X# workshop

Robert van der Hulst
Fabrice Foray
Phoenix, October 2019

Agenda

- A bit of background info about X#
- An overview of the X# runtime
- What do you want ?

Installing X#

- Run the installer
- What do you get ?
  - Look at the folder
  - Overview of the contents
- What is the compiler ?
- Where is the runtime ?
  - Assemblies folder
  - Redist folder
  - The GAC
- What do you deploy ?

Getting started

- Create an app in the FoxPro dialect
- This enables certain statements and commands in the compiler that are FoxPro specific, such as TEXT .. ENDTEXT and enables certain keywords, such as THIS, FOR EACH (in stead of FOREACH), ENDFOR, etc
- This enables the support for the FoxPro compatible “DEFINE CLASS” syntax

- Adds references to:
  - XSharp.Core.DLL
  - Xsharp.RT.DLL
  - Xsharp.VFP.DLL

Commands vs Functions, Preprocessor

- The biggest “visual” difference between other dialects and VFP is that VFP is much more command based
- The X# compiler contains a preprocessor that you can use to create commands and map these to function calls
- For example
  - #command SKIP => dbSkip(1)
  - #command SKIP <n> => dbSkip(<n>)
- Quick Look at dbcmd.xh and /ppo to the compiler options
- Look at resulting ppo file

- The functions that the commands map to are part of the X# runtime
- For DBF access X# uses Replacable Database Drivers (RDDs) to switch between various formats.

Look at the BIN folder

- It has the EXE
- It has the referenced DLLs
- It does not have the RDD or Macro compiler. When running in VS this is found in the GAC. When you deploy you need to add these.

Have a look at the result

- Open the compiled EXE with ILSpy
- Have a peek

Runtime DLLs

Component	Dialect	Depends on	Contains
Xsharp.Core	X#-Core		Functions, support classes, interfaces “native”
XSharp.RT	X#-Vulcan	Core	Common Xbase Types, Functions
Xsharp.VFP	X#-FoxPro	Core+RT	VFP specific functions and types
Xsharp.MacroCompiler*	C#	Core	Macro compiler
Xsharp.RDD*	X#-Core	Core	All the RDDs
Xsharp.VO	X#-VO	Core+RT	VO specific functions and types
XSharp.XPP	X#-XPP	Core+RT	Xbase++ specific functions & types
VO.. DLLs	X#-VO	Core+RT+VO	VO SDK Classes
* = no explicit reference needed			

XSharp.Core - 1

- Written in X# Core dialect
- Uses only standard .Net data types
- Has all the functions and types that do not require XBase types
- Includes (managed) Low level file I/O (FOpen, FRead, FWrite)
- Declares common interfaces, such as IDate, IFloat, ICodeBlock, IMacroCompiler, used by other X# components
- Has the Runtime ‘State’ and Workarea(=Cursor) information
- Includes typed RDD functions. NO “untyped” support.
- Only “safe” types and code.

Xsharp.Core – 2

- File Handles are not real Win32 file handles

- The Runtime State is prepared for multi threading, each thread has its own workareas/cursors, or in other words each thread has its own datasession

Xsharp.RT - 1

- Written in X# VO dialect
- Declares the XBase types
  - USUAL, DATE, FLOAT, SYMBOL
  - ARRAY, CODEBLOCK, PSZ
  - And more
- Has the functions that use these types (Str(), Ntrim etc)
- Adds functions to the RDD system with optional parameters
- CODEBLOCK = X# specific Lambda expression

Xsharp.RT - 2

- Has the Console API (? , ??)
- Has the support for MEMVARs (PUBLIC, PRIVATE)
- Has the support code for using the Macro Compiler (MCompile, MExec etc)
- Some functions are in XSharp.Core AND in XSharp.RT with different parameter types. The compiler will choose the best overload

Xsharp.VFP

- Written in X# FoxPro dialect
- Support functions for \, \\ and TEXT .. ENDTEXT
- System Variables (_TEXT, _PRETEXT etc)
- Helper types (enums)
- Base classes (Custom, Collection etc)
- FoxPro specific functions
- Sets the default “RDD” to DBFVFP
- Will contain DBC support etc.

Xsharp.MacroCompiler

- Visual FoxPro also has a macro compiler but it is not mentioned explicitly
- Compiles STRINGS to an object with executable code that you can pass around. Sort of like pcode or a C# Lambda Expression
- Depends on Xsharp.Core.DLL and Xsharp.RT.DLL
- Implements the macro compiler interface
- Returns objects of type _CodeBlock
- The codeblocks have USUAL parameters and return a USUAL return value
- Written in C#

Xsharp.RDD.DLL

- Only Depends on Xsharp.Core.DLL
- Written in X#, Core dialect.
- Include support for
  - Advantage
  - DBF DBT NTX, DBF FPT CDX
  - SDF and DELIM

Windows Forms or WPF

- For years Ms (and so have we) told that Windows Forms is dead and that you had to think about moving your apps to WPF
- The market (you) has decided otherwise
- So .Net Core 3.0 has come with support for Windows Forma and WPF

Conversion tools

- We plan to create a tool that takes a VFP project and converts it to a complete X# solution

- Show VO Exporter at work

What do we plan to implement

- All VFP Functions
- Cursor for SQL statements. You will be able to manipulate the cursor just like in FoxPro (create index, set filter, scan endscan etc)
- Wrap DBF based cursors in a DataTable, for .Net databinding. With “Batch” mode or “Direct mode”.
- DataSession Object = workareas, so you can give a form or report its own session
- “Embedded SQL” ? Map to VSP OleDb driver ?
- “Rushmore” ?
- Forms
  - Use native Windows Forms forms ?
  - Or write VFP compatible wrapper around these?
- Reports

Questions

- Some VFP features are very difficult to implement in a managed language, such as :

```
PROCEDURE Test()
Alines(aResult, “some text”+Chr(13)+”Line2”
? m.aResult[1]
? m.aResult[2]
```

In other languages this would generate an error because aResult is not declared yet. VFP will create a variable on the fly and will pass it to Alines() and then aLines() will change it into an array. This violates some of the normal rules of “protection”. The variable is not initialized (so should most likely be false) and is not passed by reference. So Alines() should not be able to change it from uninitialized to an array.

In our implementation Alines() woud be a “normal” function like any other and should not be able to do this.

Questions ?