XML Gauge Programming for FS2004. Chapter 1. Main Body Sections V2.0

Version 2.0

By Nick Pike June, 2005

INDEX

XML is a text based programming language. Therefore, the code can be written in a standard text editor. There are applications available specifically written for XML, but I have always used an enhanced shareware version of Notepad. If code is saved with a txt extension, rename with an xml extension. XML gauges usually consist of the xml file and bmp (bitmap) files, although gauges without bitmaps are quite common.

This tutorial provides a general introduction to gauge main body sections and their functions. If you read this tutorial first, it will give some good foundation information Colours have been used to group relative information, or to allow the reader to easily find references in code or text.

1.0 <Gauge>

XML gauges follow a common theme of being broken down into sections, each with an opening and closing instruction. A good example is that a gauge always starts with <Gauge> and ends with </Gauge>. The closing instruction always has the character '/'. This tells flightsim that the section has ended. A simple example of this:

```
<Gauge Name="f117 aileron trim" Version="1.0">
  <Image Name="bitmap1.bmp"/>
</Gauge>
```

This gauge would simply place the fixed, non-dynamic bitmap on the panel. Note that in line 1, <Gauge> is actually expanded to include the gauge name (can be any name) and the version.. Line 1 can always be the same but with maybe a different name for each gauge. Bitmaps can be 8 or 16 bit (256 or millions of colours). Note that line 2 ends '/>'. The '/' tells flightsim there is no more bitmap information to follow. In certain circumstances more information can follow to manipulate or control a bitmap, and the '/' is omitted in this case, but more on this later. You do not have to give the bitmap sizes. You do however in FS2002, but these tutorials are for FS2004 to keep this tutorial as simple as possible. The bitmap can be of any size and proportion, both in the X and Y axis. However, to optimise frame rates in FS2004, it is best to keep it as small as practical and 8 bit. The bitmap name is the same as the bitmap file name (and can be any name), including the bmp extension.

1.1 <Element>

If you want the gauge to produce dynamic effects, like a light switching on or off, or a needle to move, or have FS2004 calculate variables or show text, you have to put the instructions in an <Element> </Element> section. Note the '/' again to close the section. There is a multitude of ways to introduce instructions in this section type, but more of that later. The following code is just an example of how the Element section works.

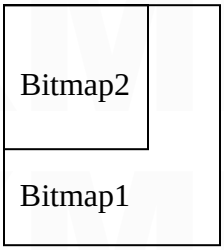
```
 <Element>
    <Image Name="bitmap2.bmp"/>
 </Element>
```

If this is introduced to our original code,

```
<Gauge Name="f117 aileron trim" Version="1.0">
  <Image Name="bitmap1.bmp"/>
                                          << Note that line gaps can be
 <Element>                                   introduced to visually
    <Image Name="bitmap2.bmp"/>              highlight separate sections.
 </Element>

</Gauge>
```

The above gauge is not much use, but shows the principle. This gauge would place bitmap2 over bitmap1, thus



Note: the drawn order is from last to first, that is, bitmaps introduced later in the code show on top.

This example shows bitmap2 that is smaller than bitmap1

Note: The 0,0 origin for a gauge bitmap is in the top left corner.
The X axis increases its value from left to right.
The Y axis increases its value from top to bottom.
Both bitmaps will anchor at 0,0, unless positional information is given, but more on that later.

1.2 <Select>, <Value> and <Case Value.

Let's take a look at this typical gauge section,

```
<Element>
  <Select>
    <Value>(G:Var1)</Value><<Don't worry about what (G:Var1) is for now.
    <Case Value="0">
      <Image Name="switch OFF.bmp"/>
    </Case>
    <Case Value="1">
      <Image Name="switch ON.bmp"/>
    </Case>
  </Select>
</Element>
```

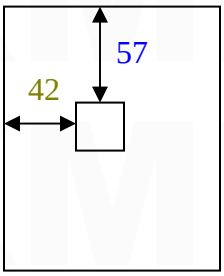
The <Select> </Select> section literally selects a bitmap. The <Value></Value> line is used to generate or calculate a value which in this case generates a value of 0 or 1. Note the <Case Value="x"> </Case> sections, that start and end a choice selection according to the variable value. If the value of the variable (G:Var1) is 0, the <Case Value="0"> instruction is activated and the associated switch OFF.bmp will be shown on the panel, and if 1, the <Case Value="1"> instruction is activated and the associated switch ON.bmp is shown.. The bitmaps could be switch pictures, say a rocker switch that will look like it's in the off or on condition. They may also be lamp pictures, again looking like they are off or illuminated.
We can introduce a positional instruction here, to position the selective bitmaps on the gauge. In the very first example, item 1.1, the bitmap can be used as a background image, and also a 'canvas' for the rest of the gauge. Note: It also sets the gauge size in pixels.

```
<Gauge Name="f117 aileron trim" Version="1.0">
  <Image Name="bitmap1.bmp"/>
```

We can add,

```
<Element>
 <Position X="42" Y="57"/>              << Added
  <Select>
   etc.
```

The inclusion of the position instruction can position the selective bitmaps on the background (canvas) bitmap.



Note: All bitmap sizes are relative. That is, the selective bitmaps should be sized to look sensible against the background bitmap. A bitmaps own size determines its size in the gauge. There are no size instructions,

So the whole gauge would be,

```
<Gauge Name="f117 aileron trim" Version="1.0">
  <Image Name="bitmap1.bmp"/>
  <Element>
  <Position X="42" Y="57"/>
    <Select>
      <Value>(G:Var1)</Value>
      <Case Value="0">
        <Image Name="switch OFF.bmp"/>
      </Case>
      <Case Value="1">
        <Image Name="switch ON.bmp"/>
      </Case>
    </Select>
  </Element>
</Gauge>
```

1.3 <Shift> and <Nonlinearity>

The <Shift> section can move bitmaps over the background (canvas) bitmap in the X or Y direction (linear movement).

The values used in this example look huge, but they are for a very long bitmap that contains all the altitude numbers (commonly called the altitude tape), and has to move a large distance in the Y direction. The low altitude numbers are at the bottom of the tape.

```
<Element>
    <Position X="399" Y="358"/>
     <Image Name="f117 pfd_ALTITUDE_number_strip.bmp">
       <Nonlinearity>
         <Item Value="       0" X="0" Y="13644"/>
         <Item Value="80000" X="0" Y="     45"/>
       </Nonlinearity>
     </Image>
<Shift>
 <Value Minimum="-1000" Maximum="69000">(A:Indicated Altitude, feet)</Value>
</Shift>
</Element>
```

Here we see element, position and value again. We also introduce <Nonlinearity> </Nonlinearity> and <Shift></Shift>. Nonlinearity and Shift work together. The value line calculation produces a numerical value. This is enclosed by Shift, and so we can get a bitmap to move by the value calculated. We need to tell FS2004 in which direction to move and by how much. For the moment, ignore the position instruction line.
In line <Item Value="   0" X="0" Y="13644"/> we are saying that when the Value is 0, the top left of the bitmap is 13644 pixels from (above) the top of the background bitmap.
In line <Item Value="80000" X="0" Y="   45"/> we are saying that when the Value is 80000, the top left of the bitmap is 45 pixels from (above) the top of the background bitmap.
Therefore, you can see that the tape bitmap will move up and down.
The X in X=”0” in both lines shows that there is no movement in the X direction.

Without the position instruction line <Position X="399" Y="358"/>, both the left hand edges of the background and the moving bitmap would align. In reality, we want the moving bitmap to be positioned somewhere across the background bitmap (X axis). We could do this by giving X in <Item Value="   0" X="0" Y="13644"/> a value. In a simple gauge, this would suffice. However, in more complex gauges, there is a better way. The line <Position X="399" Y="358"/> says that the left hand edge of the tape bitmap is 399 pixels away from the left hand edge of the background bitmap. The top of the tape is using the top of the background bitmap as the datum at present. With the introduction of Y=358, the datum for the tape is now 358 pixels down from the top of the background bitmap. Therefore, in <Item Value="80000" X="0" Y="   45"/> the top of the tape is at 358-45 = 313.
Spotted anything? Ready for something totally confusing?
Y values in <Position X="399" Y="358"/> move down. In movement instructions like <Item Value="   0" X="0" Y="   45"/>, Y movements move up. There are some anomalies in XML programming, but not too many.

The <Value Minimum="-1000" Maximum="69000"> puts limits to the bitmap movement. If the value line produces a value lower (more negative) than -1000, the bitmap will not move any further. Again, the bitmap stops if the value is above 69000.

This means that the values on the bitmap can start at -1000 and end at 69000. Without these limits, the tape would move off the gauge, and so the limits act as mechanical stops. There is a second way to produce shift. This is part of a PFD gauge (not that it matters).

```
<Shift>
<Value Minimum="-90" Maximum="90">(A:Attitude indicator pitch degrees, degrees) /-
/</Value>
<Scale Y="4"/>
</Shift>
```

Here we see a value line that generates a number in the range -90 to +90. Notice the <Scale Y="4"/> instruction. Without this, the bitmap being controlled would move the number of pixels being generated by the value line, from a datum position. This is fairly restrictive, so to give total flexibility, the scale instruction is used. The Y character tells the bitmap to move in the Y direction (up and down). This could be an X character and produce sideways movement. The Y=”4” multiplies the value generated by 4. So if the positive maximum of 90 is generated, the bitmap will move 4 x 90 = 360 pixels.

1.4 <Rotate> and <Axis>

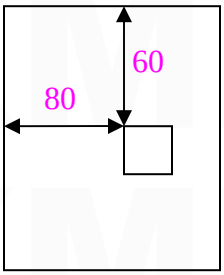
```
<Element>
  <Position X="80" Y="60"/>
  <Image Name="f117 pfd knob.bmp">
    <Axis X="20" Y="20"/>
  </Image>
  <Rotate>
    <Value>(G:Var1)</Value>
    <Nonlinearity>
      <Item Value="0"  X="70" Y="50"/>
      <Item Value="1"  X="90" Y="50"/>
      <Item Value="2"  X="90" Y="70"/>
      <Item Value="3"  X="70" Y="70"/>
    </Nonlinearity>
  </Rotate>
</Element>
```

This is very similar to <Shift>, only this time the bitmap is rotated.
This example rotates a bitmap of a knob. Looking at the code,

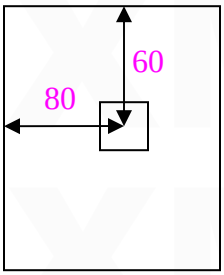
```
  <Image Name="f117 pfd knob.bmp">
    <Axis X="20" Y="20"/>
  </Image>
```

If only <Image Name="f117 pfd knob.bmp"/> was given, the knob bitmap would rotate about its top left hand corner. All rotating bitmaps will have X and Y dimensions, being knobs, needles or whatever. We need to state the centre of rotation on the bitmap. The knob bitmap is 40 x 40 pixels. We want it to rotate about its centre. The line
<Axis X="20" Y="20"/>

moves the top left hand corner 20 pixels to the left (in this case, opposite to what you might think) and 20 pixels up. Further coding will now rotate the knob about its centre. Note now that the '/' character is missing from the line
<Image Name="f117 pfd knob.bmp">
meaning that further control is expected. The instruction is finished by the code </Image>.
Without a positional instruction, <Position X="80" Y="60"/>, the knob centre would now coincide with the top left corner of the background (canvas) bitmap. In reality, I should have explained the positional instruction first, and when you get used to this technique, that is where you will start, and then apply the axis instruction. I wanted you to grasp the reasoning of the pivot point first.
So, to position the knob, apply the positional instruction first. This sets where the knob centre will end up.



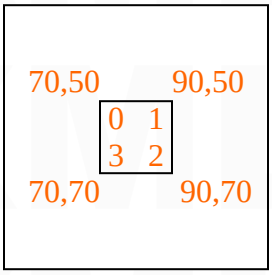
Then apply the axis information>>



We now have the knob in its correct location. The next thing to do is to get it to rotate. We use <Nonlinearity> </Nonlinearity> again but this time to rotate. Note in this case, <Nonlinearity> </Nonlinearity> is sandwiched by the Rotate instructions.
The code

```
<Item Value="0"  X="70" Y="50"/>
<Item Value="1"  X="90" Y="50"/>
<Item Value="2"  X="90" Y="70"/>
<Item Value="3"  X="70" Y="70"/>
```

works like this. The value line generates values between 0 and 3 in this case. This method works similarly to shift, except both the X and Y values change. The X and Y co-ordinates should lie on a circle. To find the X and Y positions, superimpose the knob bitmap onto the background bitmap in a graphics editor. Move the cursor onto the top left corner of the knob bitmap and read the X and Y position. This gives the X and Y values in <Item Value="0" X="70" Y="50"/>
Go to the remaining corners and do the same.



The numbers inside the small rectangle are the <Value> numbers. As the <Value> numbers increase, the top left hand corner of the knob bitmap moves to each of the corner co-ordinates.
In this case, these simple values allow the evaluation with mental arithmetic, but in reality, bitmaps are not usually sized with 'easy to use numbers'

The knob will not jump from one position to another but will turn smoothly. If the <Value> number were to go above 3, or below 0, the knob would continue to rotate.
DO NOT do the following.

```
<Item Value="0"  X="70" Y="50"/>
<Item Value="1"  X="90" Y="50"/>
<Item Value="2"  X="90" Y="70"/>
<Item Value="3"  X="70" Y="70"/>
<Item Value="4"  X="70" Y="50"/>          << this line has the same X,Y
```

co-ordinates as the Item Value="0" line. This will confuse things and the knob will jump about trying to satisfy two similar conditions. Make sure the same co-ordinates are NOT repeated.
We could add limits to the knob movement. The value line could read,
<Value Minimum="0" Maximum="4">(G:Var1)</Value>
This means the knob will be restricted to one complete revolution.

There is another way to get bitmaps to rotate. The above uses XY co-ordinates on the background bitmap. A more mathematical way is to use radians. Radians are not difficult to understand. We all know that a complete rotation in degrees is 360° (least, I hope we do). The equivalent in radians is 2π (2 times pie). Pie has a value of 3.14159 to 5 decimal places. Therefore, the value in radians for a complete rotation is 2x3.14159 = 6.28318. Another plus is that computers understand radians better (faster) than degrees.

This is the existing example but using radians this time

```
<Element>
  <Position X="80" Y="60"/>
  <Image Name="f117 pfd knob.bmp">
    <Axis X="20" Y="20"/>
  </Image>
  <Rotate>
    <Value>(G:Var1) 4 / 6.28318 *</Value>
  </Rotate>
</Element>
```

Now (G:Var1) =0 is no rotation and (G:Var1) =4 is one full rotation. Do not worry about the Value maths at this time. This is covered by another tutorial. However, for the impatient, if we consider, say, half a rotation, (G:Var1) =2. This is divided by the full rotation to give a ratio or fraction, 2/4= ½ times the full rotation in radians, so ½ x 6.28318 = 3.14159. With no other information on offer, FS2004 will take this number as being radians (in a Rotate instruction). The Value could be supplied, for example, with <Value>(A:Attitude indicator bank degrees, radians)</Value> where FS2004 is told to use radians.
The advantage here is simplicity. You do not have to determine co-ordinates, and if you understand radians, this method is more flexible.

Yet another way to rotate a bitmap is to use degrees.

```
    <Rotate>
        <Value Minimum="-10" Maximum="10">(A:AILERON TRIM, degrees)
</Value>
      <Nonlinearity>
        <Item Value="-10" Degrees="-90"/>
        <Item Value=" 10" Degrees=" 90"/>
      </Nonlinearity>
    </Rotate>
```

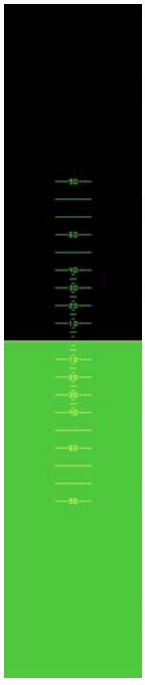
When the value line generates zero, the bitmap will not turn. When -10 is generated, the bitmap will rotate 90° anticlockwise, and +10, 90° clockwise.
In this case, the value line is calculating in degrees because of (A:AILERON TRIM, degrees). You may think that because the bitmap is rotating by a number of degrees, the value line has to calculate degrees. This is not actually the case. The bitmap will rotate 90° if the Item Value is 10. This can be 10 any units. So long as the value line calculates 10 (anything), the rotation is 90°.

1.5 Combining Shift and Rotate

As the title suggests, shift and rotate can be combined. A good example is the attitude ladder in a PFD (primary flight display). The ladder is the stacked horizontal lines that move up and down according to the aircraft pitch and rotate about the gauge centre according to the banking angle. Note, the bitmaps shown are not to scale.

Ladder

Ladder bitmap



```
<Element>
    <MaskImage Name="f117 pfd_attitude_card_mask.bmp"> <<
      <Axis X="260" Y="357"/>                          <<
    </MaskImage>                                       <<Discussed in
      <Image Name="f117 pfd_attitude_card.bmp">        <<the next section
        <Axis X="152" Y="760"/>                        <<
      </Image>                                         <<

    <Shift>
        <Value Minimum="-90" Maximum="90">(A:Attitude indicator pitch degrees,
    degrees) /-/</Value>
        <Scale Y="4"/>
    </Shift>

    <Rotate>
        <Value>(A:Attitude indicator bank degrees, radians)</Value>
    </Rotate>

</Element>
```

Looking at the Shift and Rotate instructions, they are both introduced to an Element section. The bitmap will now move up and down and rotate as a combined movement.

1.6 MaskImage

This is a good time to introduce MaskImage, seeing how it was used in the above example. In the above example, if,

```
<MaskImage Name="f117 pfd_attitude_card_mask.bmp">
  <Axis X="260" Y="357"/>
</MaskImage>
```

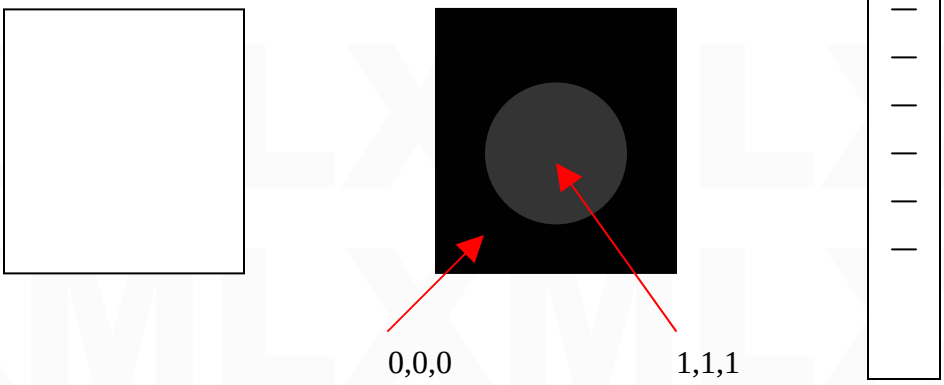
was missing, you would see the ladder bitmap from the top to bottom in the gauge. As it rotated you would see it from side to side. If you are familiar with a PFD gauge, you know that the ladder shows in a windowed area within the gauge. What the Maskimage instruction does is to provide an area where the associated bitmap (ladder, in this case) is seen, and outside this area, it is not seen. The Maskimage is a bitmap with a near black area, with colour depth 1,1,1. The part of the ladder that overlays the near black area will show. The remainder of the Maskimage is pure black, being 0,0,0. Note: Any area of an overlaid bitmap (the Maskimage is overlying the background bitmap) that is pure black will be invisible.

So we need to consider 3 bitmaps.

Background
All the eye-candy,
Screw heads etc.
1st in code, so placed
At bottom

Maskimage
Invisible part is 0,0,0
Masked area is 1,1,1
Shown dark grey here.

Ladder



To summarise, the Maskimage black area is invisible over the background, and the ladder will only show in the 1,1,1 masked area.

Maskimages have a peculiar way of positioning themselves. In this example, the Maskimage is the same size as the background bitmap. Lets say the Maskimage instruction was only <MaskImage Name="f117 pfd_attitude_card_mask.bmp"/> You would quite rightfully think that the top left hand corner of the background and the Maskimage would align. Wrong. A Maskimage left to its own devices will align its top left corner with the centre of the background. We need to introduce an axis instruction. <Axis X="260" Y="357"/> moves the top left corner 260 pixels left and 357 up. I hope you realise that the bitmap is 520 x 714. Now the Maskimage overlies the background. Now look at the code from the gauge in section 1.5,

```
<MaskImage Name="f117 pfd_attitude_card_mask.bmp">
   <Axis X="260" Y="357"/>
</MaskImage>
  <Image Name="f117 pfd_attitude_card.bmp">
    <Axis X="152" Y="760"/>
  </Image>
```

The MaskImage overlays the background completely. But where is the ladder (f117 pfd_attitude_card.bmp). Another peculiar thing. The ladder is associated with the MaskImage because they are in the same Element section. Left to its own devises, the top left corner of the ladder would align with the MaskImage centre. The ladder is 304x1520, therefore using <Axis X="152" Y="760"/> brings the centre of the ladder to the centre of the MaskImage. The centres of all three bitmaps now align. Phew…..time for a lie down.

The shift and rotate instructions now move the ladder up and down, and rotate, all about the datum which is the centre of the gauge, and the ladder only shows on the 1,1,1 area..

1.7 <Visible>

This is a very useful feature of XML. You can tell FS2004 to show (or hide) something according to a condition.
For example,

```
<!--rising runway-->
   <Element>
      <Position X="0" Y="0"/>
        <Visible>(A:Radio height, feet) 600 &lt;</Visible>
         <MaskImage Name="f117 pfd_attitude_card_mask.bmp">
           <Axis X="260" Y="357"/>
         </MaskImage>
             <Image Name="f117 pfd_rising_runway.bmp" Luminous="Yes">
               <Axis X="58.5" Y="0"/>
             </Image>
         <Shift>
         <Value>(P:Units of measure, enum) 2 == if{ (A:Radio height, meters) } els{
(A:Radio height, feet) } 0 max </Value>
          <Scale Y="0.2"/>
         </Shift>
       </Element>
```

This <Visible> code controls the visibility of a rising runway, which can be seen in the above view of the PFD in item 1.5. It is the red chevron bar. If the radio height is less than 600 feet, the rising runway bitmap will show in the gauge. The <Visible> instruction basically allows or disallows the operation of code in an <Element></Element> section.

1.8 Stacking Element sections

You can place element sections in other element sections. You can position several element sections with one positional instruction, or make several element sections visible with one visible instruction.

```
<Element>
 <Visible>(G:Var1)</Visible>
        <Element>
                Part of the gauge code here
        </Element>
        <Element>
                Part of the gauge code here
        </Element>
</Element>
```

You may have two sections of a gauge that you want to see according to one condition. In this example, if (G:Var1) satisfies a condition, the two sections will show or be hidden.

Also,

```
<Element>
    <Position X="122" Y="122"/>
        <Element>
                Part of the gauge code here
        </Element>
        <Element>
                Part of the gauge code here
        </Element>
</Element>
```

You may have two sections of a gauge that depend on the same position instruction.

.

Notice how a couple of element sections are sandwiched by a third element section. Note: in a complete gauge you should have an equal number of <Element> and </Element>.

1.9 <Keys>

This is an interesting instruction. This method actually reads key strokes, or mouse clicks on the monitor screen that has equivalent keystrokes.

```
<Keys>
<On Event="ATC_MENU_1">(A:COM1 active frequency, MHz) (&gt;L:COM1STBY, MHz)</On>
<On Event="ATC_MENU_2">(A:COM1 active frequency, MHz) (&gt;L:COM1STBY, MHz)</On>
<On Event="ATC_MENU_3">(A:COM1 active frequency, MHz) (&gt;L:COM1STBY, MHz)</On>
</Keys>
```

Don't worry too much about what this is actually achieving. ATC_MENU_1 can be any keystroke variable, and the achieved effect can be anything. In this case, if the ATC is active and key 1 is pressed, the instruction afterwards will be activated. In this case I am setting a variable to the COM1 active frequency. Why? Well, that needs to be the subject of an example tutorial. I'll put a list of keystroke variables on the site.

2.0 <Text> and <String>

So far we have looked at bitmaps as visual parts of the gauge. We can add lettering to the gauge, for example, to show label text or show actual digital values.

```
  <Element>
   <Text X="110" Y="10" Bright="Yes" Length="11" Font="Arial" Color="white" Adjust="Center" VerticalAdjust="Center">
    <String>1234567890X</String>
   </Text>
  </Element>
```

<Text> and <String> are always used together.
The <Text> line describes the lettering. Working through the line, the horizontal space allocated to the text is 110 pixels (X="110"). Be sure this is long enough to envelope the text. Some experimentation needed. The text height is 10 pixels (Y="10"). So the 'box' for the text is 110x10. The Bright literally makes it bright. This could be Luminous (see part 2.2 below). Length is the number of digits to be displayed. 1234567890X is 11 digits (count them). Font is the font type. Bear in mind this needs to be loaded on the computer. Arial is a good choice as it is a windows default. Color is white (there is a choice). I'll put a color list on the site. The color can also be given by hex values. For white, this would be Color="#FFFFFF". Adjust="Center" means the text will be horizontally centred in the text 'box'. VerticalAdjust="Center" centres the text in the vertical direction. There are other instruction, best saved for example gauges.
Now FS2004 knows what the text should look like. We now need to tell Fs2004 what to show. In this example, the text 1234567890X will show.

Another case. This line is in two parts. The variable and the print instruction. The variable has extra brackets. The '%' glues parts in the string line together.

<String>%((A:GPS WP NEXT ID, string))%!s!</String>

The variable (A:GPS WP NEXT ID, string) generates a string. That's technical talk for a group of letters. To show text in this case, we use !s!. The ID is four letters, so the Length="11" becomes Length="4" in the <Text> line.

The above examples show ways of dealing with text. We can also show numbers.
The String line could look like this,

<String>%((A:Indicated Altitude, feet))%!05d!</String>

This line is again in two parts. The variable in brackets generates the indicated altitude in feet. The !05d! part is display instructions. Again, an extra pair of brackets has been added, and the two parts are cemented together using the '%' character. Here, the 5 means that 5 digits will show, and the preceding 0 (zero) will fill any preceding digits without a value with zeroes. The d means no decimals. If the height is 755 feet, you will see 00755. Note, the preceding 0 cannot be used if a value can go negative; you can get something like 00-5. If 5 digits are shown, the Length="11" becomes Length="5"

The !05d! could be changed for, say, !8.2f!. The .2f means there are two decimal places.

<String>%((A:Indicated Altitude, feet))%!8.2f!</String

This would produce, say, 755.47. There is no preceding 0, so any preceding value of zero is not shown. The 8 means there will now be 8 digits, made up of the five possible figures to the left of the decimal point, the decimal point itself, and the two figures to the right of the decimal point. That's 5+1+2=8 Don't forget that the .2 (decimal 2) means shown to two decimal points. The Length ="11" now becomes Length ="8". Note, altitude is not

normally shown using decimal places, this is just for the example. Again, example gauges should be consulted for different ideas.

2.1 Section Titles and Comments

Text can be placed at the start of each section to remind you what it does.
Putting any text between <!-- and --> will not effect the gauges operation, thus

<!--Frequency window--> would entitle that section.

Also, if you are developing a gauge and you want to turn off a section during testing, or you want to put in some text, maybe instructions, you can sandwich a section with <Comment> in front and </Comment> at the end. Anything between these two instructions will be ignored.

2.2 Luminous and Bright

The way a bitmap or text looks during dusk, night and dawn can be controlled with the instructions Luminous and Bright. For example

<Image Name="f117 ailr trim bg.bmp" Luminous="Yes"/>
<Image Name="f117 ailr trim bg.bmp" Bright="Yes"/>

The Luminous instruction will cause the bitmap or text to glow according to the values, Luminous=255,50,50 found usually at the end of the panel.cfg file, thus:

[Color]
Day=255,255,255
Night=100,80,80
Luminous=255,50,50

The Bright instruction quite literally gives a bright coloured level.
Without either instruction, the lighting level will be according to the Night levels.

2.3 Gauge Cycle Time or Refresh Rate

At the start of a lot of gauges, you will see lines like,

<Gauge Name="f117 radios" Version="1.0">
  <Image Name="f117 radio bg.bmp"/>
<Update Frequency="6"/>

The Update Frequency dictates how often the gauge will be run. Without this, the gauge will refresh 18 times per second.

For example, <Update Frequency="1"/> means the gauge will be run once a second.

<Update Frequency="2"/> means the gauge will be run once every half second.
<Update Frequency="3"/> means the gauge will be run once every third of a second.
<Update Frequency="4"/> Actually 4.5 times per second
<Update Frequency="5"/> or ="6"/> six times a second.
<Update Frequency="7"/>, ="8"/> or ="9"/> nine times a second.

There is not much point going to higher numbers than 6 or 7 as this would appear to have little effect. By controlling the refresh rate, frames rates may be improved.
If a gauge does not need to run faster than once a second, then set the rate to 1.
Note: The total running time of each gauge cycle will depend on its complexity and length, but they are completely run extremely quickly.

2.4 Size

Earlier you learned that a background bitmap sizes the gauge and acts as the 'canvas' for placing other gauge objects. You can produce a gauge with no background bitmap. This might be a gauge that shows digital speed on a black background.

Looking at the original code that starts the gauge,

```
<Gauge Name="f117 aileron trim" Version="1.0">
  <Image Name="bitmap1.bmp"/>
```

The image can be replaced with,

```
<Size X="100" Y="150"/>
```

This tells Fs2004 that the overall pixel dimensions of the gauge are 100 wide and 150 deep. Objects can be placed on the gauge using the positional instruction for each <Element>, for example <Position X="50" Y="50"/> as discussed before.
There are ways of not having a background bitmap or size instruction and the gauge will still work, but I believe it is good practise to use one of these methods.

2.5 <Value> and more <String> instructions

These deserves a separate tutorial because in most cases, these instructions depend on calculation lines of code, and this can get quite involved.

2.6 <Mouse>

The sections discussed so far involve the main body of a gauge, and control the visual output and calculations. Another part of the gauge is the user interaction, where control or inputs are done with mouse clicks. A separate tutorial will deal with this.

2.7 Final discussion

Well, that's all for now.. The best way to put the above into action is with some complete examples. See further tutorials where complete gauges are shown and their inner workings explained. There are still a few instruction code types for the main body to learn about, but these are best covered by other tutorials or the examples. I hope I have shown enough code here to help the beginner to understand the more straightforward gauges.
Please bear in mind that in some cases, there are many ways of achieving the same effect. With experience, you begin to build up a portfolio of XML knowledge in your head, and the most optimised method will come to mind. However, I have found it impossible to remember every twist and turn. I often refer to previous gauges I have made to act as a foundation or template of a new gauge or to remind me what to do.
For those of you who have studied XML, you will know about the default GPS gauge. This gauge introduces a completely new raft of instructions and variables, and requires its own tutorial.
At first, this may all look a bit daunting. I actually became reasonably adept with XML after about 6 months. OK, not a five minute job, but then it takes time to learn anything properly. I also had a very good teacher, a man by the name of Arne Bartels. I have also learned useful snippets by regularly visiting the forums at Avsim.com..

Credits for additional information,

Rob Barendregt