Dot Net

An Overview for those
Coming From VO

Development For dot Net

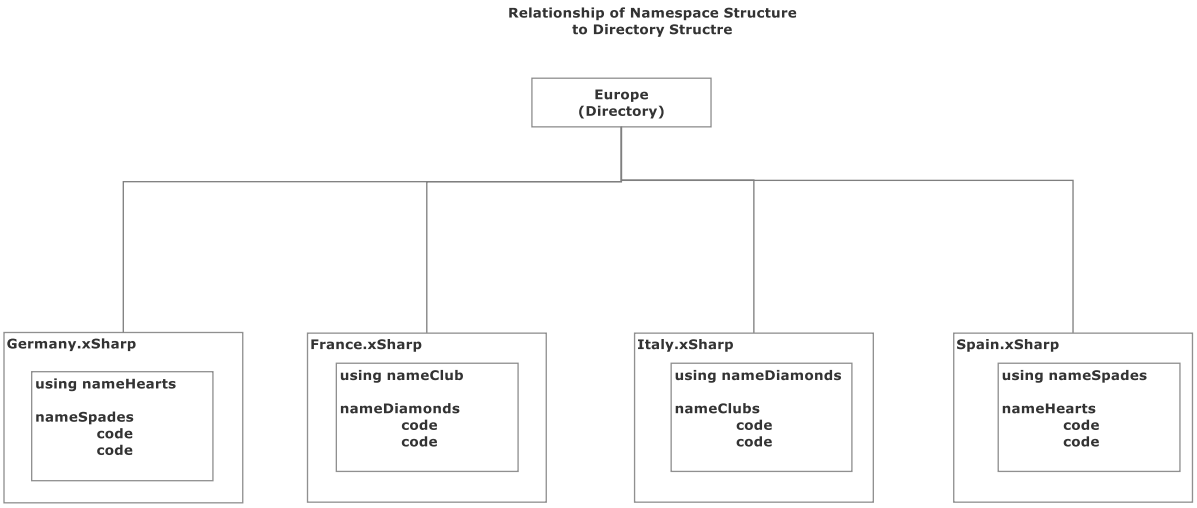
Development coding for .Net is File Based. Unsurprisingly that means it is hosted in the Directory/ File System with which we are all familiar. In a sense it is not unreasonably to consider it as the repository of VO.

But that structure is far from convenient when, at the end of the development process, we need to produce one readable file for submission to the CLR.

Some way of identifying “grouping” code so that it can be extracted logically, unambiguously and automatically from the file structure is needed. Furthermore, it is up to us, as developers, to identify, in our code, how we want things to be brought together.

We do all that with the .Net concept of namespaces (which embrace our code), and the keyword “using” which references them. This results in a sort of “holding” structure which sits atop of the filing system.

By judicious naming we can illustrate the situation, as shown in the diagram below:



.Net is based on Types

This is true, but do not confuse .Net Types with types you are already familiar with. Everything in .Net is considered to be a Type. It is the CLR concept of a type. And there is reason for it – that is to do with the way the CLR works. Basically, it is a way of packaging things that work the same way.

The CLR is extremely complex. To fulfil its primary task of facilitating (JIT) compilation on the user machine, it has to regenerate our program. It does this from highly efficient compacted code.

Fortunately, we don't need to go into the details at this time. But we can easily understand the principle by relating the process to what we do every day of the week. We could go to the baker, armed with ingredients and get him to bake us a cake, we could go to the doctor and get him to give us some pills (to cure the headache we've got from reading this), we can take a method to the CLR and get it to process that and so on.

In fact, the Types that can be accepted by the CLR are fairly limited: class, fields, enumerations being among them. These will all have been "mapped down" from our source code.

Much of our source code will have become superfluous by the time the CLR output is read by the JIT compiler. Its instructional meaning will be inherent in the way in which the read is done. This means the operation is quick and limited to those instructions that are required at that moment.

It is wrong to think that this compilation process must necessarily impact performance. The first time it takes place it will, but after that an already compiled version of the code, derived from an Intermediate Language (IL) will be used. Any performance hit is not likely to be a concern in an X-Sharp development. (and there are ways round it anyway)

To be contined