Visual Studio

An Overview for those
Coming From VO

To Start

For those coming from VO I would advise that you forget all about how it worked. This does NOT mean forget about VO syntax – indeed the X-Sharp development team has done all it can to preserve the VO/Vulcan syntax, but you will be transitioning into the world of .Net and using Visual Studio to allow you to operate within that world as easily and efficiently as possible.

The reason I say forget about how VO worked (repository and things) is that X-Sharp is not predicated on VO but, just as VO was, the workings of a compiler. (albeit implemented slightly differently as we shall see in a moment).

I hope all this will become clear as we proceed.

The Overall Picture

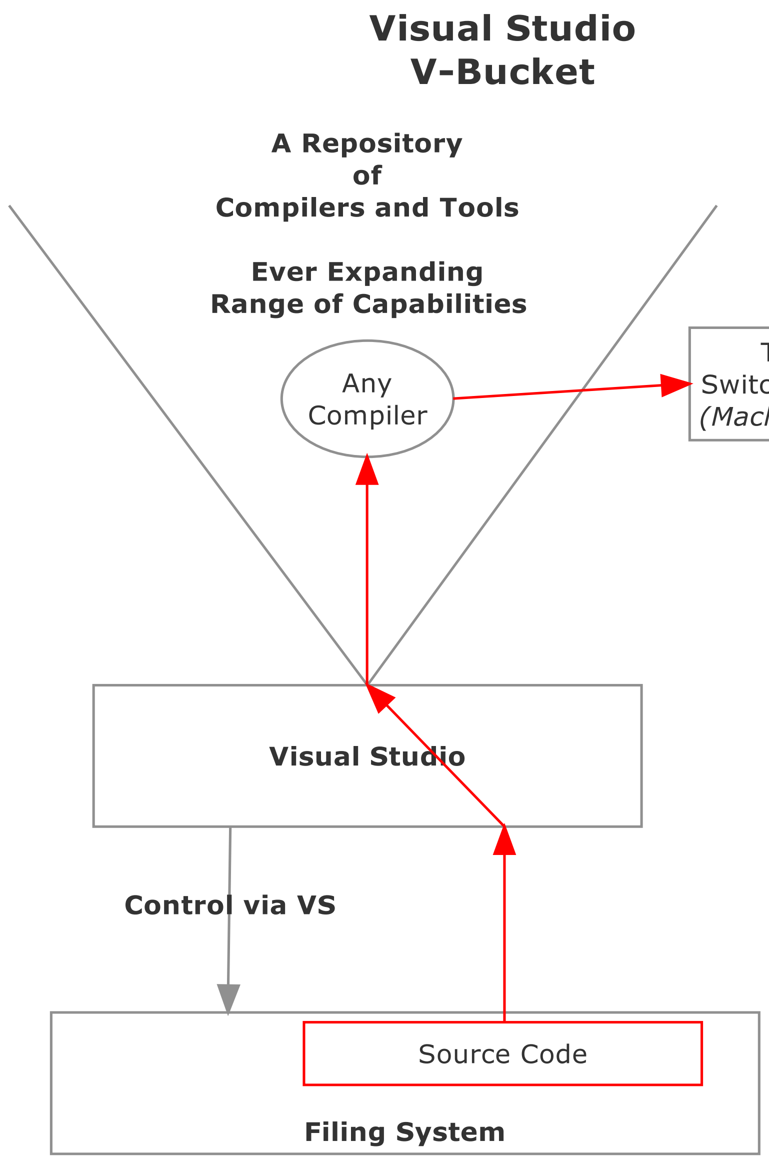
Let's start with the computer itself: it is a machine; and as with any machine it must operate within strict limits of adaptability which are physically "built into" the machine and become a characteristic of that machine.

A computer program runs (adapts the electronic switching etc) within those limits. It cannot change them.

This means that any compiler translates a sequence of logic (different programs) into "cast in stone switching logic" – the detail of this switching logic is dictated by the Central Processing Unit (CPU) of the computer. But you knew all that anyway!

Now, your program, whatever the computer language, is hosted in the filing system. This means that there must be a controlled route (allowing information from different files to be purposefully combined) through from that filing system to an appropriate complier, which itself is likely to be hosted on the same filing system. Again, you knew that anyway!

At top level Visual Studio acts as a controlling host to all the above. I try to show it in the form of a V-Bucket in the figure below:



With the above firmly in mind, well move on and look at what is the essential difference between the VO you are coming from and the .Net you are jumping into.

The Essential Difference of .Net

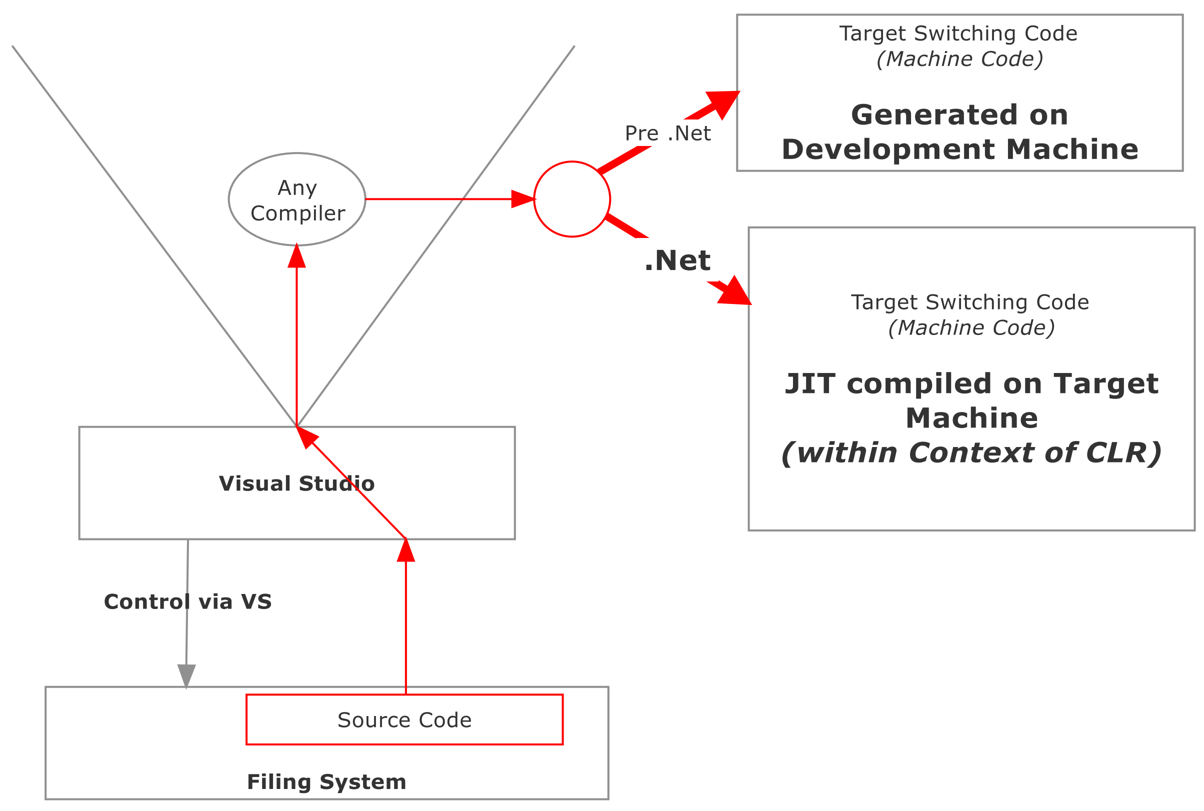
To illustrate this, we have to consider something we have not had to consider before. It is <u>where</u> things are done.

Pre .Net compilers generated code to run on specific machine architectures. .Net changes that: .Net compilers generate code that runs in the Common Language Runtime (CLR) which runs on the Target machine. It is <u>on the target machine</u> that the ultimate machine switching instructions are generated.

This is done as your program runs, on an "as required" basis, by the "Just-in-Time" (JIT). This offers many operating benefits which we'll see as we progress. But perhaps (arguably) the greatest is that it abstracts away details of target system hardware.

I have tried to illustrate this by a slight modification to the above diagram:

The Essential Difference



It will be useful, at this point, to briefly consider computer operation and the process of program development.

With **a few obvious differences** it is exactly the same interaction that exists between you – the reader and me – the writer.

As I write this I have no idea when or if anyone will read it. Of course, I hope so. But one thing is sure: no one will read it until I've finished writing it. Nor will they read it until I put it into the X-Sharp forum (or somewhere similar) and give them the opportunity to read it. Nor will they read it until something prompts them to do so.

Now as I've been writing this I haven't committed to paper (computer) everything I written without a bit of cut and paste and a few phraseology modifications on the way. Changes are/were based on my thinking "have I made this clear enough"? "Could it be understood"? How far I succeeded is not something I can answer.

This process is exactly the same as the process of software operation. During development we are producing code to write and read files all over the place. It is **Visual Studio** (just as the VO IDE), that provides a tool to ensure that this is done in a controlled manner, and that the changes we make are totally consistent.

In the writing/reading analogy inconsistencies can easily arise: we are relying on a fallible human reader spotting inconsistency and could easily end up with an adjective masquerading as an adverb. And this is one obvious difference – a human reader would likely overlook such things and derive the same meaning from the text. A computer program would crash.

Remember that we are writing code that will be read. We are not writing code instructions that "drive" anything. Ultimately it is the CPU "reads" switching instructions first and follows it through in a read/write process. Thankfully, that's all we need to know about the electronics.

In the case of managed languages, we are writing code to be read by the Common Language Runtime (CLR).

As stated computer operation is a read/write process throughout. Obviously, it can't read before what it is reading has been written. So what triggers or gives it the incentive to read in the first place and what is there to read.

The answer, of course, is the incentive to read is given when the computer is switched on. The first thing it reads is already written in non-volatile memory, and read. This then "reads in" the operating system. Then, under user control, we'll "read in" / load Visual Studio.

The Road Ahead

We're starting a journey that will take us not only into X-Sharp but into a whole range of features and capabilities available within .Net itself. Do not worry: understanding it is not as daunting as you may at first think and actually developing practical programs is tightly controlled by Visual Studio giving valuable and immediate feedback before things get out of hand.

Visual Studio - the Holdall of our Development Efforts

A VS solution may be looked at as the embracing of numerous files, exercising control over them, organising them, modifying their contents in any way that suits us. We'll look at the details of this later. The point at this stage is to note it is all done under the control of VS which ensures things don't get in a muddle. If woolly thinking on our part gets around all the safeguards, rest assured we'll be alerted to the fact somewhere along the line.

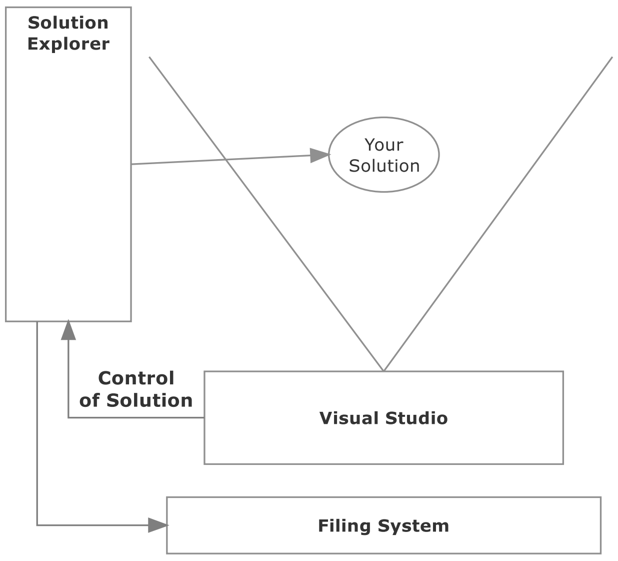
Visual Studio is just one IDE having the same role as the VO IDE. But with far wider embrace.

Unsurprisingly, since we are generating code for the CLR to read, the way our program needs to be structured is dictated by the way the CLR reads. This is not sequential – it is more like reading a technical book of instructions and jumping around for more detailed bits as needed.

To do that it introduces the concept and practical implementation of a Solution File capable of directing input in a controlled way to all files used (by MS Build) to aggregate all files in an application.

I've tried to show how a solution and its control (Using Solution Explorer) fits into the overall picture in the diagram below:

Controlling the Organisation of our Solution



With the above in mind we are in a position to move on further into how VS gives us the flexibility to develop file-based solutions,

To Be Continued

(Please note that all I have written above applies to any .Net development running under control of Visual Studio. I am not qualified or in a position to take things further in respect of X-Sharp. I may guess but others here will be far more able to go into X-Sharp than me.

I can go further, in the same vein as above, to try and bring in a common understanding across the board, and would be happy to do so if you feel it worthwhile)