XML Gauge Programming for FS2004. Chapter 3. Variables

Version 3.0

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 variables and their functions. If you read this tutorial third, 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 Introduction

This tutorial deals with numbers. After all, gauges are all about the manipulation of numbers. I talk about variables, that is, basically numbers that vary or code that is associated with numbers. The variables dealt with here are codes that extract a number from FS2004, inputs a number to Fs2004 or stores and retrieves numbers in particular ways.
There are basically six types of variables. I may not name these correctly for the sake of a beginner's understanding. They are,

a) Output variables
b) Input variables
c) G type
d) L type
e) P type and
f) s0 and l0

I have listed output type ahead of input, as the latter are easier to understand once you have understood the former.

1.1 Internal Variables (Output)

If you open the reference list Internal variables (output) file you will see entries like,

HSI STATION IDENT
HSI DISTANCE
HSI SPEED
HSI HAS LOCALIZER
Etc

Let's take one from the list and use it as an example. Let's take RADIO HEIGHT. BTW, make sure you write this variable correctly. I always copy and paste from the list to make sure. Also, they can be typed in lower or upper case.
I call these output variables because they read and provide a value or indication of what is happening in the inner workings of FS2004. When used properly in XML code, RADIO HEIGHT will provide us with a value. I hope you realize that this height is obtained by a type of radar device that measures the height between the ground and the aircraft.

We can see this number using the following line,

```
<String>%((A:RADIO HEIGHT,feet))%!5d!</String>
```

See Chapter 1 for the associated code that works with String instructions.

We should know that string lines will show a number on the gauge.

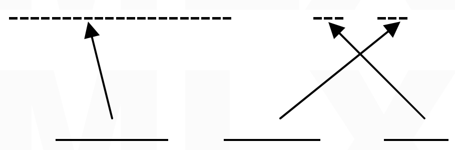
All the variables of this type are prefixed by 'A:'
After the variable is a comma, and then a unit of measure.
See the reference list Units
The unit of measure here is feet, but it could be, for example, meters. FS2004 will show feet or meters automatically according to the unit used.
As we should already know, the !5d! dictates that the numerical value shown will be an integer (no decimal places) and of up to 5 digits long. Remember from before, good practice with a height value would be to use !05! so we would get at say 500 feet, a display of 00500. Also remember, that the zero prefix cannot be used in an example where the value could go negative. We are safe with radar height as this will always be positive.

These variables can also be used for value instructions, like,

<Value>(A:RADIO HEIGHT,feet) 500 &lt;</Value>

I hope you remember from Chapter 1 that a value instruction could be obtaining the variable value to produce an event. In this example, we are saying that if the radio height is less than 500 feet, FS2004 will generate a 1 (one, meaning true), and if 500 or above, will generate a 0 (zero, meaning false). This line could be used with Case instructions so as to illuminate a warning light. Note the code 500 &lt; &lt; means less than. The opposite to this would be &gt. You can write the mathematical symbols < or >, but if you cannot remember which way round these go, then the first way is easier to remember. It maybe of interest to know that &lt; is the ASCII character for <.

The logic seems the wrong way round. Quite often with XML you have to think backwards. To clarify the code, the thinking is,
(A:RADIO HEIGHT,feet) 500 &lt;



When this value is less than this value.

OK, the above example provides us with a number.
Other output variables work with a unit called 'Bool'. For those in the know, this is short for Boolean logic. I'm not going to bog you down with this field of mathematics. All you need to know is that Bool is involved with producing 0(false) or 1(true). Let's take an example from the list again. We'll use,
LIGHT NAV
This variable is supplied purely to say whether the navigation lights are on or off. If used in a line of code, we cannot possibly get a numerical value. If we write the code,

<Value>(A:LIGHT NAV,bool)</Value>

This will produce a 0(false) or 1(true) according to the lights condition. This could be used with the Case instruction to control two bitmaps showing a switch in the on or off position.

A third output type produces a string (a line of letters).

<String>%((A:ADF1 IDENT,string))%!s!</String>
So you look at the list of output variables, and ask, which give a number, which give a bool and which give a string. Well some will be obvious (IMHO) and some will not. I do not think it is realistic to state them all here, but example gauges will help. When looking at example gauges, you should at least know what you are looking at.

Some considerations using A: type variables.

A: type variables are ONLY useable if applicable for the aircraft/engine type.
Example1:
If an aircraft does not have an Afterburner enabled, the variables A:TURB ENGx AFTERBURNER will remain 0 and is not useable
Example2:
For an aircraft with 2 jet engines (engine1 and 2) AND Afterburner Enabled, the variables A:TURB ENG3 AFTERBURNER and A:TURB ENG4 AFTERBURNER will again remain 0.
There are other examples where the aircraft may not contain certain features which nullify certain A: type variables.

1.2 Key Events (Inputs)

If you open the reference list Key Events (inputs) file you will see entries like,

ABORT
ADD_FUEL_QUANTITY
ADF
ADF_1_DEC
ADF_1_INC
ADF_10_DEC
ADF_10_INC
ADF_100_DEC
Etc.

Let’s take one from the list and use it as an example. Let’s take,
THROTTLE_FULL. Again, make sure you write this variable correctly, and the text can be typed in lower or upper case.
I call these input variables because they allow you to send information to FS2004.
Take a look at,

<Click>(&gt;K:THROTTLE_FULL)</Click>

When the line of code is activated by a mouse, the throttle will be set to full. The 'K:' denotes the variable is an input, or more correctly, a key event, because this input can also be achieved with keystrokes. The &gt; prefix is FS2004 convention to activate a key event instruction. There are no units associated with this variable type.
The opposite event would be,

<Click>(&gt;K:THROTTLE_CUT)</Click>

These can also be used in value instructions. Say you want the throttle cut in a certain circumstance, the line would be,
<Value>reason to want to cut throttle is true, then (&gt;K:THROTTLE_CUT)</Value>
The above are examples where key events are used to basically switch an event. There are others where you can input values to FS2004. Look at,

<Click>16384 (&gt;K:AXIS_THROTTLE_SET)</Click>

Variables with SET will literally set a value for input into Fs2004 for a particular variable. 16384 is a value that FS2004 uses to represent the full amount of movement. This example will set the throttle to full. The range for this value can be -16384 to 0 to 16384. If we used 8192, the throttle would be set at half throttle. A negative value will give reverse thrust (if the aircraft has this feature).
Again, this can be used in a value instruction. You may want to automatically set the throttle according to particular circumstances. There would be a line of code that may monitor conditions, such as taxi speed, and if the speed varies from a set value, the throttles would adjust.
<Value>code to monitor taxi speed and produce a value from 0 to 16384 (&gt;K:AXIS_THROTTLE_SET)</Value>

Some considerations using K: type variables.

Sometimes, "unexpected" events may crash FS2004 (possible bug).
Example:
If the aircraft has two non-jet engines, events K:TOGGLE_AFTERBURNER1 and K:TOGGLE_AFTERBURNER2 (NOT: -3 and -4) will CRASH FS2004.

1.3 G Type (Read WARNINGS at the end of this section)

G type variables are used to store values. If a value has not yet been assigned, they have the value zero by default. They follow the format,
(G:Var1) and
(&gt;G:Var1)

To store a value the code is,

```
<Click>1 (&gt;G:Var1)</Click>
```

A mouse click will store the value 1 (but this could be any numerical value). Then the G variable can be used in value or string instructions elsewhere in the gauge to maybe switch an event, thus,

```
<Value>(G:Var1) if = 1 then an event will be activated</Value>
```

You see them commonly used in,

```
<Click>(G:Var1) ! (&gt;G:Var1)</Click>
```

The '!' character means equal-to zero in FS2004. This code means if (G:Var1) is equal-to zero, a 1(true) is produced because the condition is true and (&gt;G:Var1) then sets (G:Var1) to 1. If the line has been clicked before, (G:Var1) will equal 1, and is not equal to zero, so a 0(false) is produced. (&gt;G:Var1) then sets (G:Var1) back to zero. So every time the code line is clicked, the value (G:Var1) changes 0, 1, 0, 1, etc. This can be used to chose between two bitmaps of an off and on panel switch.
G variables are gauge specific, which is, they do not transmit their value to another gauge. If you have two gauges, and both have a (G:Var1) in their code, they are not set to the same value. They will store their own individual values in the safety of their own gauge.

You can store up to nine values. From (G:Var1) to G:Var9). So (G:Var1) will store one value, and say (G:Var4) will store another.

Drawbacks. There are only nine available. They will not link between gauges. You have to remember how (G:Var5), for example, plays it part. They will not allow you to 'name' them to remind you what their function is. If you have a gauge with a lot of these, it can be a real headache keeping tabs on their function.
WARNINGS: If you change from windowed to full screen view, or vice versa, or reload the panel for any reason, G:Vars reset to zero. Therefore, gauge behavior can change.
A much better variable to use is the L type, (see section 1.4). These do not reset as above, but then this could be a drawback if you are developing a panel/gauge and require a reset. So you have to make a choice. Certainly, for 'normal' use, L types are probably the best to use. (See Credits section).

1.4 L Type

These are very similar to G type. They store numbers. However, you can name them so their use is readily known. They will also communicate between gauges if given the same name in each gauge.

They follow the format,

```
(L:anynameyoulike,unit) and
(&gt:L:anynameyoulike,unit)
```

Assigning a value is the same as G type. Note that one big difference between these and G type is the inclusion of unit text, which could be, feet, gallons, knots, etc. etc. or Bool, but you can also just use the text ‘number’. They are not actually storing units, but just the value itself. If the value to be stored is in feet, for example, it’s a good idea to use the unit feet to remind you. Bool is used if storing a 0 (false) or 1 (true), that is, an on or off condition.
L (and G) type is for storage only. They will not communicate with FS2004. An L (and G) type can be used to manipulate a value, but eventually the value is communicated to FS2004 using, say, a SET type key event variable.
However, they can produce dummy events. A dummy panel switch can be animated with two bitmaps and operated by a change to the L variable. The same L variable could be in another gauge that controls two bitmaps for an off/ on light that is associated with the dummy panel switch.
There is no restriction to the number of different L variables used, so long as they have different names.
Be aware that if the same name was inadvertently used in two gauges, they will be assigned the same value.

1.5 P Type

These are used in exactly the same way as Internal Variables (Output) that use the A: suffix shown in section 1.1. They are shown in a different, much shorter list P Variables and use the letter P instead of A. For example,

```
<String>%((P: SIMULATION RATE, number))%!5d!</String>
```

That’s about all I can say about these for now.

1.6 s0 and l0

This tutorial is a good place to talk about the next topic as it’s about another type of number storage. What more, you say, isn’t there enough of these already. I shall ignore that comment from the back of the class and continue. This a convenient way of storing and retrieving numbers but only in the same line of code. The variables used are, s0 (Sierra, zero) and l0 (Lima, zero), for example,

```
<Value>(code to produce a value in percent) s0 95 &lt; l0 5 &gt; &amp;&amp; (if true,
activate an event)</Value>
```

If a percentage value is produced, s0 remembers it. Then the value is compared to 95, and if less than 95, FS2004 produces a 1 (true). The line then compares with 5, but needs to retrieve the percentage value because Fs2004 has ‘lost’ this value as it is remembering 1 (true). The l0 repeats the percentage value and now the comparison is made. If the percentage value is greater than 5, then Fs2004 produces another 1 (true). The &amp;&amp; means that if both comparisons are true, that is two 1’s (true), Fs2004 produces a final 1 (true) and the rest of the code is activated. So, the original percentage value is saved, and then retrieved for later use. This does not have to be percentage values only. That was used for the example. Any number can be used. It maybe of interest to know that &amp; is the ASCII character for & (ampersand).
s0 and l0 therefore work together. Once s0 is set, l0 can appear any number of times in the line. These are independent of other lines of code. In another section of the gauge, s0 can be used again but will operate independently of any other s0 values in the gauge. There are fifty possibilities, from s0 to s49 and hence l0 to l49. So you could remember and retrieve, say 3 numbers, using s0,s1 and s2, and use l0,l1, and l2 to retrieve.

1.7 More on <Keys>

These were mentioned in Chapter 1 as being part of the main body of gauge code, but now we have covered variables, we can get involved with <Keys> in some more depth. In Chapter 1 I basically said,

```
<Keys>
<On Event="key event">activate some gauge code</On>
</Keys>
```

This "Monitors" a key event and when detected, it executes the gauge code.

Also, you can use,

```
<Keys>
<On Key="K"> activate some gauge code </On>
</Keys>
```

This "Traps" keystroke 'K' and when detected, it executes the gauge code.

The difference between "Monitor" and "Trap"? With 'On Event', the ‘key event’ and ‘activate some gauge code’ are both executed.
With 'On Key', the specified keystroke is NOT executed (so any event assigned to this keystroke is NOT executed) but the ‘activate some gauge code’ is executed.

Instead of using:

```
<On Key="Z">...</On>
```

FS also accepts

```
<On Key="90">...</On>
```

with '90' being the Windows key code for the 'Z' key.
This alternative syntax is important, because adding multiples of 256 to the key code allows you to also trap shifted/controlled keys:

Examples:
<On Key="90">...</On> traps 'Z'
<On Key="346">...</On> traps 'Shift-Z'
<On Key="602">...</On> traps 'Control-Z'
<On Key="858">...</On> traps 'Shift-Control-Z'
A list of all Windows key codes can be found in Peter Dowson's FSUIPC zipped package in the FSUIPC for Advanced Users.doc, available from
http://www.schiratti.com/dowson.html

You can use both On Event and On Key in the same <Keys> section.
<Keys>
<On Event="key event">activate some gauge code</On>
<On Key="K"> activate some gauge code </On>
</Keys>

Note that <Update Frequency="6"/> has been mentioned in earlier Chapters and determines the number of times a second the gauge code is run. However, the <Keys> section ignores this speed control and will still activate at the maximum rate of 55 mSec. Be aware of this as problems mat occur.

1.8 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.
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.
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: Rob Barendregt for his observations regarding the behavior of G and L type variables when gauges are reset. Further considerations regarding the use of A: and K: type variables. Further uses and information on <Keys>.