General Programming Tips

By George W. Denyer – GeoDen@blueyonder.co.uk

Here are some tips and useful rules to follow if you are planning to do some XML gauge programming. They follow general and common principles applied by professional programmers writing code that others must be able to understand. Hopefully, the amateur XML programmer will benefit from this advice since much of it is designed to make life easier.

First, I give you this bulleted list then I'll expand on each item.

- Get a language-sensitive editor (free).
- Use syntax highlighting.
- Indent your code.
- Use Hungarian Notation.
- Comment your code meaningfully.
- Choose variable and other names wisely and frugally.
- Don't write lines longer than your screen width.
- Don't be a "boastware" programmer.
- Tips from experience

Editor

The source code of nearly all computer languages starts out as text; you can therefore write code using a simple text editor such as Windows Notepad. That's if you want to make life harder, but I guess you don't, so an editor that is aware of the structure and syntax of the language is a wise first move. When I first started XML a few weeks ago I began by searching the net for an appropriate (and preferably free!) editor – I found one, it's called XmlSpy by Altova, the freeware version will be adequate for your needs and after using it for a little while you'll wonder how you managed without it. Some examples of what this and other language-sensitive editors can do for you, are as follows.

It will warn you about tagging errors e.g. <Element> without </Element> and prompt you to correct the fault.

Since it "knows" what you are trying to do, it tries to be helpful e.g. if you type & it will offer you & for quick auto-completion.

Many other features will be available; it depends on the editor of your choice. One of the hardest things to do is to separate a programmer from his/her favourite editor – that gives you some idea of the importance of one.

Syntax Highlighting

This is another facility usually provided by a good programmer's editor, it uses colour to highlight the language structure and so make typing errors easy to spot. An example is the best

way to show this, here is a line of XML code with an error, first in plain text then repeated with syntax highlighting.

<Image Name “MilRadioBg.bmp” Luminous=”Yes”/>

```
<Image Name "MilRadioBg.bmp" Luminous="Yes"/>
```

As you can see the ‘=’ sign is highlighted in blue so I think you’ll agree that the missing ‘=’ after <Image Name is much easier to spot with the benefit of highlighting.

Most editors allow you to choose the highlight colour scheme.

Indented Code

This is very important. The readability of your code is vastly improved by indenting, if you ever have to revisit code you wrote some time ago you’d be glad you indented. The main purpose is to clarify the structure and nesting levels of the code as can be seen below.

Unindented

```
<Element>
<Position X="196" Y="24"/>
<Select>
<Value>(L:TuneCom, bool)</Value>
<Case Value="1">
<Image Name="KnobUp.bmp" Luminous="Yes"/>
</Case>
<Case Value="0">
<Image Name="KnobDn.bmp" Luminous="Yes"/>
</Case>
</Select>
</Element>
```

Indented

```
 <Element>
   <Position X="196" Y="24"/>
     <Select>
       <Value>(L:TuneCom, bool)</Value>
       <Case Value="1">
          <Image Name="KnobUp.bmp" Luminous="Yes"/>
       </Case>
       <Case Value="0">
          <Image Name="KnobDn.bmp" Luminous="Yes"/>
       </Case>
     </Select>
 </Element>
```

"Nesting" is defined as a block of code matching the structure rules, in the example above, all the code between <Element><Element/> is nested, so is that between the Select and the Case tags. Although the code in the first example would work, I'm sure you can see that the indented second example is more easily read and nesting errors would be easier to spot.

Again, most good editors can handle nesting for you, XmlSpy for example, has a menu option called "Pretty print", if your indenting discipline has been lax, just click the option and all your code will instantaneously be nicely indented! Finally, other people will appreciate your well-presented code if they must read and understand it.

Hungarian Notation

This is the most efficient way of writing variable and file names. Some history is in order.

Back in the days of DOS (where I come from!) we had the 8.3 rule enforced upon us, file names could only be 8 characters long plus the dot and 3 character extension. This made us think carefully about names, we had to make them meaningful without wasting space with underscores and the like that convey no useful information. Modern operating systems like Windows allow us long file names so we are no longer restricted but that's not a reason to waste space with names like:

I_am_a_long_and_wasteful_file_name.xml

Or

I am a long and wasteful file name.xml

In many languages spaces are not allowed in variable names so Hungarian Notation was used to maximise readability without using unnecessary characters:

ThisIsMyVariable

Or

DigitalRadioAltimeter.xml

As you can see the HN variable and file names are just as easy to read as versions with underscores and spaces in them and have the added advantage of being shorter and less wasteful.

NOTE:

FS2004 XML like other languages has pre-defined variables that have been named with underscores and upper case characters these "defines" as they are generally called, must be used as they are - you have no control over their names.

Commenting Code

Another important part of good programming practice; comments are a way of making your code easier to understand, not just to others – mostly to yourself!

Commenting forces you to think clearly and logically about what you're trying to achieve and results in better programs. To be effective, comments must be meaningful and used where your intentions are not clear purely from reading the code. For example the code:

```
<Visible>(A:Circuit avionics on,bool)</Visible>
```

Preceded by:

```
<!-- Only visible when the avionics are on -->
```

Would be a useless if not downright stupid comment!

Comments are best used when doing something unusual or particularly clever that may not be readily understood by someone or at a later date, by you! They can also be used to help beginners in a language and may then sometimes state what may be obvious to experts.

Another powerful use of comments is to assist with debugging, there is no built-in XML debugger in FS2004 so we must use another tool – we can use the comment specifiers <!-- and --> The code below nested between the comments will not be executed when your gauge code runs in FS2004, therefore we can use this fact to "comment out" (as it's usually called) a suspect piece of code that may contain an error. Using this technique will allow you to find coding errors that we all make – you can isolate down to individual lines if need be.

```
<!--
   <Element>
     <Position X="71" Y="56"/>
     <Image Name="AltDigits.bmp" Luminous="Yes"/>
  </Element>
-->
```

IMPORTANT NOTE: Be careful with this if you use these comment specifiers as your normal delimiters. You can find yourself creating nesting errors when commenting out, but again, your trusty XML editor will help you catch these. Best thing is to avoid using the <!-- and --> specifiers completely for commenting, use them only for "commenting out". The highlight colour of your editor will make the "commented out" code stick out like a sore thumb for easy visibility and possible reinsertion.

Many languages use the // double slash specifier for comments; I'm so accustomed to them that I accidentally discovered that FS2004 XML allows their use! Actually, to be more precise, it ignores them and any following text, it's important to realise this because it means that although they can be used to comment, you cannot use them to "comment out" (they will be ignored!).

You can use the // characters in header comments like this:

```
<Gauge Name="Altimeter" Version="1.0">
//-----------------------------------------------------------------
// This gauge does blah blah
// Copyright Joe Blow
//
// Anything you like etc, etc.
//-----------------------------------------------------------------
```

NOTE: the header comment is usually before the start of any code but FS2004 complains if it comes before the Gauge tag so place it as shown.

Another use is EOL (end of line) commenting like this.

```
<Nonlinearity>
  <Item Value="0" X="100" Y="48"/>      // You may want to explain the degree
  <Item Value="5" X="100" Y="152"/>     // of nonlinearity in these lines
</Nonlinearity>
```

FS2004 will ignore all these comments as the gauge code is executed. Comments are useful; make the most of them!

Variable Names

Some of this was covered in a previous section but here are some more examples.

Make variable names pertinent to their function e.g.

(L:RadarWindowActive, bool) is a good name that tells you it's purpose but the name (L:AnyCrapWillDo, bool) would work just as well and FS2004 wouldn't care, but you should!

I once knew a programmer that loved fruit, he revelled in creating variables like BigJuicyPear, RipeBanana and TastyApple, of course, the program we worked on had nothing whatever to do with plants so our fruit-lover had to be shown the error of his ways!

Try also to avoid very long names (why type more than needed?), (L:RadAltCtrl, bool) is just as understandable as (L:RadioAltimeterControl, bool) and a lot shorter, just be sure not to sacrifice readability for ease.

Long Lines

No, not fishing – coding!

This is one of my pet hates! Why would you want to write lines of code that force people to scroll sideways in order to read them? This seems to be is very common in examples of XML code

around, there is no reason for it and again, readability suffers. Choose a length less than the width of your screen at your working resolution and stick to it; this can be set as an option in many editors.

I have worked in software houses in the UK and the USA, all of them had rules about line length that programmers had to comply with, I think FS2004 XML coders would do well to adopt a similar standard, it would benefit everyone.

BoastWare

This is only my personal opinion (although I'm not alone) but this practice irks me – it smacks of childishness; perhaps I should define it.

BoastWare is the practice of writing code in a freeware product that includes the author's name and perhaps other worthless information in the user interface (visible on screen) part of the program or gauge in this case. By all means, you should include your name and other details in your source code (XML file) so that credit be given where due, but please, don't try to show how clever you are by making your name part of the gauge! In XML gauges the source code is open and therefore easily changed by the user and I'm certain this would happen if I put out a gauge with my name all over the face of it! In a gauge written in a compiled language it's not so easy because you don't get the source. But it can still be changed and can be quite fun to see the reaction of the original author if he/she happens to see the "new" version of their creation.

You never see "boastware" versions of professional programs, hard to find easter eggs, yes, but that's it – where are the names of the team who wrote FS2004? Pro's do one of two things; they sell their work and guard the source or they give it away as is, not crippled or uselessly adorned.

Tips

Here are some tips and things I've found to be true after 20 years of software experience.

Follow Einstein's maxim; keep things as simple as possible but not *too* simple.

If you're having difficulty coding your design, stop – you are probably going about it the wrong way, rethink the design.

Get help if you need it, but try wherever possible to work things out yourself, you'll learn faster; it's no good having someone write code for you that you don't understand! The FS2004 GPS code is a good place to study and experiment with copies of the XML it contains.

Bear in mind that most importantly, the programmer must have a clear idea of what is required *before* coding commences. In a simple gauge this may mean just keeping what's needed in mind, in complex projects you may have to write down your intentions or even make a flow chart of how things will work. Believe me, you will save much time and frustration by working this way, not to mention umpteen versions of scrapped code!

Software, like painting, is a creative endeavour; there are many ways to achieve the same end result. If you asked two artists to paint a horse the results would look like a horse but would not look the same as each other, software is like that and the best software achieves the result with the

minimum possible code. As you become proficient at coding you will find yourself getting good results quickly with elegant code, looking back at your early efforts will make you laugh – laugh heartily, it's a good sign.

Finally

I have tried to keep computer and programming jargon and terms to a minimum because I know a beginner finds them off-putting, but a few things can't be avoided. If the reader finds that any of this needs clarifying, let Don and Nick know, maybe we can come up with a glossary of terms.

I hope this article will be of some help to all you coding beginners – keep writing the code and doing the graphics (art is my weak point) you never know, maybe one day someone may be willing to pay you for it!