Les Chroniques du Campagnol
Season 2 Episode 4
Mercator - Choosing a development platform

April 9, 2022

While it is very possible that developing a Windows Service using Python or Typescript could be fun, I decided to use a platform that is easier to use for such endeavour. So I catalogued the languages that we use, have used or could use at Data Innovations for software development in a Windows environment.

1 Selecting a programming language

First of all, I need to choose the language I will use. From there the platform will be obvious.

1.1 Vb 6

No. Just... no.

1.2 Delphi

Could be, but the Delphi platform is not portable to Linux, unless we use alternatives like Lazarus. We could use Oxygen to target the .Net environment instead, but then why bother? Oxygen is so different from Delphi that there is no hope to ever re-use the existing code from LPM or EE. And the question remains: do we really need or even *want* to reuse code from those?

1.3 C

Too low level. Maybe I will write some interop code one day (I think I still have C♯ code somewhere that interops with the LPM Driver Manager) but not now.

1.4 C++

Much better than C, but I no longer want to deal with wild pointers and unexpected type conversions. A reasonable alternative to C++ exists, it's C♯.

1.5 Rust

A little better than C++ regarding memory management, but you still have to do it yourself. Rust takes you by the hand so you don't fall, but it's still something I no longer want to do.

1.6 Java

I dont' like Java. While it is very similar to C, its ecosystem is irremediably flawed. I still strongly believe that when you are in doubt on how to implement something, look at how Java did it; that implementation is a good example of what **not** to do.

1.7 Go

No Go.

Go, for me, has several sins, one of them being deadly. Oh and one heresy also. Yes, this is very opiniated. Go is an opiniated language (everything I do is genius, everything else is so silly that the compiler goes to great lengths to make it illegal to use).

The first sin is trying to have a syntax like C (look, I have braces!), but then not at all like C at the same time. Please, language designers, stop trying to spare three keystrokes by declaring functions with the `func` keyword instead of `function`. Wanna spare keystrokes? Do it the C way, where there is *no* keyword needed to declare a function.

The deadly sin is trying to force style conventions into the syntax. It is a syntax error not to put the brace at the end of the previous line? It is (almost) a syntax error to indent using spaces? Come on! Grow a pair (of neurons, of course).

And then there is the heresy to make the semicolon optional at the end of statements. For what? Save one keystroke? Or are you trying to hold the claim that you actually **improve readability by removing the terminating semicolon?**

That claim is plain stupid the semicolon is there not to tell the compiler that the statement has ended the compiler is smart enough to deduce that by itself the semicolon is there to tell **our brain** that the statement has ended and there is no need to read the next line for an optional continuation having a required semicolon at the end of statements helps readability instead of hindering it but as always they think about the ease of writing the code once instead of thinking about the ease of reading the code several times.

So what?

My pet project. My time. My rules. I don't like the rules they try to impose on me, I don't use their language.

But you want to impose your rules!

Actually, no I don't. I prefer my way of indenting code, I use it in my code. I don't force other people to do the same, in exactly the same way I don't force them to admit that considering that the Earth orbits the Sun is more logical than the reverse. I recognize that the way to go is to simply write a VS Code extension that shows the code the way you like it, instead of forcing your choice on other people. That way, you could look at code that I wrote using my indentation and it would be displayed with the braces placed differently. Even better, I could author code with my indentation and publish it with yours. Please do so if you like.

1.8 C♯

A definite yes. There is no need to hide it, I like that language and I believe its the best compromise you can make. Especially it does not try to hide its C/C++ heritage.

Except of course when it is multiple...

Indeed, but at least they do not try to make you believe that multiple inheritance is inherently[1] bad. They did not include multiple inheritance because its paradigm differs depending on the language you use, and the .Net platform wants to be language neutral. Multiple inheritance is implemented completely differently in C++ and in Python, so you would not be able to mix the two languages into a single application.

I find this very unfortunate, but reasonable. Of course, they could have added mixins instead; there is hope that someday mixins will be available. Anyway, one of the first C♯ tools I wrote was precisely a way to have compile time, statically typed, mixins available in C♯. This took advantage of the *partial class* feature, where the implementation of a single class could be spread across multiple source files. The took was called MyXin and used a pre-build step to inject additional source files into the implementation of a class. I need to dig into my archives to retrieve that and see if it still works. Also, I'll have an episode explicitly talking about auto-generating the code you need.

[1] Yes this was done on purpose.

1.8.1 Pascal Case

The multi-language paradigm of the .Net platform has forced its engineers to think about openness and inter-language compatibility. This leads to choices driven by logic instead of personal preferences. Yes, I'm talking about `PascalCase` instead of `camelCase`.

Except in parameter names

Yes, again proof that logic does not permeate everything. When the .Net platform emerged, the names of the parameters were not part of the public interface of a function, so they did not formally need to make them PascalCase. I find this highly infortunate.

1.8.2 Code indenting

And did I say yet that I like the positioning of braces?

1.9 Prolog, F♯ or Lua

Of course, not for the main application. But maybe as a scripting language (see for example Moon♯)

2 Isn't .Net inextricably bound to the Windows environment?

It was. A long time ago. That is no longer the case. The .Net platform has been completely rewritten and is now totally platform agnostic. Based on prior work (Mono) the platform is now divided in two independant implementations, one that runs virtually everywhere (.Net Core), and one that contains the platform-specific components (.Net Framework).

To prove that, I will install the C♯ compiler on my Raspberry Pi and compile a little program.

So strictly speaking, I'm not targeting the .Net environment; I'm targeting the .Net Core environment. This means that for the moment I will not be able to write GUI native applications (actually, writing GUI applications in the .Net Core environment is less and less restricted, especially since MAUI is in the pipeline). For the moment, I will target .Net Core 6.0, but I might try a .Net Core 7 preview someday.

3 Code Reuse

Another advantage of using C♯ is that we can reuse existing code or at least techniques, because many of our applications already make use of some C♯ code:

- The *Software Security Key* library is written in C♯ and contains several useful tools to access the **WMI** (*Windows Management Instrumentation*) subsystem. My application will need to query WMI probes to obtain some information.
- The *Instrument Manager Interop DLL* library is also written in C♯ and provides access to Windows features such as starting, querying and stopping Windows Services.
- The *LabGPS ETL Driver* also uses a C♯ layer to communicate with LabGPS and more generally the LabX infrastructure. It has tools to establish communication with the LabX server, upload data, etc.

4 And then there is Scripting

As I said earlier, we need to be able to script some decisions. There are a few scripting languages available for the .Net platform, but the best seems to use C♯ itself.

And we can invoke the compiler from within the application, give it access to the model libraries, so that it can compile additional code and inject it into the runtime.

We have access to compilers for C♯, Visual Basic, etc. A good choice also is F♯ as it is a functional language, and rules are very good candidates for functional programming.

One of the problems of including a fully fledged language as script platform is that your program can do absolutely anything it fancies. And since the service is a local administrator, that can be dangerous. We'll see how to remedy this in the future. I first opted for using AppDomains but these are - of course - now obsolete...

5 What we achieved

I managed to choose the language and platform I will use for development. I also managed to sneak in my usual rants about programming languages and features I don't like.