



Look at X Sharp!

*Eric Selje
Salty Dog Solutions, LLC
Madison, WI USA
Voice: 608-213-9567
Website: www.SaltyDogLLC.com
Email: Eric@SaltyDogLLC.com*

Would you be interested in a product that compiles your existing Visual FoxPro projects into .NET code? Well, that product doesn't exist, but there is a compelling product called X# that is working towards that goal. They've got a mature product that compiles many dialects of xBase to .NET already, and they're now actively working on VFP syntax.

In this session we'll look at X# from this Visual FoxPro developer's point of view.

You will learn:

- A brief history of the xBase language family tree
- How close a real FoxPro to .NET compiler is
- Whether this is a product you can move forward with, given your current skill set
- Whether this is even a direction you want to go

Preface

This whitepaper cannot be a complete documentation about all things X#. My intended audience is the Visual FoxPro developer who's looking at options for either new development or a candidate for migrating their existing projects. I hope after you read this whitepaper as well as my own conclusions you'll be more informed about what X# is and what it is not and will be able to make a decision about whether this is a direction you want to pursue.

A Brief History of xBase

In the beginning, there was Vulcan.

Created in 1978 by Wayne Ratliff at JPL Laboratories to run his football pool, Vulcan was the ur-language of what would become the bread and butter of every developer who used any of its descendants. For Vulcan begat dBase, which begat FoxBase, which begat FoxPro, which begat Visual FoxPro, which is why we all gather here at Southwest Fox today.

It's important to understand a little of this history if we want to understand X#, because X# descends from a different branch of the family tree than Visual FoxPro does. When you see terms like Vulcan (not the same Vulcan as above) or VO mentioned in the docs but have no sense of xBase history, it's difficult to orient yourself. So let's take a brief look at the history of xBase.

Interpreters

The original Vulcan by Wane Ratliff was licensed to Ashton-Tate, who named their derivative dBase II.¹ Vulcan ran on CP/M, but Ashton-Tate also got versions running on Apple II and DOS. dBase III added support for VMS and Unix as well, and with the addition of ASSIST in dBase III+, the product had really gained traction. This is actually where I became aware of the product, and probably many of you as well.

It wasn't long until competitors began cloning dBase. We all owe a great debt of gratitude to Fox Software for creating a dBase III+ clone called FoxBase+. I can still remember taking my existing dBase PRGs and running them under FoxBase for the first time and being absolutely *amazed* by the speed difference. It was as if someone brought up the anchor and just let the program run!

Meanwhile Ashton-Tate came out with dBase IV, which we all know was an absolute dog, and FoxBase thrived. A desperate Ashton-Tate sued Fox Software for cloning the dBase language, which they eventually lost because they didn't actually own it in the first place. Remember I said that Wayne Ratliff merely licensed Vulcan to them? The court decided that Ashton-Tate had no case because dBase itself was derived from the work that became Vulcan, and not Vulcan itself. It's a little confusing and made even more so when you add in the fact that by the time the lawsuit was concluded, dBase had been sold to Borland who was *also* being sued by Lotus for cloning the look and feel of Lotus 1-2-3. So at that time Borland was trying to argue both sides of very similar suits!

Borland (renamed Inprise) eventually sold dBase to KSoft, a company run by a man who made a ton of money selling his dBase apps and didn't want to see the product die (similar to VFP Advanced, but that's a different session). KSoft was renamed Databased Intelligence, then simply dBase LLC. You may be surprised to learn that dBase is still around, and dBase LLC just released dBase 2019.

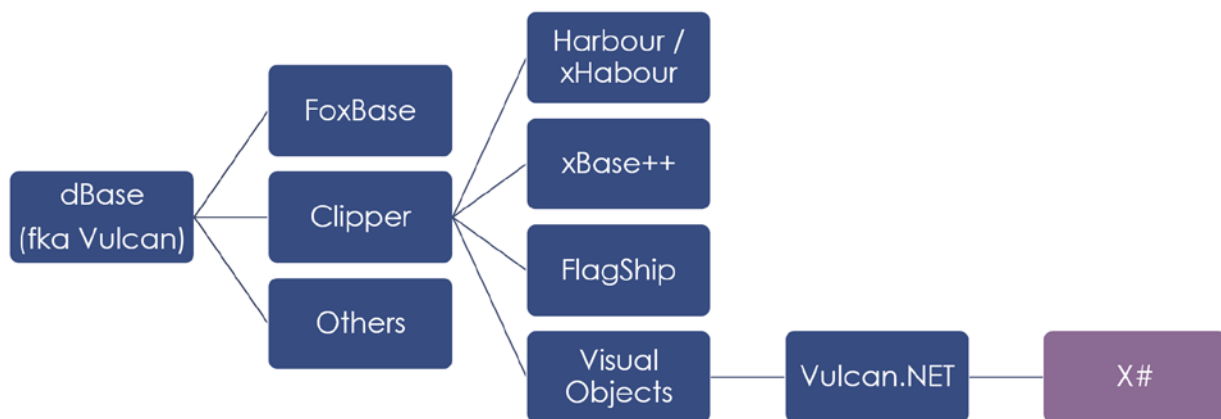
Compilers

Nantucket Software took a different approach to their dBase clone. Rather than a faithful reproduction of dBase and its runtime interpreter, they decided to create a true linker and compiler for the language. Their product, Clipper, was also very successful and eventually was bought by massive conglomerate Computer Associates (CA) and renamed CA-Clipper to match their other products. The last version of DOS-based CA-Clipper was released was released in 1997, but by that that Windows became mainstream.

CA created a corresponding Windows version of Clipper they codenamed Aspen but which eventually was released as Visual Objects (VO). Remember that abbreviation, VO, because that's the origin of X#. In CA's liquidation, Visual Objects was sold to a company called GrafX and continued until its last release in 2012.

Clipper inspired multiple descendants¹. Some of the more notable ones are Harbour, an open source, multi-platform clone created by folks who weren't happy with CA's proprietary stewardship of Clipper. Another was Alaska Software's xBase++, which we're familiar with from their participation at Southwest Fox for many years. A third product called FlagShip was geared towards Linux/UNIX developers, with a Windows version eventually released as well. Each of these created small variations of the base dialect

But the most interesting descendant was GrafX's own attempt to rewrite the Clipper compiler from scratch to take advantage of the .NET framework on Windows. This endeavor was dubbed Vulcan.NET as a nod to Wayne Ratliff's original product. If you never heard of Vulcan.NET, it may be because (the now-defunct) GrafX didn't put a lot of marketing behind it. This frustrated a core group on their development team, who left in 2015 to start a new effort. The result is X#.



¹ Quicksilver, dbXL, Multibase, Recital, Eagle, Arago to name a few.

First Look at XSharp

X# is not Visual FoxPro. This is not a product that will allow you to load your existing projects, recompile them, and run flawlessly like FoxBase was to dBase. To paraphrase the immortal words of David S. Pumpkins, "It's its own thing!"ⁱⁱ Its closest kin is really C#.

As a language, X# is in itself another xBase derivative, which they call "Core." But because XSharp has a long ancestry, its developers have included support for different dialects such as its immediate predecessors Vulcan.NET and Visual Objects, as well as more distant cousins Harbour, xBase++. There's a concordance on the X# site to show differences in dialects.² The developers are adding support for the Visual FoxPro dialect, but as of this writing very little of your code will run without modification. Now before you get too discouraged about that please read on because you were probably going to have to rewrite your code a bit anyway, and you may find that the X# way of doing things is preferable.

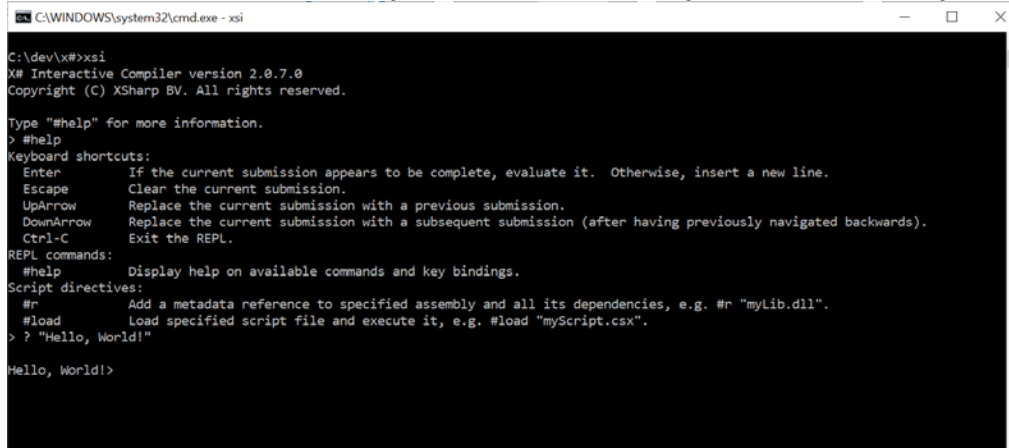
The Development Environment(s)

Let's begin our look at X# with the very simplest things you could do, "Hello, World". In VFP we could fire up our VFP9.EXE, type

```
? "Hello x#!"
```

into the Command Window and see the result (not shown because you get it I'm sure).

X# doesn't quite work like that. There are three ways to run X# code. You can use Visual Studio or the included XIDE to create compiled applications, but neither has what we would call a Command Window. Visual Studio has an "Immediate Window" which comes close but it doesn't support X# syntax yet. I'll get to those two in a bit. To get a Command Windows as we're used to, we'd use **XSI.exe**, an implementation of Windows Command Line³ that comes with X#. If you're familiar with Python's REPL utility, it's a lot like that.⁴



```
C:\WINDOWS\system32\cmd.exe - xsi
C:\dev\x#\>xsi
X# Interactive Compiler version 2.0.7.0
Copyright (C) XSharp BV. All rights reserved.

Type "#help" for more information.
> #help
Keyboard shortcuts:
Enter      If the current submission appears to be complete, evaluate it.  Otherwise, insert a new line.
Escape     Clear the current submission.
UpArrow    Replace the current submission with a previous submission.
DownArrow  Replace the current submission with a subsequent submission (after having previously navigated backwards).
Ctrl-C     Exit the REPL.

REPL commands:
#help      Display help on available commands and key bindings.

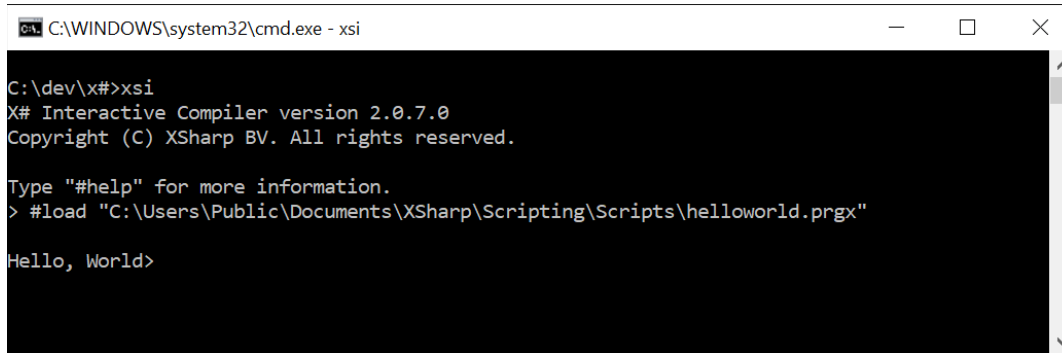
Script directives:
#r         Add a metadata reference to specified assembly and all its dependencies, e.g. #r "myLib.dll".
#load      Load specified script file and execute it, e.g. #load "myScript.csx".
> ? "Hello, World!"
Hello, World!>
```

² <https://www.xsharp.info/help/dialects.html>

³ <https://devblogs.microsoft.com/commandline/>

⁴ REPL stands for Read-Evaluate-Print-Loop: Read the Input, Evaluation the Input, Print the Output, Loop back to the beginning.

XSI is a nice way to get used to trying out some of the X# commands, and you can also run PRGs (scripts) you created with your favorite text editor, but it's not an editor itself.



```
C:\WINDOWS\system32\cmd.exe - xsi
C:\dev\x#\>xsi
X# Interactive Compiler version 2.0.7.0
Copyright (C) XSharp BV. All rights reserved.

Type "#help" for more information.
> #load "C:\Users\Public\Documents\XSharp\Scripting\Scripts\helloworld.prgx"
Hello, World>
```

Note that the extension on this sample is PRGX, which X# associates with XSI.exe in Windows upon installation, but there's nothing else special about it. You should not run entire applications in XSI.

XIDE

The first "real" IDE that you may want to use with X# is called XIDE. (I'm not quite sure how one would pronounce that but I'd like to think it's similar to "Excite".) XIDE is an optional component when you're installing X#. One very interesting fact about XIDE is that it is written and compiled in X# itself, so you can see the potential of the language! It's very quick to load and has a lot of nice features including a project manager, form designer, comprehensive help files for both itself and the X# language, and a debugger.

Despite having a Project tab, you cannot load your existing Visual FoxPro projects. X# has no idea what PJX files are (or SCX, FRX, LBX or any other binaries for that matter, but we'll get to those later). Projects in XIDE are collections of Applications. In Figure 1 you can see the SWFox2019 Project is open, and it contains two applications: DemoForm1 and VFPClasses.

In the Project Explorer you can see another difference, the <References> branch of the project's treeview. References are external libraries that your project uses, somewhat akin to the class libraries that you SET CLASSLIB TO in VFP. Because X# is a .NET Language, you can reference and use anything in the .NET Framework, as well as 3rd party libraries. We'll get to that later when we discuss the X# language itself.

You can also see similarities in XIDE to Visual FoxPro. The debugger, toolbox, properties, and locals tabs should all look familiar (**Figure 2**), and the Form Designer is pretty easy to figure out. There is intellisense. All in all XIDE is a very straightforward environment that Visual FoxPro developers with no prior experience should be able to grasp fairly quickly.

Look at X Sharp!

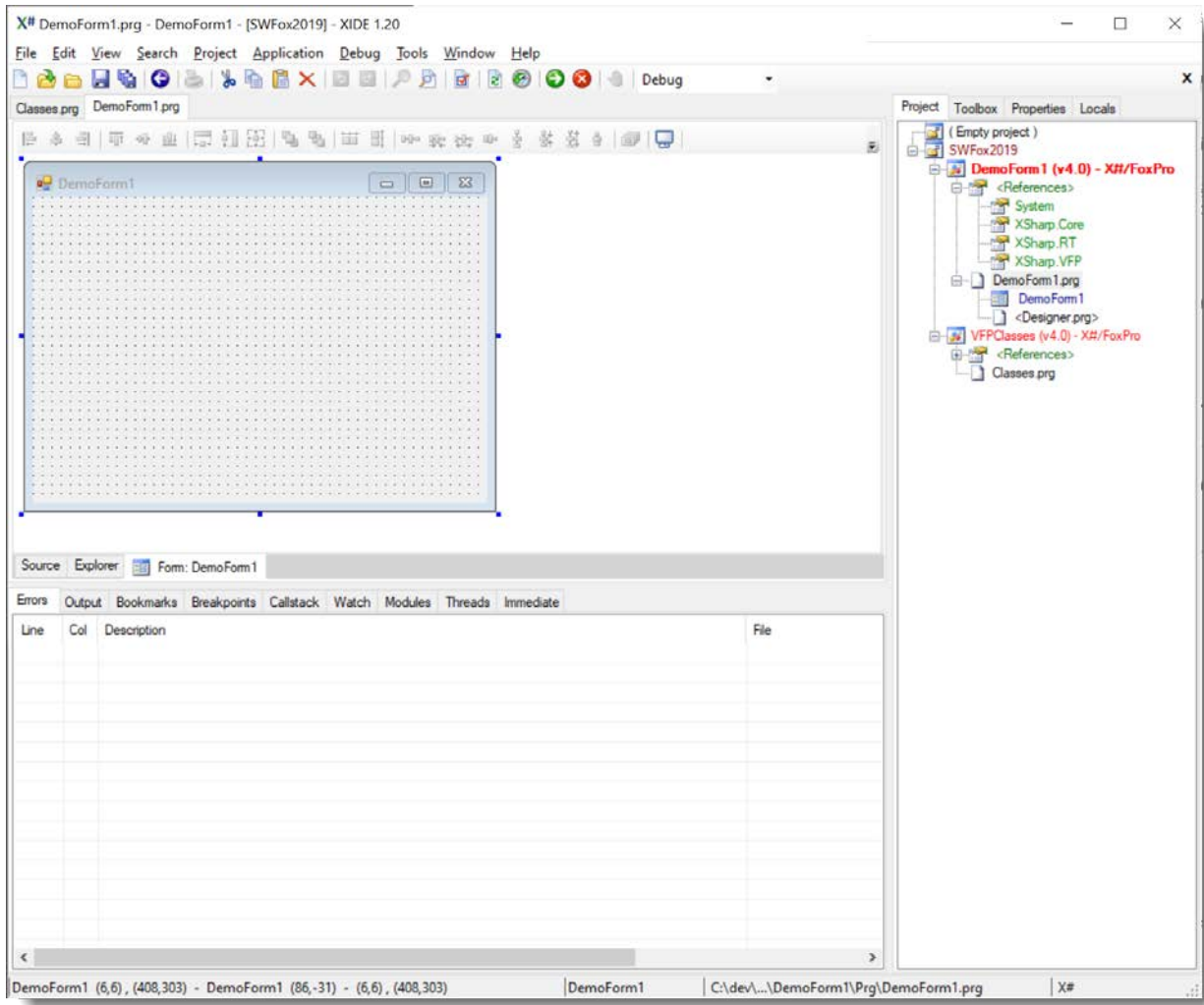


Figure 1: XIDE with Form Designer showing

Project	Toolbox	Properties	Locals
DemoForm1 (System.Windows.Forms.Form)			
General Styles Events ExEvents Design			
Click		<Auto>	
DoubleClick		<Auto>	
GotFocus		<Auto>	
LostFocus		<Auto>	
Enter		<Auto>	
Leave		<Auto>	
Validated		<Auto>	
Validating		<Auto>	
Resize		<Auto>	
SizeChanged		<Auto>	
RegionChanged		<Auto>	
Move		<Auto>	
LocationChanged		<Auto>	
Paint		<Auto>	
Disposed		<Auto>	
PreviewKeyDown		<Auto>	
KeyDown		<Auto>	

Figure 1: Property Sheet

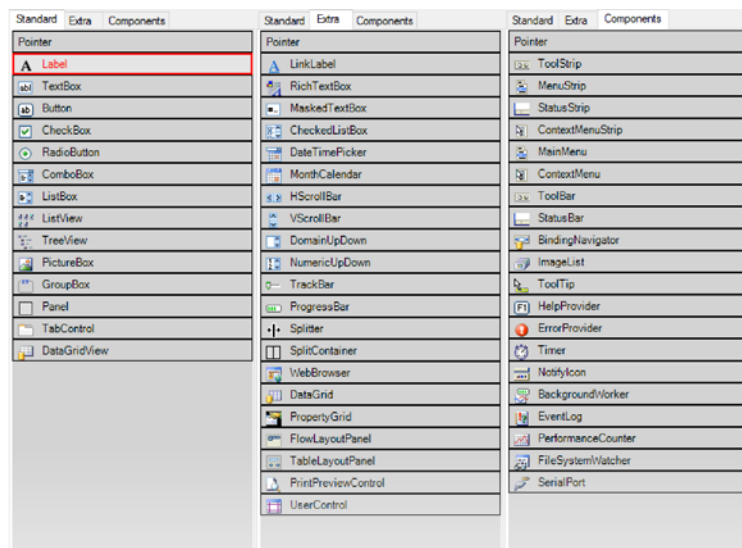


Figure 3: Extensive Form Toolbox

Notice in the Toolbox (**Figure 3**) that X# has built-in access to some controls that you'd have to use ActiveX controls for in Visual FoxPro, including TreeView, RichTextBox, and DateTime pickers. In addition to the built-in controls you can use any .NET capable controls, such as DBi's Studio Controls for .NET and Solutions Schedule for .NET.

At the moment there is no panacea for converting VFP's forms directly into .NET WinForms because of the large differences, though there are some great posts about strategies for doing this manuallyⁱⁱⁱ and some conversion consultants claim to have created tools to do a lot of the manual labor for them. There is no facility at all for reports or labels in XIDE.

Visual Studio

The other IDE for X# is Visual Studio. You can use either the free Community Edition or the full-blown version of Visual Studio, which is not free.

Visual Studio has all the same features as XIDE as well as a lot more such as built-in revision control, database explorer, integrated unit testing, and code and performance analyzers. Visual Studio also has an Object Browser that lets you view the interfaces of those referenced libraries so you can see what properties and methods they have.

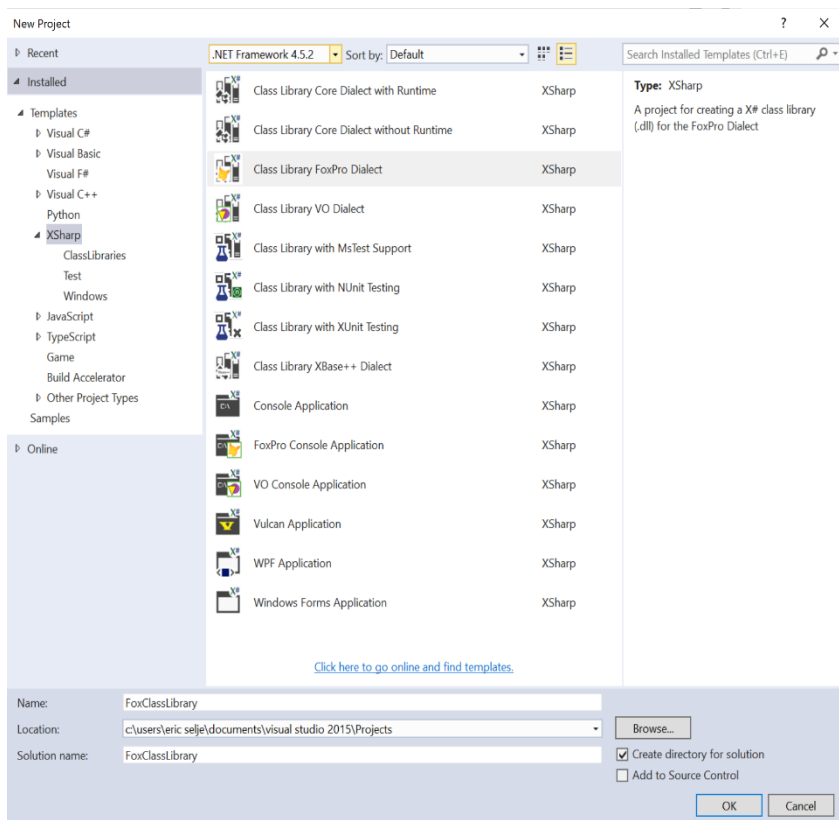


Figure 4: Visual Studio X# Templates

X# ships with some simple templates for creating applications, including some “FoxPro Dialect” templates (**Figure 4**) as well as VO and Vulcan (you know what those are now thanks to our brief history lesson).

Look at X Sharp!

When you choose *Class Library FoxPro Dialect*, the project that gets created defaults to the FoxPro Dialect and the output will be a DLL that you can use in other projects (**Figure 5**). Notice how Visual Studio's nomenclature is Solutions and Projects, rather than Projects and Applications like XIDEs uses. Also notice how this template automatically include a lot of the references you're likely to need but does not include references to User Interface classes that you're not likely to need (System.Console, e.g.).

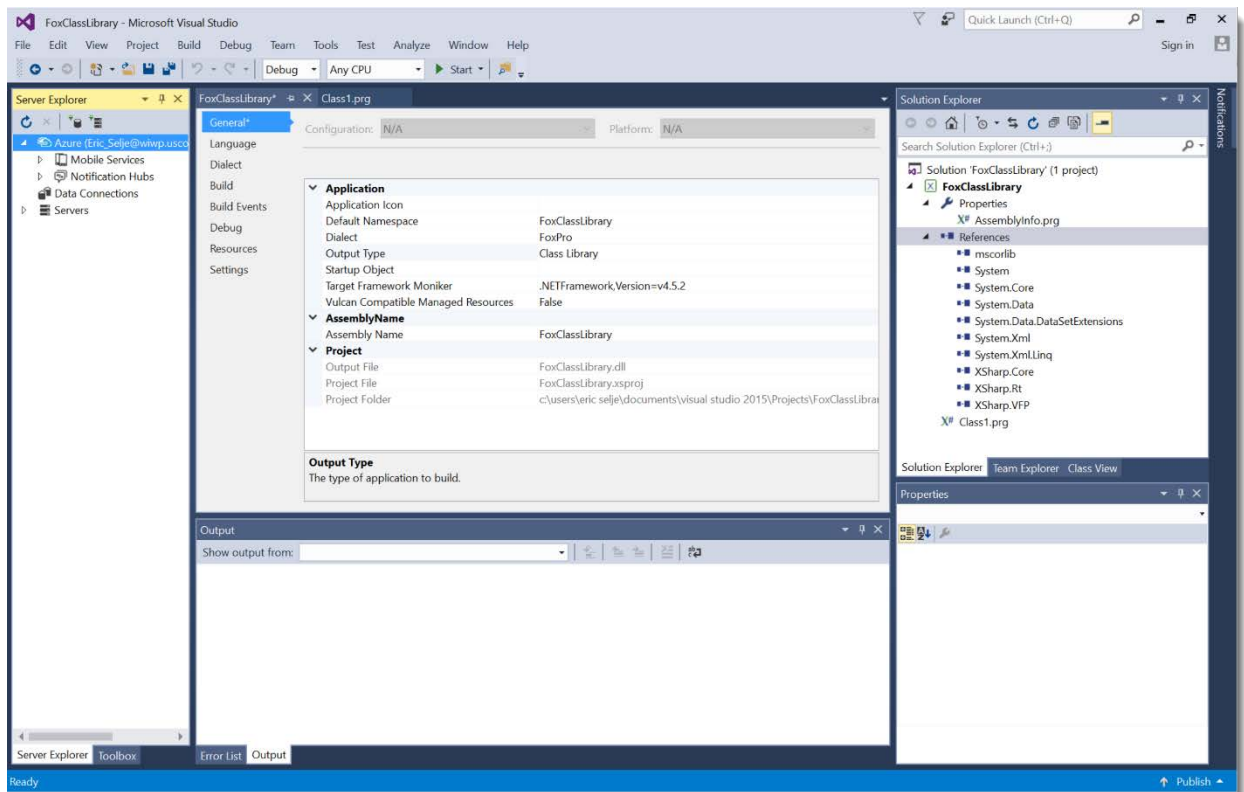


Figure 5: Visual Studio, showing Project Properties and the Server and Solution Explorers

When you choose the *FoxPro Console Application* template, the defaults change to an EXE output but the dialect stays at FoxPro, which means the compiler is tuned to the syntax of FoxPro code.

What if you want to create a Windows App with a FoxPro dialect? There's no template specifically for that but you could choose either WPF or Windows Forms (WebForms) Application from the template list and manually switch the dialect to FoxPro.

What about a Web Application? An ASP.Net based web application can absolutely use the class libraries created in X#, and from any dialect including FoxPro, but cannot use the user interface (forms).

The Language

Let's look at the code that is provided to us by each of those templates and you'll start to get a feel for some of the major differences. Starting with the FoxPro Console Application, which creates this PRG by default

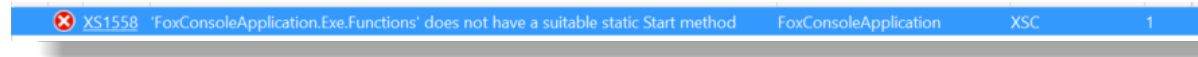
```
USING System
USING System.Collections.Generic
USING System.Linq
USING System.Text

FUNCTION Start() AS VOID STRICT
    ? "Hello World! Today is ", ToDay()
    WAIT
RETURN
```

A couple of things to notice.

First, the USING statements. These are akin to Visual FoxPro's SET LIBRARY TO statements. Include these and you'll have access to all of their functionality. [I like this syntax better so quite a while ago I added a USING() function to my apps that essentially checks to see if a class library is already in scope, and if not, adds it].

Second, the application *must* have a function called Start() somewhere in it. This is its jumping in point. Take it out or rename it and compilation will fail.



Lastly, if you have a sharp eye you may have noticed that this function uses the **ToDay()** function, and you've been around long enough to know that VFP does not have one of those. We specifically chose a FoxPro Console Application, and doublechecked that it used the FoxPro dialect, so where does this ToDay() function come from? It turns out that choosing the FoxPro dialect does not constrain us only to the commands that Visual FoxPro used; We still have the entire X# Core language at our disposal.

It has already been mentioned, but it's important to remember always that X# is a .NET language, which means a few things:

You have access to the entire .NET ecosystem

This is exciting because .NET is almost twenty years old and there are a massive number of third-party tools in addition to what's already in the existing .NET Framework. Need the latest decryption algorithm? A JSON handler? Logging? Exception handling? SSH transfers? Include a reference to the library in your application and it's available to you. Almost anything you can think of that has a VFPx (or some other third party) solution has a .NET equivalent, and a lot more.

Everything is an object

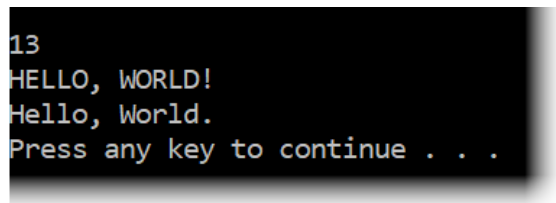
In VFP a string is a string, an integer is an integer, and a date is a date. Those are the base classes. But in .NET everything is an object, and all objects are derived from the base class OBJECT. A string is actually a String object, integers are Integer objects, a date is a Date object, and they're all OBJECT objects. And X# has a lot more datatypes (see Appendix A).

You may wonder how you do a VARTYPE() test if everything is going to return 'O' for object. X# has an IsInstanceOf() / IS function that can tell you.

```
? IsInstanceOf(cString, "STRING") // .T.  
? cString IS STRING // .T.
```

Everything being an object means *a lot* of functions that are necessary in VFP are already built-in methods, or properties, of the objects in X#. ⁵

```
FUNCTION Start() AS VOID  
  LOCAL cString AS STRING  
  cString := "Hello, World!"  
  ? cString.Length // Length is a property of String  
  ? cString.ToUpper() // No need for a separate UPPER() function  
  ? cString.Replace('!', '.') // Hats off to you if you can always remember the name  
of the VFP function that does this  
  Wait  
RETURN
```



```
13  
HELLO, WORLD!  
Hello, World.  
Press any key to continue . . .
```

You can do similar things with date fields. Notice how every object also has a ToString() method that allows you to tweak the format of the output. ⁶

```
VAR dToday := datetime.now // Using the now property of the static DateTime type  
? dToday.date // 10/06/2019 12:00:00 AM  
? dToday.DayOfWeek // Sunday  
? dToday.ToString("d") // 10/06/2019  
? dToday.ToString("D") // Sunday, October 6, 2019
```

And with numbers

```
VAR iValue := 1234.56  
? iValue.ToString("C") // $1,234.56
```

⁵ For a complete list of all the methods and ToString() parameters, refer to the .NET documentation on Microsoft Docs for the type of object you're dealing with. If it works in .NET it should work in X# as well.

⁶ VFPx's FoxTypes class library adds much of this functionality to native VFP.

Everything is strongly typed

For better and worse, VFP is a loosely-typed language. This allowed us to pay less attention to declaring types and allowed us to get away with a lot of things. The core X# language is strongly typed. The advantage to this is the Intellisense is more helpful and the compiler will catch errors before runtime. This comes at the expense of flexibility, but it's a fair price to pay. There are workarounds to make variables more dynamic.

Namespaces

Let's take a look at what those templates created for us when we chose a *FoxPro Class Library*. Here we get a PRG that looks like this:

```
BEGIN NAMESPACE FoxClassLibrary
    CLASS Class1
        CONSTRUCTOR() STRICT
            RETURN
    END CLASS
END NAMESPACE
```

You'll notice that we now get "Namespaces". This allows us to have classes with the same name without having conflicts. The class that's created from this code will be called `FoxClassLibrary.Class1`, and we can have a class named `Class1` in other namespaces without issues.

You'll probably also notice that this isn't the way we `DEFINE CLASS` in Visual FoxPro. There's no `INIT()` method and no `DESTROY()` method either. As of this writing the developers are actively working on implementing VFP's `DEFINE CLASS`, but because of some inherent differences between the way VFP has always done things and the way .NET does things they are deliberating the proper defaults (e.g. Is our `INIT()` the same as the Constructor or a separate method?^{ivv}).

Overloading and Getters/Setters

In X# (and all of .NET), you can *overload* the `Constructor()` method to accept different number and types of parameters. In the following example of Animal classes, we could have constructors that instantiate animals with 8 legs by sending 8 as a parameter in the constructor and assigning that to `iFeet`.

Working Application and Class Libraries Together

Here is an example expands the templates to show you how classes are defined in X# and how to use those classes in your application. Even though the syntax mixes X# and .NET, I think any Visual FoxPro developer will be able to easily comprehend what's going on here.

```
BEGIN NAMESPACE FoxClassLibrary
CLASS Animal
    PUBLIC iFeet AS Int

    CONSTRUCTOR()
        iFeet := 4
        RETURN

    public Function Sound() As String
        Return ""
    END Function
END CLASS

CLASS Primate INHERIT Animal
CONSTRUCTOR()
    iFeet := 2
    RETURN

    public FUNCTION Sound() As String
        Return "Screech!"
    END FUNCTION

END CLASS

CLASS Human INHERIT Primate

    public FUNCTION Sound() As String
        Return "Hello, World!"
    END FUNCTION

END CLASS

END NAMESPACE
```

Figure 3: Class Library Showing Inheritance

```
// Animals.prg
USING System

FUNCTION Start() AS VOID

    VAR oAnimal := FoxClassLibrary.Animal{}
    VAR oPrimate := FoxClassLibrary.Primate{}
    VAR oHuman := FoxClassLibrary.Human{}

    System.Console.WriteLine("Animals generally say '{0}' and have {1} feet.", oAnimal.Sound(),
oAnimal.iFeet)
    System.Console.WriteLine("Primates say '{0}' and have {1} feet.", oPrimate.Sound(), oPrimate.iFeet)
    System.Console.WriteLine("Humans say '{0}' and have {1} feet.", oHuman.Sound(), oHuman.iFeet)

    wait

    RETURN
```

Figure 2: Console Application that uses Classes

New Language Features

Every derivative of xBase has added features, and X# is no exception. There are many new commands that Visual FoxPro doesn't have because they came from the Clipper side of things. Here are some other new features that I'd like to highlight:

Enhanced Strings

In Visual FoxPro you'd often see syntax like this, which maxes out the string delimiters and concatenates different parts to make it all proper.

```
cString = [Bob wrote, "It's time to went ] + "[sic]" + [to a new version".] + chr(13)
```

In X# you can use *enhanced* strings to simplify the syntax:

```
cString = e"Bob wrote, \"It's time to went [sic] to a new version\".\n"
```

Interpolated Strings

Interpolated strings allow you to easily embed variables into a string.

```
cHeader = "This is a header line"  
cHTML = ei"<TABLE><TH>{cHeader}</TH></TABLE>"
```

and you can combine the two:

```
cHeader = "This is a header line"  
cClass = "myTableClass"  
cHTML = ei"<TABLE class=\"{myTableClass}\"><TH>{cHeader}</TH></TABLE>"
```

LINQ

We love our embedded SQL commands (SELECT, INSERT, DELETE) in Visual FoxPro, but because of the order of the syntax we don't get Intellisense. Microsoft derived LINQ to overcome that issue, so instead of

```
SELECT field FROM table alias WHERE otherfield = VALUE ORDER BY field INTO output
```

LINQ turns that around to say

```
Var Output =  
  FROM alias IN table  
  WHERE <condition>  
  ORDERBY <fields>  
  SELECT <fields>
```

We usually think of SELECT as pulling from a table, but LINQ can extract data from *any* enumerable data source: arrays, lists, even literals like strings. And because we throw the *alias* at the beginning, Intellisense can help us out.

Codeblocks

This comes from the Clipper side. Codeblocks are similar to macros but more like anonymous functions. EVAL() is used to call them, with the parameters afterwards.

```
FUNCTION Start() AS VOID
  LOCAL cb AS CODEBLOCK
  cb := {|a,b| a * b}
  ? Eval(cb, 1,2)
  WAIT
RETURN
```

Anonymous Functions (Lambdas)

Lambda functions may seem to be the “new” thing, but they’ve actually been around since 1958 in LISP and their roots go back to math theory much older than that. It’s essentially a way of saying “here’s a function that I don’t really need to name because I’m only using it for a very short and specific time,” like the del function below.

```
DELEGATE Multiply(x AS REAL8) AS REAL8
FUNCTION Start AS VOID
  LOCAL del AS Multiply
  // Lambda with untyped parameters
  del := {e => e * e}
  ? del
  ? del(1)
  ? del(2)
  ? "Lambda with typed parameters"
  del := {e as REAL8 => e * e * e}

  ? del(3)
  ? del(4)
  ? "Anonymous Method Expression"
  del := DELEGATE(e as REAL8) {
    e := e * e * e * e
    return e
  }

  ? del(5)
  Console.ReadLine()
RETURN
```

Working with Data

Visual FoxPro is a fine general purpose language for business apps, but it's really good at manipulating data. That data can be from any ODBC source, but it also has its native DBF table format, CDX index format, and DBC database container format.

X# is also made for manipulating data, but it cannot assume that local data is going to be in the DBF/CDX format because its ancestry came from the DBF/NTX side of things. So they have the concept of Replaceable Database Drivers (RDDs) that allow their classes to work on any of the popular xBase table types *[note that X# often still uses the nomenclature of database when they're referring to individual tables, a habit some VFP can't shake either though many have.]* as well as Advantage Database Server. The RDD that we want for our native local tables is called "DBFVFP," a successor to "DBFCDX" that includes VFP-specific data types. You tell X# that that's the default RDD you want to use by issuing

```
RDDSetDefault("DBFVFP")
```

If you've ever wished that tables were more object-oriented, then you're going to love X# because data manipulation is accomplished through the CoreDb class. While you'd normally issue the

```
USE <tablename>
```

in VFP, that syntax doesn't work (by default) in X#. Instead, X# natively "uses" a table with this syntax:

```
CoreDb.UseArea(True, "DBFVFP", cTable, cAlias, lShared, lReadOnly)
```

A couple of things to note:

1. That's just one of 3 overloaded constructors for CoreDb.UseArea()
2. When you use CoreDb methods, you cannot leave parameters off because X# takes them for null, not False, and null is not allowed.
3. There are quite a few commands and functions that are really just wrappers around the CoreDb function, and will allow you to leave parameters off. For example,

```
DBSkip()
```

is the same as

```
CoreDb.Skip()
```

But

UseArea in an alias for CoreDb.UseArea, and you may see these aliases used interchangeably in sample code. is the same as DBSkip, e.g. There are quite a few native functions in X#, like Regardless of which syntax you use, there is a CoreDB class underneath, and it has a lot of properties and methods that are separate commands in Visual FoxPro. I really appreciate the objectification of tables.

4.

If you're thinking all this CoreDB class is dealbreaker because you don't want to learn new syntax, do not fret. The X# team has included a header file (see **Appendix B**) that you can include in your source code that aliases a lot of our beloved commands to the CoreDb method. If you include that

```
#include "C:\Program Files (x86)\XSharp\Include\dbcmd.xh"
```

You can now go use the USE.

This include file is actually pretty powerful. I discovered that SCAN/ENDSCAN aren't supported in X#, but by adding these two lines to the dbcmd.xh file, they now are:

```
#command SCAN      => CoreDb.GoTop(); DO WHILE NOT CoreDb.Eof()
#command ENDSCAN   => CoreDb.Skip(1); ENDDO
```

Here's what some X# code looks like before including dbcmd.xh

```
LOCAL cDBF = "c:\dev\x#\zips.dbf"
CoreDb.UseArea(True, "DBFVFP", cDBF, "ZipCodes", True, False)
CoreDb.GoTop()
? CoreDb.Alias(CoreDb.GetSelect())
DO WHILE NOT CoreDb.Eof()
    ? zips->city, zips->state, zips->zip_code
    CoreDb.Skip(1)
ENDDO
CoreDb.CloseArea()
```

And this familiar looking code does the same exact thing after I added that (and include my SCAN/ENDSCAN aliases)

```
USE (cDbf) ALIAS ZipCodes
SCAN
    ? city, state, zip_code
ENDSCAN
USE
```

Does the *table->field* bring FoxBase flashbacks for you? It did for me. The X# team is working on the *table.field* syntax, but by adding the

```
FIELD city,state,zip_code // Need this to avoid workarea-> syntax
```

command you can avoid adding an alias altogether. At this point the FoxPro dialect in X# does not prioritize field names over memory variables like Visual FoxPro does.

What's Missing?

Although X# has added a lot of features, there are things that we Visual FoxPro developers love to use that aren't natively in there yet. Here's a list of the big ones:

- *Embedded SQL Commands.* There is LINQ, but that's not quite the same. And you can work with database servers by sending SQL commands much like we do `SQLExec()` now, but there's not yet native support for doing `SELECT` from native VFP tables.
- *Cursors.* X# (well, .NET) has a *datatable* class which is similar, though object-based and powerful but not exactly the same as cursors in Visual FoxPro.
- *Built-in Reports.* There are a lot of 3rd party reporting products for .NET but nothing is included in X#
- *Scatter/Gather*
- *Database Support, including referential integrity*

Compiling

X# is a .NET language (have I mentioned that?). X# gets a leg up on other xBase compilers because it takes advantage of Microsoft's Roslyn compiler, which was open-sourced a few years ago.^{vi}

Both XIDE and Visual Studio development environments have direct hooks into the compiler. I like the immediate and interactive feedback you get from the compilers.

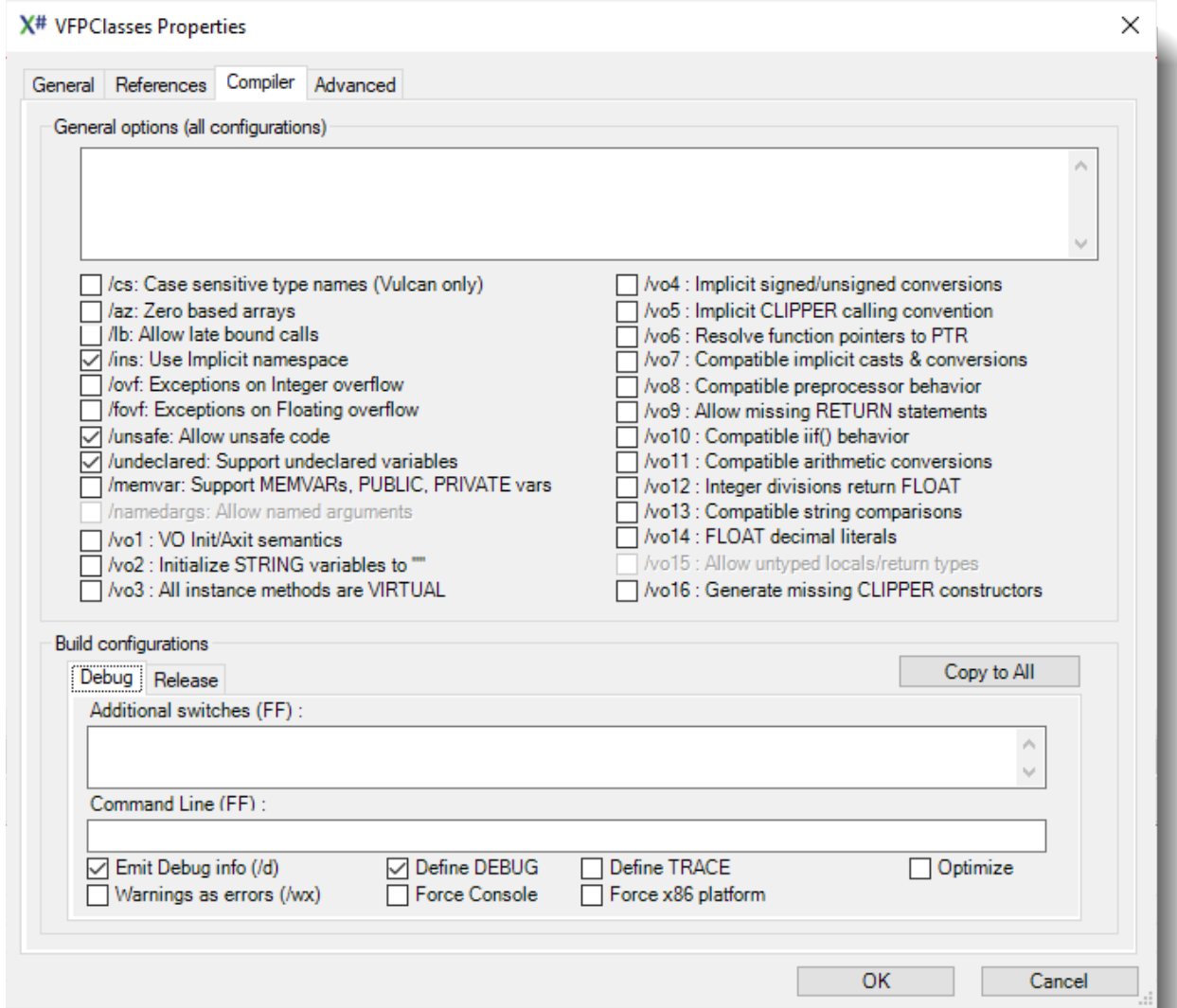


Figure 5: XIDE Compiler Options

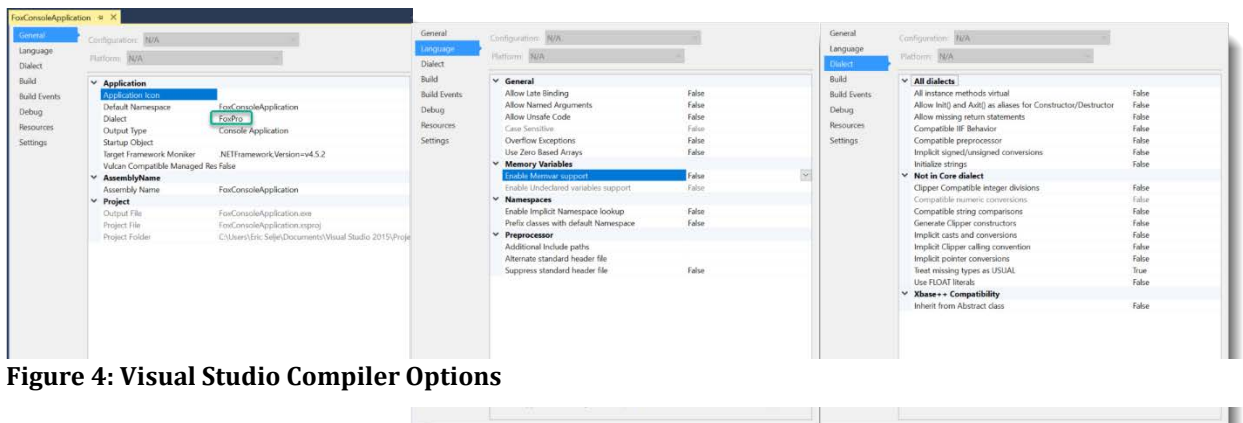


Figure 4: Visual Studio Compiler Options

The resulting EXE's may seem very small

Name	Date modified	Type	Size
DemoBasicApp1.exe	09/22/2019 8:12 PM	Application	4 KB
DemoBasicApp1.sdlb	09/22/2019 8:12 PM

But in order to run the client must have the targeted .NET Framework installed on their PC (which they *probably* do as their distributed by Microsoft as part of Windows Updates or other applications as needed), as well as the runtime for the dialect of X# that was chosen.

Conclusion

I am very impressed with X# as a development tool. I believe the core development team is very, ahem, sharp and it looks to be a viable entity. Because it takes advantage of the .NET Compiler, a future version of X# that uses just the .NET *Core* could compile applications that run on Linux and Mac platforms as well.

The X# developers are actively working on integrating Visual FoxPro functionality. I suspect if we get on board and support them, they will continue to do this, but if there's apathy or indifference they'll shift their limited resources elsewhere. You can support them by participating in the forums at www.xsharp.info, and by joining the "Friends of XSharp". This support program gives you access to all of the X# source code, premier customer support, direct access to the developers, and more frequent product updates.

In true open source fashion you can also fork the source code, enhance it, and submit a Pull Request back to the developers. If your code is accepted it will go into the product. There's also work to be done on the documentation and help file.

This has been a brief overview, and I would have liked to have more time to get into more details. The target is also moving quickly as new updates are being released often.

My conclusion is that although X# is not a trivial port for existing applications, it does provide developers a way to step into a modern development ecosystem while leveraging their existing knowledge. Try rewriting your smaller applications in it to get a feel for what it can do. Migrate your business logic and test it from your existing FoxPro applications using `wwDotNetBridge`.

It's very well suited for new data-driven Windows applications. Once you learn X# it's a small step from knowing C# as well. I've really enjoyed playing with X# and look forward to converting some applications and creating new ones with it. X# has put some joy back into programming for me.

Appendix A: X# Data Types

Type	Category	.Net Name	Size in Bits
BYTE	Unsigned Integer	Byte	8
CHAR	Character	Char	16
DWORD	Unsigned Integer	UInt32	32
DECIMAL	Numeric	Decimal	96
DYNAMIC	Multi purpose	Dynamic	Reference (32 or 64 bits)
INT	Signed Integer	Int32	32
INT64	Signed Integer	Int64	64
LOGIC	Logic	Boolean	8
LONGINT	Signed Integer	Int32	32
OBJECT	Multi purpose	Object	Reference (32 or 64 bits)
PTR	Multi purpose	IntPtr	Reference (32 or 64 bits)
REAL4	Floating Point	Single	32
REAL8	Floating Point	Double	64
SBYTE	Signed Integer	SByte	8
SHORT	Signed Integer	Int16	16
STRING	String	String	Reference (32 or 64 bits)
UINT64	Unsigned Integer	UInt64	64
VOID	Not a type	Void	0
WORD	Unsigned Integer	UInt16	16

xBase Specific Types

Type
<u>ARRAY</u>
<u>CODEBLOCK</u>
<u>DATE</u>
<u>FLOAT</u>
<u>PSZ</u>
<u>SYMBOL</u>
<u>USUAL</u>

User Defined Types

Type
<u>CLASS</u>
<u>DELEGATE</u>
<u>ENUM</u>
<u>Generic Types</u>
<u>INTERFACE</u>
<u>STRUCTURE</u>
<u>UNION</u>
<u>VOSTRUCT</u>

Appendix B: The dbcmd.xh file

Including this header file allows you to alias our beloved FoxPro commands to X#'s syntax, most of which are methods on the CoreDb class.

```
////////////////////////////////////
// DbCmds.xh
//
// XSharp Database commands
//
// Copyright (c) XSharp BV. All Rights Reserved.
// Licensed under the Apache License, Version 2.0.
// See License.txt in the project root for license information.
//
// IMPORTANT: Functions beginning with an underscore
//            character are reserved, version-dependent functions.
//            These functions should not be called directly.
//
// Caution: do not modify this file. It will be overwritten during product updates
//

****
*   DB SETs
*

#command SET EXCLUSIVE <x:ON,OFF,&>      => SetExclusive(<(x)> )
#command SET EXCLUSIVE (<x>)           => SetExclusive( <x> )

#command SET SOFTSEEK <x:ON,OFF,&>      => SetSoftSeek( <(x)> )
#command SET SOFTSEEK (<x>)           => SetSoftSeek( <x> )

#command SET UNIQUE <x:ON,OFF,&>       => SetUnique( <(x)> )
#command SET UNIQUE (<x>)            => SetUnique( <x> )

#command SET DELETED <x:ON,OFF,&>      => SetDeleted( <(x)> )
#command SET DELETED (<x>)           => SetDeleted( <x> )

****
*   DB
*

#command SELECT <whatever>              => dbSelectArea( <(whatever)> )
#command SELECT <f>(<[list,...]>))      => dbSelectArea( <f>(<list>) )

#command USE                            => dbCloseArea()

#command USE <(db)>                      ;
        [VIA <rdd>]                      ;
        [ALIAS <a>]                       ;
        [<new: NEW>]                     ;
        [<ex: EXCLUSIVE>]               ;
```



```

        [<sh: SHARED>] ;
        [<ro: READONLY>] ;
        [INDEX <(index1)> [, <(indexn)>]] ;
    => dbUseArea( ;
        <.new.>, <rdd>, <(db)>, <(a)>, ;
        if(<.sh.> .or. <.ex.>, !<.ex.>, NIL), <.ro.> ;
    ) ;
    [; dbSetIndex( <(index1)> )] ;
    [; dbSetIndex( <(indexn)> )] ;

#command SET INDEX TO [ <(index1)> [, <(indexn)>]] ;
    => dbClearIndex() ;
    [; dbSetIndex( <(index1)> )] ;
    [; dbSetIndex( <(indexn)> )] ;

#command INDEX ON <key> TO <(file)> [<u: UNIQUE>] ;
    => dbCreateIndex( ;
        <(file)>, <"key">, <{key}>, ;
        IIF( <.u.>, TRUE, NIL ) ;
    ) ;

#command REINDEX => dbReindex()
#command SET ORDER TO <n> => dbSetOrder( <n> )
#command SET ORDER TO => dbSetOrder(0)

#command APPEND BLANK => dbAppend()
#command PACK => dbPack()
#command ZAP => dbZap()
#command UNLOCK => dbUnlock()
#command UNLOCK ALL => dbUnlockAll()
#command COMMIT => dbCommitAll()

#command GOTO <n> => dbGoto(<n>)
#command GO <n> => dbGoto(<n>)
#command GOTO TOP => dbGoTop()
#command GO TOP => dbGoTop()
#command GOTO BOTTOM => dbGoBottom()
#command GO BOTTOM => dbGoBottom()

#command SKIP => dbSkip(1)
#command SKIP <n> => dbSkip( <n> )
#command SKIP ALIAS <a> => <a> -> ( dbSkip(1) )
#command SKIP <n> ALIAS <a> => <a> -> ( dbSkip(<n>) )

#command SEEK <xpr> => dbSeek( <xpr> )

```

```

#command FIND <*text*>          => dbSeek( <(text)> )
#command FIND := <xpr>         => ( find := <xpr> )
#command FIND = <xpr>         => ( find := <xpr> )

#command CONTINUE              => dbContinue()

#command LOCATE                ;
    [FOR <FOR>]                 ;
    [WHILE <WHILE>]             ;
    [NEXT <NEXT>]               ;
    [RECORD <rec>]              ;
    [<rest:REST>]               ;
    [ALL]                       ;
                                ;
=> dbLocate( <{FOR}>, <{WHILE}>, <NEXT>, <rec>, <.rest.> )

#command SET RELATION TO      => DbClearRelation()

#command SET RELATION        ;
    [<add:ADDITIVE>]          ;
    [TO <key1> INTO <(alias1)> [, [TO] <keyn> INTO <(aliasn)>]] ;
                                ;
=> if ( !<.add.> )              ;
    ; DbClearRelation()        ;
    ; END                       ;
                                ;
    ; dbSetRelation( <(alias1)>, <{key1}>, <"key1"> ) ;
    [; dbSetRelation( <(aliasn)>, <{keyn}>, <"keyn"> )] ;

#command SET FILTER TO      => dbClearFilter(NIL)
#command SET FILTER TO <xpr> => dbSetFilter( <{xpr}>, <"xpr"> )

#command SET FILTER TO <x:&> ;
=> IF ( Empty(<(x)>) )         ;
    ; dbClearFilter()         ;
    ; ELSE                     ;
    ; dbSetFilter( <{x}>, <(x)> ) ;
    ; END                       ;

#command REPLACE [ <f1> WITH <x1> [, <fn> WITH <xn>] ] ;
    [FOR <FOR>]                 ;
    [WHILE <WHILE>]             ;
    [NEXT <NEXT>]               ;
    [RECORD <rec>]              ;
    [<rest:REST>]               ;
    [ALL]                       ;
                                ;
=> DBEval(                     ;
    {|| _FIELD-><f1> := <x1> [, _FIELD-><fn> := <xn>]], ;
    <{FOR}>, <{WHILE}>, <NEXT>, <rec>, <.rest.> ;

```

```

        )

#command REPLACE <f1> WITH <v1> [, <fN> WITH <vN> ] ;
=> _FIELD-><f1> := <v1> [; _FIELD-><fN> := <vN>]

#command DELETE ;
[FOR <FOR>] ;
[WHILE <WHILE>] ;
[NEXT <NEXT>] ;
[RECORD <rec>] ;
[<rest:REST>] ;
[ALL] ;
=> DBEval( ;
    {|| dbDelete()}, ;
    <{FOR}>, <{WHILE}>, <NEXT>, <rec>, <.rest.> ;
) ;

#command RECALL ;
[FOR <FOR>] ;
[WHILE <WHILE>] ;
[NEXT <NEXT>] ;
[RECORD <rec>] ;
[<rest:REST>] ;
[ALL] ;
=> DBEval( ;
    {|| dbRecall()}, ;
    <{FOR}>, <{WHILE}>, <NEXT>, <rec>, <.rest.> ;
) ;

#command DELETE => dbDelete()
#command RECALL => dbRecall()

#command CREATE <(file1)> [FROM <(file2)>] ;
=> _DbCreate( <(file1)>, <(file2)> )

#command COPY [STRUCTURE] [EXTENDED] [TO <(file)>] ;
=> dbCopyXStruct( <(file)> )

#command COPY [STRUCTURE] [TO <(file)>] [FIELDS <fields,...>] ;
=> dbCopyStruct( <(file)>, { <(fields)> } )

#command COPY [TO <(file)>] [DELIMITED [WITH <*delim*>]] ;
[FIELDS <fields,...>] ;
[FOR <FOR>] ;

```

```

    [WHILE <WHILE>] ;
    [NEXT <NEXT>] ;
    [RECORD <rec>] ;
    [<rest:REST>] ;
    [ALL] ;
    ;
=> dbCopyDelim( ;
    <(file)>, <(delim)>, { <(fields)> }, ;
    <{FOR}>, <{WHILE}>, <NEXT>, <rec>, <.rest.> ;
) ;

#command COPY [TO <(file)>] [SDF] ;
[FIELDS <fields,...>] ;
[FOR <FOR>] ;
[WHILE <WHILE>] ;
[NEXT <NEXT>] ;
[RECORD <rec>] ;
[<rest:REST>] ;
[ALL] ;
=> dbCopySDF( ;
    <(file)>, { <(fields)> }, ;
    <{FOR}>, <{WHILE}>, <NEXT>, <rec>, <.rest.> ;
) ;

#command COPY [TO <(file)>] ;
[FIELDS <fields,...>] ;
[FOR <FOR>] ;
[WHILE <WHILE>] ;
[NEXT <NEXT>] ;
[RECORD <rec>] ;
[<rest:REST>] ;
[ALL] ;
=> dbCopy( ;
    <(file)>, { <(fields)> }, ;
    <{FOR}>, <{WHILE}>, <NEXT>, <rec>, <.rest.> ;
) ;

#command APPEND [FROM <(file)>] [DELIMITED [WITH <*delim*>]] ;
[FIELDS <fields,...>] ;
[FOR <FOR>] ;
[WHILE <WHILE>] ;
[NEXT <NEXT>] ;
[RECORD <rec>] ;
[<rest:REST>] ;
[ALL] ;
=> dbAppDelim( ;
    <(file)>, <(delim)>, { <(fields)> }, ;
    <{FOR}>, <{WHILE}>, <NEXT>, <rec>, <.rest.> ;
) ;

```

```
#command APPEND [FROM <(file)>] [SDF] ;
    [FIELDS <fields,...>] ;
    [FOR <FOR>] ;
    [WHILE <WHILE>] ;
    [NEXT <NEXT>] ;
    [RECORD <rec>] ;
    [<rest:REST>] ;
    [ALL] ;
    ;
=> dbAppSDF( ;
    <(file)>, { <(fields)> }, ;
    <{FOR}>, <{WHILE}>, <NEXT>, <rec>, <.rest.> ;
    ) ;
```

```
#command APPEND [FROM <(file)>] ;
    [FIELDS <fields,...>] ;
    [FOR <FOR>] ;
    [WHILE <WHILE>] ;
    [NEXT <NEXT>] ;
    [RECORD <rec>] ;
    [<rest:REST>] ;
    [ALL] ;
    ;
=> dbApp( ;
    <(file)>, { <(fields)> }, ;
    <{FOR}>, <{WHILE}>, <NEXT>, <rec>, <.rest.> ;
    ) ;
```

```
#command SORT [TO <(file)>] [ON <fields,...>] ;
    [FOR <FOR>] ;
    [WHILE <WHILE>] ;
    [NEXT <NEXT>] ;
    [RECORD <rec>] ;
    [<rest:REST>] ;
    [ALL] ;
    ;
=> dbSort( ;
    <(file)>, { <(fields)> }, ;
    <{FOR}>, <{WHILE}>, <NEXT>, <rec>, <.rest.> ;
    ) ;
```

```
#command TOTAL [TO <(file)>] [ON <key>] ;
    [FIELDS <fields,...>] ;
    [FOR <FOR>] ;
    [WHILE <WHILE>] ;
    [NEXT <NEXT>] ;
    [RECORD <rec>] ;
    [<rest:REST>] ;
    [ALL] ;
    ;
```

```

=> dbTotal(
    <(file)>, <{key}>, { <(fields)> },
    <{FOR}>, <{WHILE}>, <NEXT>, <rec>, <.rest.>
)
;
;
;
;

#command UPDATE [FROM <(alias)>] [ON <key>]
[REPLACE <f1> WITH <x1> [, <fn> WITH <xn>]]
[<rand:RANDOM>]
;
;
;
=> dbUpdate(
    <(alias)>, <{key}>, <.rand.>,
    {|| _FIELD-><f1> := <x1> [, _FIELD-><fn> := <xn>]}
)
;
;
;
;

#command JOIN [WITH <(alias)>] [TO <file>]
[FIELDS <fields,...>]
[FOR <FOR>]
;
;
;
=> dbJoin( <(alias)>, <(file)>, { <(fields)> }, <{FOR}> )
;
;
;
;

#command COUNT [TO <var>]
[FOR <for>]
[WHILE <while>]
[NEXT <next>]
[RECORD <rec>]
[<rest:REST>]
[ALL]
;
;
;
;
=> <var> := 0
; DBEval(
    {|| <var> := <var> + 1},
    <{for}>, <{while}>, <next>, <rec>, <.rest.>
)
;
;
;
;

#command SUM [ <x1> [, <xn>] TO <v1> [, <vn>] ]
[FOR <for>]
[WHILE <while>]
[NEXT <next>]
[RECORD <rec>]
[<rest:REST>]
[ALL]
;
;
;
;
=> <v1> := [ <vn> := ] 0
; DBEval(
    {|| <v1> := <v1> + <x1> [, <vn> := <vn> + <xn> ]},
    <{for}>, <{while}>, <next>, <rec>, <.rest.>
)
;
;
;
;

#command AVERAGE [ <x1> [, <xn>] TO <v1> [, <vn>] ]
;
;
;
;

```

```
[FOR <for>] ;
[WHILE <while>] ;
[NEXT <next>] ;
[RECORD <rec>] ;
[<rest:REST>] ;
[ALL] ;
;
=> M->__Avg := <v1> := [ <vn> := ] 0 ;
;
; DBEval( ;
  {|| M->__Avg := M->__Avg + 1, ;
  <v1> := <v1> + <x1> [, <vn> := <vn> + <xn>] }, ;
  <{for}>, <{while}>, <next>, <rec>, <.rest.> ;
  ) ;
; <v1> := <v1> / M->__Avg [; <vn> := <vn> / M->__Avg ] ;

// NOTE: CLOSE <alias> must precede the others
#command CLOSE <alias> => <alias>->( dbCloseArea() )

#command CLOSE => dbCloseArea()
#command CLOSE DATABASES => dbCloseAll()
#command CLOSE INDEXES => dbClearIndex()

** EOF
```

Bibliography

ⁱ https://www.dbase.com/Knowledgebase/dbulletin/bu03_b.htm

ⁱⁱ <https://www.youtube.com/watch?v=rS00xWnqvwI>

ⁱⁱⁱ <http://www.pinter.com/ShowArticle.aspx?ArtNum=400>

^{iv} <https://www.xsharp.info/forum/public-vfp/1446-foxpro-syntax-properties-and-fields#10647>

^v <https://www.xsharp.info/forum/public-vfp/1446-foxpro-syntax-properties-and-fields#10638>

^{vi} <https://www.xsharp.info/articles/blog/have-we-lost-our-mind>