

XML Gauge Programming for FS2004. Chapter 3. Variables

Version 3.0

By Nick Pike

June, 2005

INDEX

1.0	Introduction.....	Page 2
1.1	Internal Variables (Output).....	Page 2
1.2	Key Events (Inputs).....	Page 4
1.3	G Type.....	Page 5
1.4	L Type	Page 6
1.5	P Type	Page 7
1.6	s0 and l0	Page 7
1.7	More on <Keys>.....	Page 7
1.8	Final discussion	Page 9

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 I0

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;&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 <<](#) means less than. The opposite to this would be [>](#). 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 [<](#) 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>(>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 > 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>(>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 (>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 (>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

(>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

(>G:Var1)

To store a value the code is,

