90%,50%)'/>

<stop offset='28%' stop-color='hsl(270,90%,50%)'/>

<stop offset='36.33%' stop-color='hsl(240,90%,50%)'/>

<stop offset='44%' stop-color='hsl(210,90%,50%)'/>

<stop offset='49.67%' stop-color='hsl(180,90%,50%)'/>

<stop offset='53%' stop-color='hsl(150,90%,50%)'/>

<stop offset='55.22%' stop-color='hsl(120,90%,50%)'/>

<stop offset='57.44%' stop-color='hsl(90,90%,50%)'/>

<stop offset='59.67%' stop-color='hsl(60,90%,50%)'/>

<stop offset='63%' stop-color='hsl(30,90%,50%)'/>

<stop offset='66.67%' stop-color='hsl(0,90%,50%)'/>

<stop offset='71.33%' stop-color='hsl(0,90%,35%)'/>

<stop offset='100%' stop-color='hsl(0,90%,20%)'/>

</linearGradient>

</defs>

<rect

x='0'

y='0'

width='" & wth\_ScaledWidth & "'

height='" & wth\_ScaledHeight & "'

fill='url(#TempGradient)'

/>

</svg>"

)

)

)

)

We won’t notice much of a change just yet, but we will once we make a couple more additions. Let’s make the following changes to img\_TempGradient\_1:

Width: With(

{

wth\_TempSpread: txt\_HiTemp\_1.Text - txt\_LoTemp\_1.Text,

wth\_MaxSpread: Max(txt\_HiTemp\_1.Text, txt\_HiTemp\_2.Text) - Min(txt\_LoTemp\_1.Text, txt\_LoTemp\_2.Text)

},

img\_TempBackground\_1.Width \* wth\_TempSpread / wth\_MaxSpread

)

X: With(

    {

        wth\_TempLo: *txt\_LoTemp\_1*.Text,

        wth\_MinLo: Min(*txt\_LoTemp\_1*.Text, *txt\_LoTemp\_2*.Text)

    },

    With(

        {

            wth\_MaxSpread: Max(*txt\_HiTemp\_1*.Text, *txt\_HiTemp\_2*.Text) – wth\_MinLo

        },

*img\_TempBackground\_1*.X + ((wth\_TempLo – wth\_MinLo) / wth\_MaxSpread) \* *img\_TempBackground\_1*.Width

    )

)

And similarly, let’s change the Width and X of img\_TempGradient\_2 to:

Width: With(

{

wth\_TempSpread: txt\_HiTemp\_2.Text - txt\_LoTemp\_2.Text,

wth\_MaxSpread: Max(txt\_HiTemp\_1.Text, txt\_HiTemp\_2.Text) - Min(txt\_LoTemp\_1.Text, txt\_LoTemp\_2.Text)

},

img\_TempBackground\_2.Width \* wth\_TempSpread / wth\_MaxSpread

)

X: With(

    {

        wth\_TempLo: *txt\_LoTemp\_2*.Text,

        wth\_MinLo: Min(*txt\_LoTemp\_1*.Text, *txt\_LoTemp\_2*.Text)

    },

    With(

        {

            wth\_MaxSpread: Max(*txt\_HiTemp\_1*.Text, *txt\_HiTemp\_2*.Text) – wth\_MinLo

        },

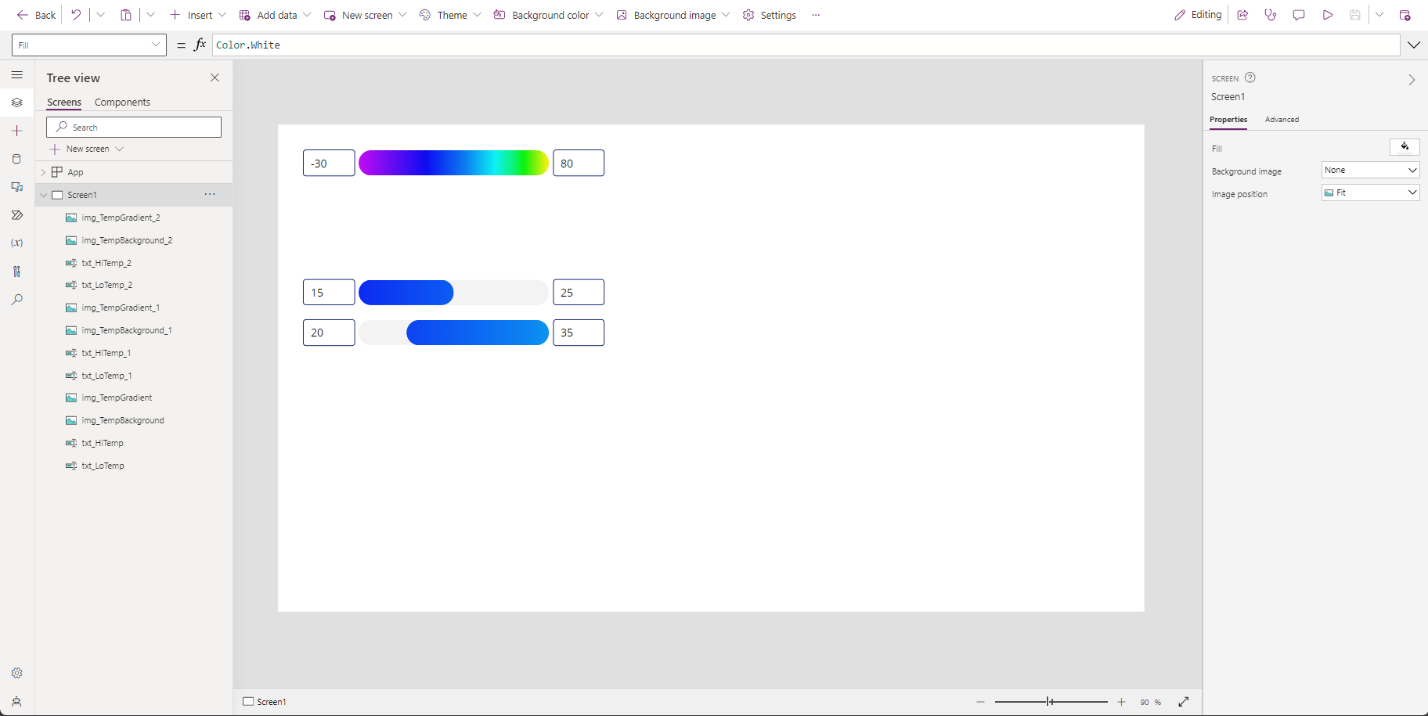
*img\_TempBackground\_2*.X + ((wth\_TempLo – wth\_MinLo) / wth\_MaxSpread) \* *img\_TempBackground\_2*.Width

    )

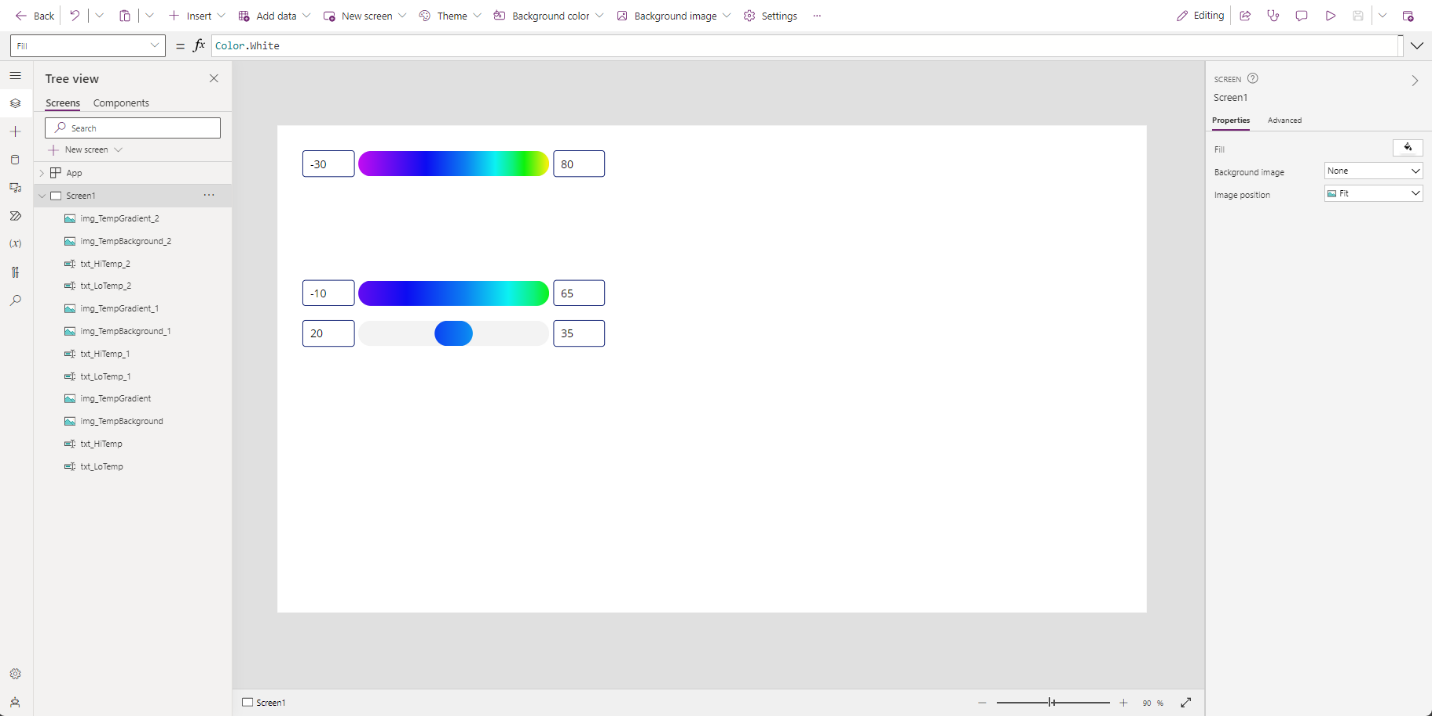
)

So…what did we do there exactly? Three things: First, we are treating the temperature backgrounds (img\_TempBackground\_1 and img\_TempBackground\_2) as placeholders for the total temperature spread (i.e. the difference between the max high temp and min low temp). Second, we are scaling the width of each temperature gradient (img\_TempGradient\_1 and img\_TempGradient\_2) to fit its individual temperature spread relative to the total temperature spread. And third, we are adjusting the starting point of each temperature gradient, again relative to the total temperature spread.

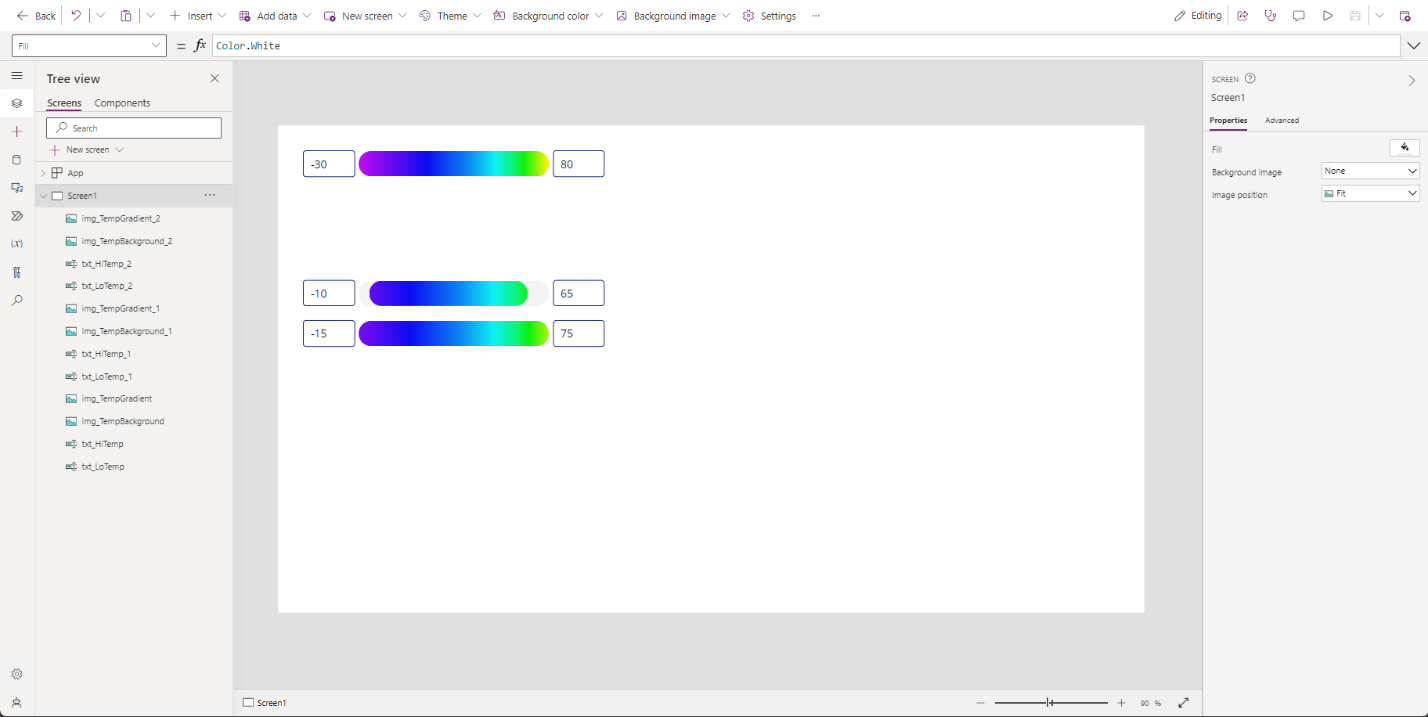
It should look like this:



Again, let’s test this out to make sure it is working properly. Set the low and high temperatures of the upper set of text input controls (txt\_LoTemp\_1 and txt\_HiTemp\_1) to -10 and 65, respectively. Here’s what we should see:



And let’s change the bottom set of text input controls (txt\_LoTemp\_2 and txt\_HiTemp\_2) to -15 and 75, respectively. It should look like so:



And once again, we have success!

## Final Thoughts

By replacing the text inputs with data from a weather API, we can have a nice UI for our users to enjoy. Furthermore, because the SVG can be used with both Imperial and Metric units, it can be easy to localize by setting up a condition where, based on the units used it will set the wth\_MinTemp and wth\_MaxTemp to the necessary values.

In using the MSN Weather connector for instance, we can find the unit system and then use that. If we used the MSNWeather.TodaysForecast call and set it in a global variable called glb\_TodaysForecast, we could access it like so:

glb\_TodaysForecast.units.system

In my case, it would return “Imperial”. So, setting the wth\_MinTemp and wth\_MaxTemp to the following should localize things nicely:

{

// For imperial units, use -100 for min, 200 for max

// For metric units, use -73.33 for min, 93.33 for max

wth\_MinTemp: If(  
 glb\_TodaysForecast.units.system = “Imperial”,  
 -100,

-73.33  
),

wth\_MaxTemp: If(  
 glb\_TodaysForecast.units.system = “Imperial”,  
 200,

93.33  
)

}

As with everything, there are more ways to accomplish this task, but this is one decent way.

Another idea is to do this with more than two temperature ranges. The main changes that would need to be made would be to take the min and maxes of all the temperatures and replace all the places that do that with just two (i.e. the X, Width, and wth\_ScaledWidth and wth\_StartingPosition in the SVG code for each of the temperature gradient image controls).

To further extend the use of this idea, here is a screenshot of one possibility for using the MSN Weather connector and this temperature gradient to show temperature data in apps.

A screen shot of a computer

Description automatically generated

(Just so it’s known, I do understand that using modals in Power Apps in the way shown isn’t great for screen readers. I can feel a finger wag of consternation coming across the pond from my buddy Sancho Harker. Also, this is an uncommonly nice couple of days for February in Wyoming. T-shirts and shorts all around! 😁)

I hope you have enjoyed this little journey into the realm of SVG color gradients and temperature. I will have a link to download the example app we built at the bottom of this post. It will be an app, not a solution, so import accordingly! If you have any questions or comments, feel free to share them here or hit me up on social media. Thank you for reading!