Moving wLib2 from VO to X#

A sample for a migration from VO to X#

Architecture of my VO programs

First, let me explain the structure of my VO applications:

- The base library is the wLib2, that is about 20 years old and is enhanced when needed, at least once a month. It contains the base classes for servers, controls and windows and a lot of other functions and classes. This library contains only handwritten code and is shared between different projects through external modules.
- Then there is the BasicMapi lib, that contains basic mail functionality and DynMenu, the basic classes for dynamic menus (I don't use the menu editor because most menus are to build depending on the current user
- The next library is wLibDlg, that uses the first three libraries and contains the most used dialogs, like LookupDialog, ChoiceDialog and about 20 others.
- Next in the list is wStdBrowser, that uses bBrowser 4, defines my bBrowser base class and the standard databrowser window from which nearly all databrowser windows are inherited, and the relative filter functionality.
- And then, as last in the list, I have wErrorsys, my base error handler.

There are much more libraries, but they are loaded either dynamically like the Zip (using Compression Plus) or the report classes (using VPE), or used only by a few applications like the Vo2Ado or the MySql lib.

As you can immagine, first candidate in the migration process was the wLib2 library.

Since my libraries and my applications are living, I need the possibility to repeat the migration often, until all of my VO applications (about 50) are migrated to X#.

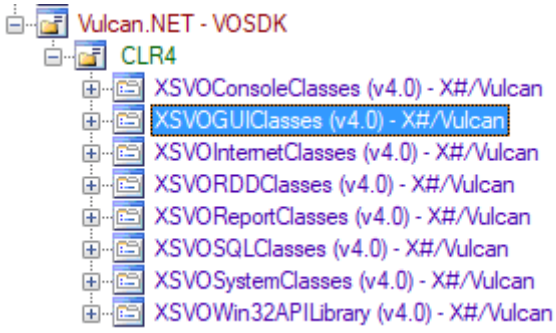
And the basic rule is to not change anything in the X# version of the library that is not changed also in the VO version.

And as last note: I have licenses to Vulcan 4 (as I'm currently subscribed also to VPS) and to the bBrowser4.NET.

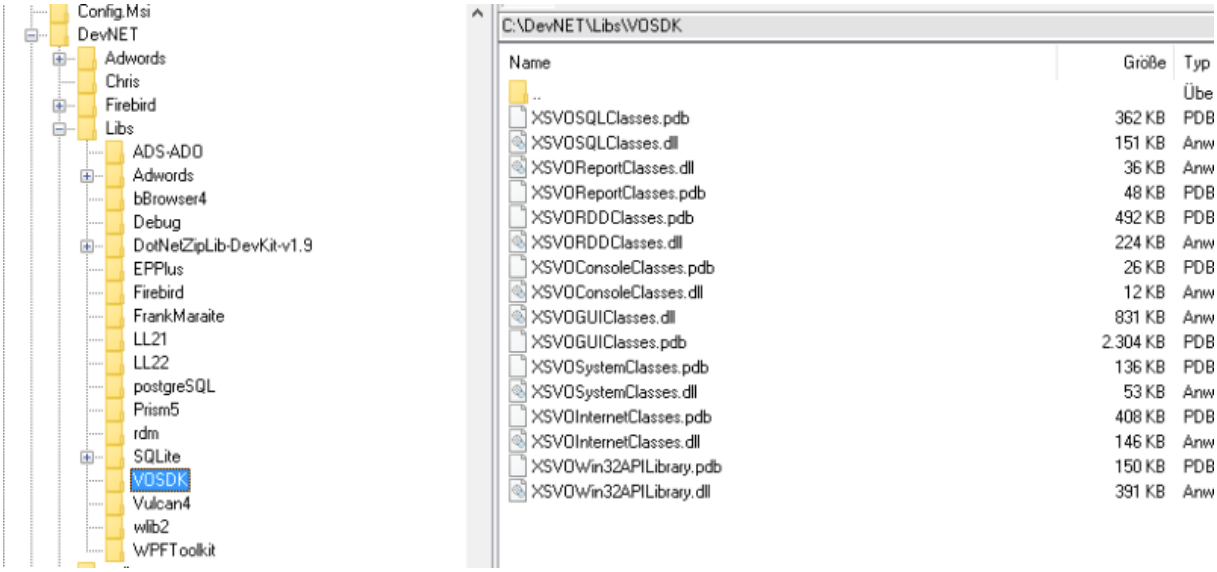
Preparation and first migration of wLib2

Before starting the migration of my most important library, I have imported the source code of the Vulcan version of the VO class libraries into my XIDE projects.

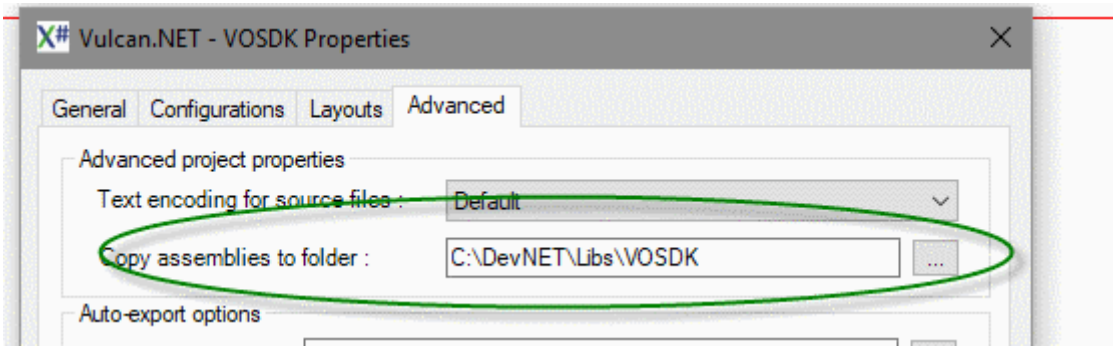
Then I have changed the names of the libraries from VulcanVO* to XSVO*, changed the compiler from Vulcan to X#/Vulcan dialect and compiled them.



Another note: on my C drive, I have a folder called DevNET\Libs, where I put all compiled versions of my libraries:



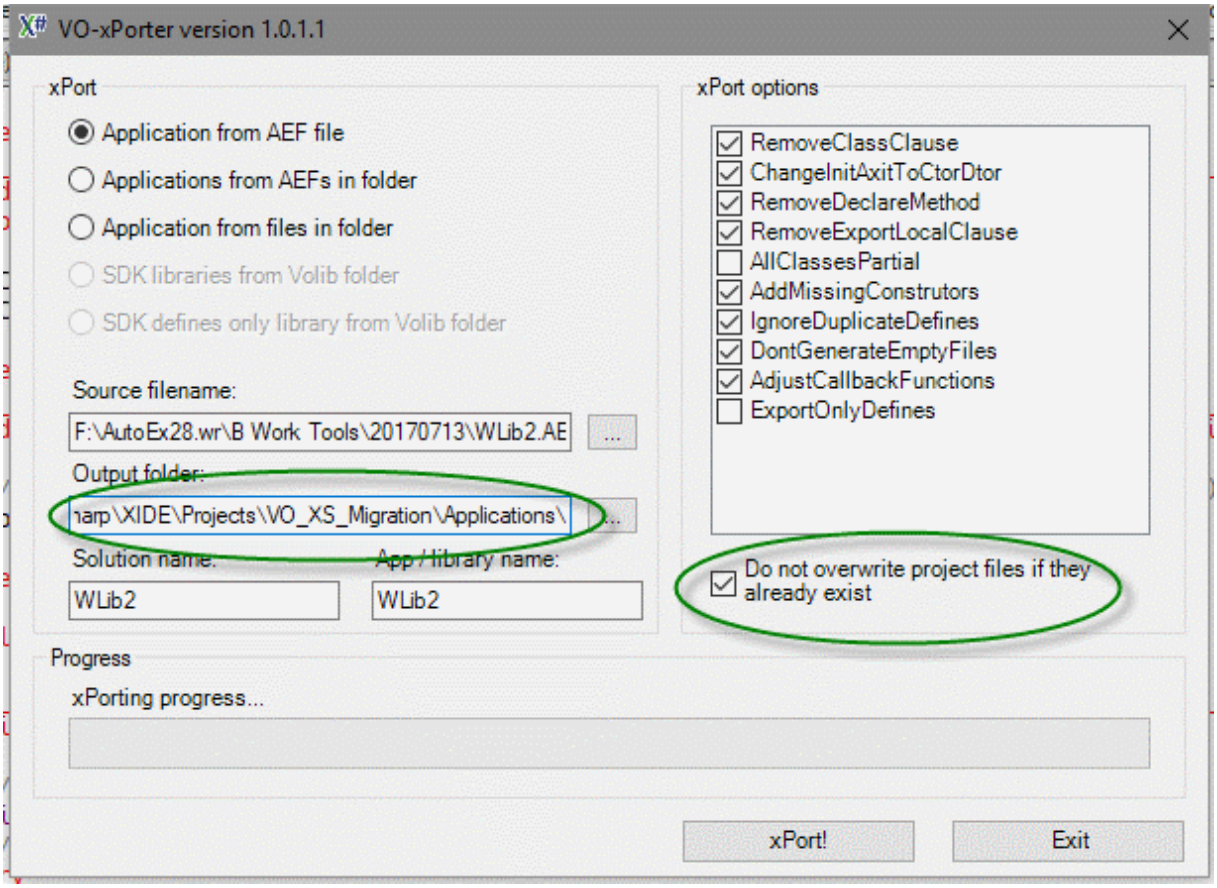
Upon my request, Chris has added a special function in XIDE to copy the binaries of a project to a directory, and I use this funtionality to maintain my c:\devnet\libs directory content up to date:



When using then the libraries in an application references, I select them from the relative c:\devNET\libs folder.

Back to the migration process itself: I created a new XIDE project, called VO to X# migration, using the folder C:\XSharp\XIDE\Projects\VO_XS_Migration\.

Then I fired up the VO-xPorter, selected the most recent version of my wLib2 AEF and used the following settings:

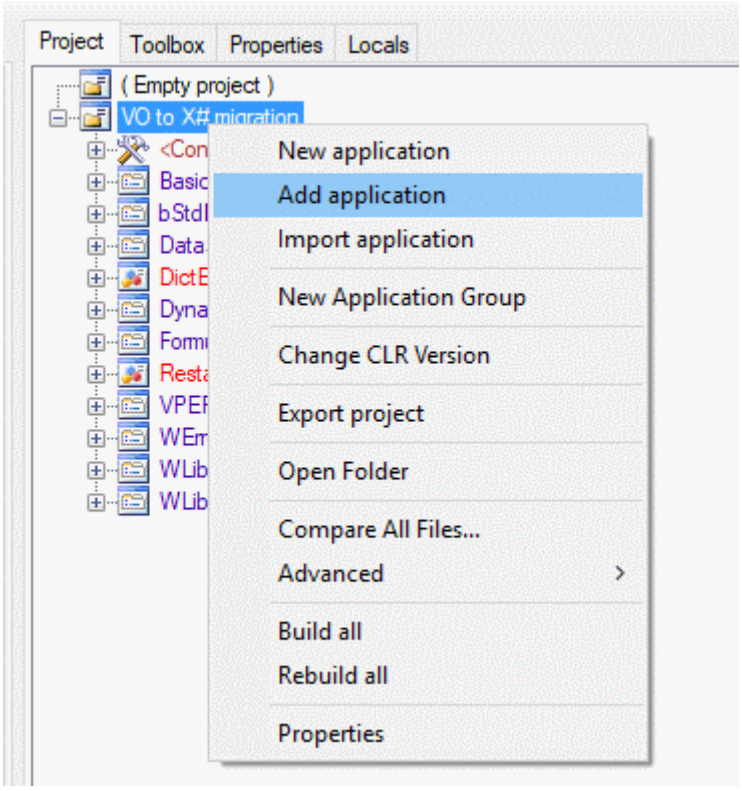


Two settings are very important:

- the output folder points to the Applications directory of the new project
- and the "do not overwrite project files" is checked

These settings are important so I can repeat the migration process how often as I like without the need to reconfigure the projects settings after each run, as only the prg files are written.

After xPorting the AEF, I added the application through the Project - Add application context menu:



After this operation, my library was ready to be compiled the first time – and I was really surprised about the errors.

wLib2 – Compile errors and the relative fixes

This will be the longest and IMHO most important part of this document, and every error will have ist own page to be easier to read and and easier to find.

The first compile gave only very few errors:

Line	Col	Description	File	Application	Path
		Compilation failed (2 errors, 8 warnings)			
716	1	warning XS1030: #warning: 'Callback function modified to use a DELEGATE by xPorter. Please review.'	DictDBServer.prg	WLib2	C:\XSharp\XID...
757	1	warning XS1030: #warning: 'Callback function modified to use a DELEGATE by xPorter. Please review.'	DictDBServer.prg	WLib2	C:\XSharp\XID...
568	39	error XS9002: Parser: unexpected input 'pascal' expecting {WHERE, EOS}	DictDBServer.prg	WLib2	C:\XSharp\XID...
2577	1	warning XS1030: #warning: 'Callback function modified to use a DELEGATE by xPorter. Please review.'	Miscellaneous.prg	WLib2	C:\XSharp\XID...
2489	77	error XS9002: Parser: unexpected input 'callback' expecting {WHERE, EOS}	Miscellaneous.prg	WLib2	C:\XSharp\XID...
21	1	warning XS1030: #warning: 'Callback function modified to use a DELEGATE by xPorter. Please review.'	Globals.prg	WLib2	C:\XSharp\XID...
137	1	warning XS1030: #warning: 'Callback function modified to use a DELEGATE by xPorter. Please review.'	SpawnProcess.prg	WLib2	C:\XSharp\XID...
31	1	warning XS1030: #warning: 'Callback function modified to use a DELEGATE by xPorter. Please review.'	Start.prg	WLib2	C:\XSharp\XID...
992	1	warning XS1030: #warning: 'Callback function modified to use a DELEGATE by xPorter. Please review.'	Printing.prg	WLib2	C:\XSharp\XID...
105	1	warning XS1030: #warning: 'The following method did not include a CLASS declaration'	VOInt27.prg	WLib2	C:\XSharp\XID...

And both of them were caused by the VO-xPorter (Chris Pyrgas is informed about these and they will be fixed in a later version):

```
delegate DictClose_delegate( ) as void pascal
```

changed to

```
delegate DictClose_delegate( ) as void
```

and

```
delegate EnumWindowsProc_delegate( hWnd as ptr, aWindows as array ) as word callback
```

changed to

```
delegate EnumWindowsProc_delegate( hWnd as ptr, aWindows as array ) as word
```

After these corrections, the next compile gave finally the expected number of errors:

Line	Col	Description	File
		Compilation failed (85 errors, 9 warnings)	
314	29	error XS0029: Cannot implicitly convert type 'Vulcan.Codeblock' to 'string'	DictDBServer.prg
316	28	error XS1503: Argument 1: cannot convert from 'string' to 'Vulcan.Codeblock'	DictDBServer.prg
222	17	error XS0029: Cannot implicitly convert type 'void' to 'object'	GeneralDBServer.prg
23	37	error XS0123: No overload for 'AxitGlobal' matches delegate 'AxitGlobal_delegate'	Globals.prg
24	5	error XS0619: 'Functions._RegisterExit(void*)' is obsolete: '"_RegisterExit()' is not supported. Use an event handler ...	Globals.prg
668	15	error XS0619: 'Functions.Buffer(dword)' is obsolete: '"Buffer()' is not supported, use MemAlloc() and MemFree() inste...	Miscellaneous.prg
317	3	error XS9035: The first argument to PCall must be a 'typed function pointer'.	Miscellaneous.prg
116	15	error XS0619: 'Functions.MemVarGet(string)' is obsolete: '"MemVarGet()' is not supported'	MacroEval.prg
719	8	error XS0619: 'Functions._RegisterExit(void*)' is obsolete: '"_RegisterExit()' is not supported. Use an event handler ...	DictDBServer.prg
760	5	error XS0619: 'Functions._RegisterExit(void*)' is obsolete: '"_RegisterExit()' is not supported. Use an event handler ...	DictDBServer.prg
1174	22	error XS0246: The type or namespace name '_GCDUMP' could not be found (are you missing a using directive or ...	Miscellaneous.prg
1175	22	error XS0246: The type or namespace name '_GcEntry' could not be found (are you missing a using directive or an...	Miscellaneous.prg
1192	44	error XS0246: The type or namespace name '_GcEntry' could not be found (are you missing a using directive or an...	Miscellaneous.prg
1197	4	error XS0246: The type or namespace name 'struDump' could not be found (are you missing a using directive or an...	Miscellaneous.prg
1208	31	error XS0246: The type or namespace name 'struDump' could not be found (are you missing a using directive or an...	Miscellaneous.prg
1208	71	error XS0246: The type or namespace name '_GcEntry' could not be found (are you missing a using directive or an...	Miscellaneous.prg
1213	36	error XS0246: The type or namespace name 'struEntry' could not be found (are you missing a using directive or an ...	Miscellaneous.prg

XS0028: Cannot implicitily convert type .....

This error was occurring in this part of code:

```
if cValidBlock == NULL_STRING
    cValidBlock     := MCompile( cValidString )
endif
uResult     := MExec( cValidBlock )
```

where „cValidBlock“ was defined „as string“

I have made this change (again on the VO and X# side the same):

```
#ifndef __XSHARP__
    if cValidBlock == NULL_STRING
        cValidBlock     := MCompile( cValidString )
    endif
    uResult     := MExec( cValidBlock )
#else
    if oValidBlock == null
        oValidBlock     := MCompile( cValidString )
    endif
    uResult     := MExec( oValidBlock )
#endif
```

Defining „oValidBlock as CodeBlock“

It seems that the Vulcan.NET runtime defines a different return type for the function „MCompile“, and the best option (other than to define „cValidBlock as usual“) is to use conditional compilation.

The VO compiler does not knows the compiler variable __XSHARP__, and therefore ignores the code between the „#else“ and the „#endif“, whereas the X# compiler defines this variable and leaves out the code between „#ifndef“ and „else“. This is a very elegant solution to keep the source code unique and respect different definitions. I have used this method very very often in the migration.

This change fixed the first two errors, and you will seen in your own migration that very often one change removes several errors.

error XS0619: 'Functions._RegisterExit(void)' is obsolete*

The exact error was:

error XS0619: 'Functions._RegisterExit(void*)' is obsolete: ''_RegisterExit()' is not supported. Use an event handler added to the AppDomain.CurrentDomain:ProcessExit event instead'

and it was occurring in this code:

```
    if lInit == false
    #warning Callback function modified to use a DELEGATE by xPorter. Please review.
//    _RegisterExit( @AxitGlobal() )
    static local oAxitGlobalDelegate := AxitGlobal as AxitGlobal_Delegate
    _RegisterExit( System.Runtime.InteropServices.Marshal.GetFunctionPointerForDelegate(oAxitGlobalDelegate) )
      lInit := true
    endif
```

Now, this code was modified by the xPorter because the language construct is not supported anymore in .NET, the original code was simply

```
    if lInit == false
       _RegisterExit( @AxitGlobal() )
       lInit    := true
    endif
```

But since the .NET garbage collector works in a totally different manner than the VO one, this is not more needed. So I changed the code to :

```
    #ifndef __XSHARP__
    if lInit == false
       _RegisterExit( @AxitGlobal() )
       lInit    := true
    endif
    #endif
```

About the different behavior of the .NET garbage collector, this is the explanation that Chris Pyrgas gave me:

„For example about RegisterAxit(), it works the opposite way in .NET than in VO. In .NET, destructors are always called by default and if you want to prevent the GC for calling one, you need to call GC.SuppressFinalize(). (in VO you need to call RegisterAxit() if the destructor must be called). Many of those things are also explained in the vulcan help file in the chapter about migration, I think we need to duplicate this in the x# help file..“

error XS0619: 'Functions.Buffer(dword)' is obsolete

The full error message was:

error XS0619: 'Functions.Buffer(dword)' is obsolete: ''Buffer()' is not supported, use MemAlloc() and MemFree() instead'

and the offending code

```
cBuffer:= Buffer( 4096 )
```

and I have changed it to

```
#ifdef __XSHARP__
    cBuffer := Space( 4096 )
#else
    cBuffer := Buffer( 4096 )
#endif
```

If I'm honest: I don't understand why the Buffer() function is not more supported – it returns a not initialized string. I hope the development team supports this function in their own runtime.

At the moment the function can be replaced with a call to Space()

error XS9035: The first argument to PCall must be a 'typed function pointer'.

The original code is:

```
PCall( ptrFunction, @strWinOsVersionInfo )
```

And I have changed it to

```
#ifdef __XSHARP__
PCallNative<int>( ptrFunction, @strWinOsVersionInfo )
#else
PCall( ptrFunction, @strWinOsVersionInfo )
#endif
```

So this is too an easy fix, and you can see that the #ifdef __XSHARP__ is very important to keep code compatible between VO an X#.

error XS0246: The type or namespace name '_GCDUMP' could not be found

I had this error on a long list of functions and variable definitions, all of them are related to the garbage collector, and therefore they are totally outdated. For the moment I have decided to leave them out in the X# version (again using #ifndef __XSHARP__). I hope to need to diagnose the .NET garbage collector, and if I should need that, I will search for an implementation in C# and translate then to X#

If transported code shows this error, most likely it is a function or class or structure that makes no sense to have in X# or they simply have forgotten it (I had a problem with the wsprintf() function missing from the Vulcan runtime – the X# team is aware of it and will implement it). I'm pretty sure you will encounter some missing functions too.

error XS0619: 'Functions._VOLoadLibrary(string)' is obsolete

Again, the complete error message:

error XS0619: 'Functions._VOLoadLibrary(string)' is obsolete: ''_VOLoadLibrary()' is not supported, use VulcanLoadLibrary() instead.'

The code was this one:

```
hDLL            := _VOLoadLibrary( cDLL )
if hDLL == null_PTR
    ErrBox( nil, ParmSubst( LoadResString('benötigte Bibliothek %1 nicht gefunden!',ID_BENOETIGTEBIBLIOTHEK1NICHTG,;
                    nLangHandle), cDLL ) )
    return false
else
    hProc           := GetProcAddress( hDLL, String2Psz( cInitFunction ) )
    if hProc == null_ptr
        ErrBox( nil, StrTran( LoadResString('Fehler beim Initialisieren von VOScript!',ID_FEHLERBEIMINITALIALISIERENV,;
                    nLangHandle), "VOScript", cClass ) )
        return false
    endif
    PCall( hProc )
    if ! IsClass( String2Symbol( Upper( cClass ) ) )
        ErrBox( nil, StrTran( LoadResString('Fehler beim Initialisieren von VOScript!',ID_FEHLERBEIMINITALIALISIERENV,;
                    nLangHandle), "VOScript", cClass ) )
        return false
    endif
endif
```

I'm using the dynamic loading of VO DLLs quite a lot. Originally it was to save time on application load (a DLL load before first use will not delay the application load), and only for functionality that was not needed every time (like printing/reporting or zipping). In the case of the report DLL, I could decide at runtime which DLL to load, saving not only license costs, but also use different versions depending on the environment.

In these cases I have decided to leave the load code completely out: if the relative library was in the application references, the IsClass() function would return true, otherwise an error message would be returned:

```
#ifdef __XSHARP__
return false
#else

.... remainder of code


#endif
```

warning XS1030: #warning: 'The following method did not include a CLASS declaration'

This is a warning, but IMHO it should be an error, because it refers to the following code on the VO side:

```
method AutoSize( ) class ListViewColumn
//p AutoSize-Eigenschaft der Spalte setzen
//s
```

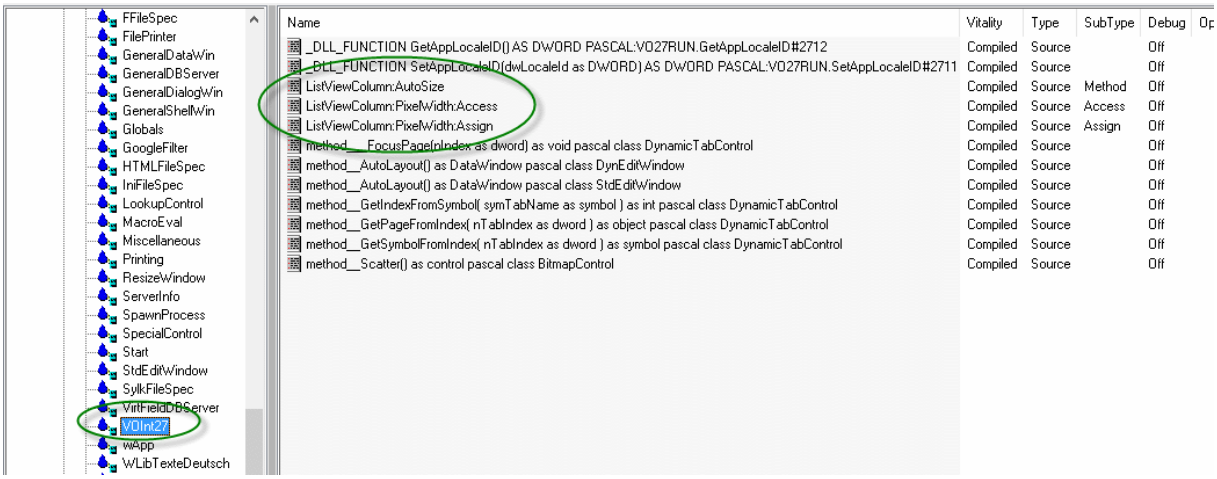
I have seen such constructs in many applications, and unfortunately they are not more supported in .NET as the .NET runtime does not let distribute the class of more assemblies. That has to do with the fact that the VO runtime creates the class method table at runtime, whereas in .NET it is builded at compile time.

To solve this problem, there are at least 3 different possibilities:

- use a subclass, i.e. class MyListViewColumn inherit ListViewColumn. In the case of this method this unfortunately does not work, as the ListView returns the ListViewColumns object as types of ListViewColumn and not MyListViewColumn
- use extension methods (I will present a sample below), with a few limits:
  when using late binding, the Vulcan runtime will not use these methods, access/assign will not work as properties cannot be extended. And as last problem: you have not access to protected instance variables inside the class.
- In most cases the best method would be to extend the VO GUI classes. Since you have to recompile them in X#, you can add also these methods.

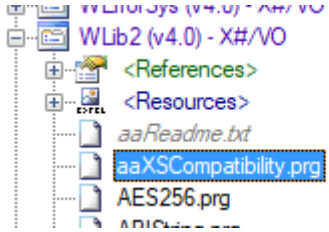
In my case, I have used the extension method. Unfortunately, in this case our friend #ifdef does not works anymore, so we have to find another method.

In VO, I have moved this type of code to a special module. When xPorting the first time, I have removed the relative prg file from the application and added another one. This is the VO side:



I had used this method in the VO 2.7 to 2.8 migration, to keep my library source compatible between different VO versions, therefore the strange name (and textblock-ed entities).

On the X# side it looks so now:



With the VOInt27 module removed and the aaXSCompatibility.prg added. Since the xPorter does not overwrite my project file, but only the prg files, I can repeat the process as often as I like without the need to reconfigure my project.

And this is the code on my X# side:

```
static class XsharpCompatibility

#region ListViewColumn extension

static method AutoSize( self oListViewColumn as ListViewColumn ) as void
//p AutoSize-Eigenschaft der Spalte setzen
//s
    local nIndex        as int
    local nWidth        as int
```

The problem continues: I have also an added assign in my VO library:

```
access PixelWidth class ListViewColumn
    local dwIndex as dword
    local nPixelWidth as int

    if oOwner != NULL_OBJECT
        dwIndex         := oOwner:__GetColumnIndexFromSymbol(self:NameSym) - 1
        nPixelWidth     := ListView_GetColumnWidth(oOwner:Handle(), int(_cast, dwIndex))
    endif

    return nPixelWidth
```

```
assign PixelWidth( nNewWidth ) class ListViewColumn
    local dwIndex as dword

    IF (oOwner != NULL_OBJECT) .and. (oOwner:CurrentView == #ReportView)
        IF (nNewWidth > 0)
            dwIndex         := oOwner:__GetColumnIndexFromSymbol(self:NameSym) - 1
            ListView_SetColumnWidth(oOwner:Handle(), INT(_CAST, dwIndex), nNewWidth)
        ENDIF
    ENDIF

    nWidth      := nNewWidth

    return self
```

This code cannot work as there is no possibility to use extension properties in .NET.

So I need to change this code completely to a pair of X# methods:

```
static method GetPixelWidth( self oListViewColumn as ListViewColumn ) as int
//p Breite der Spalte in Pixeln zurückgeben
//s
    local dwIndex as dword
    local nPixelWidth as int
    local oOwner as ListView

    oOwner          := oListViewColumn:Owner
    // if this column is already attached to a ListView control, it may have been
    // resized by the user, so try to determine the updated width, otherwise just
    // return the stored width
    if oOwner != NULL_OBJECT
        // get the current column width
        dwIndex         := oOwner:__GetColumnIndexFromSymbol(oListViewColumn:NameSym) - 1
        nPixelWidth     := ListView_GetColumnWidth(oOwner:Handle(), int(_cast, dwIndex))
    endif

    return nPixelWidth
```

```
static method SetPixelWidth( self oListViewColumn as ListViewColumn, nNewWidth as int ) as void
//p Breite der Spalte in Pixeln setzen
//s
    local dwIndex as dword
    local oOwner as ListView

    oOwner          := oListViewColumn:Owner
    // if this column is already attached to a ListView control, and the view is report view,
    // the column must be updated visually
    if (oOwner != NULL_OBJECT) .and. (oOwner:CurrentView == #ReportView)
        // calculate the pixel width of the column using the given character width
        if (nNewWidth > 0)
            dwIndex         := oOwner:__GetColumnIndexFromSymbol(oListViewColumn:NameSym) - 1
            ListView_SetColumnWidth(oOwner:Handle(), int(_cast, dwIndex), nNewWidth)
        endif
    endif

    return
```

To be prepared, I have also removed the access/assign pair on the VO side and added two new methods to the ListViewColumn class. (a change that requires to fix also all my VO applications!)

error XS0029: Cannot implicitly convert type 'void' to 'object'

This error was caused by this code:

```
if _lNotFound
    ErrBox( nil, StrTran( LoadResString('Tabelle %1 nicht gefunden!',ID_TABELLE1NICHTGEFUNDENRZ,nLangHandle), ;
                  "%1", cFileName ) )
else
    // Versuche, um den read error line 816 zu umgehen
    nTries          := 1
    lReturn         := false
    cbError         := ErrorBlock( {|oError| _Break( oError ) } )
    while ! lReturn
        begin sequence
        oServer         := super( oFileSpec, lShareMode, lReadOnlyMode, cDriver )
        if IsInstanceOfUsual( oServer, #DBServer )
            lReturn         := oServer:Used
        endif
        recover using oError
        ++nTries
        if nTries > 99
            lReturn         := true
        endif
        end sequence
    end
    ErrorBlock( cbError )
    _lAutoClose := true
endif
```

A fully valid construct in Visual Objects, but not valid anymore in .NET, as there is no possiblity to leave out the constructor chaining.

So I had to change the code to :

```
else
    // Versuche, um den read error line 816 zu umgehen
    #ifdef __XSHARP__
    super( oFileSpec, lShareMode, lReadOnlyMode, cDriver )
    lReturn         := self:Used
    #else
    nTries          := 1
    lReturn         := false
    cbError         := ErrorBlock( {|oError| _Break( oError ) } )
    while ! lReturn
        begin sequence
        oServer         := super( oFileSpec, lShareMode, lReadOnlyMode, cDriver )
        if IsInstanceOfUsual( oServer, #DBServer )
            lReturn         := oServer:Used
        endif
        recover using oError
        ++nTries
        if nTries > 99
            lReturn         := true
        endif
        end sequence
    end
    ErrorBlock( cbError )
    #endif
```

error XS0619: 'Functions.MemVarGet(string)' is obsolete: ''MemVarGet()' is not supported'

The next error is caused by the fact that the Vulcan runtime does not supports memory variables – a feature that I need in my macro evaluation. The development team has announced that the X# runtime will introduce support for these variables again. But in the meantime I have to comment out the code. This is the changed part of code:

```
    do case
    case aEvalVars != null_array .and. ( nArrPos := AScan( aEvalVars, {|o| o:symName == symName } ) ) > 0
        uValue          := aEvalVars[nArrPos]:uValue
    #ifndef __XSHARP__
    case IsMemVar( symName )
        lMemVar         := true
        uValue          := MemVarGet( Symbol2String( symName ) )
    #endif
    case IsInstanceOfUsual( oServer, #DataServer ) .and. oServer:FieldPos( symName ) > 0
        lField          := true
        uValue          := oServer:FieldGet( symName )
    case IsAccess( oServer, symName )
```

Other errors of the same series:

error XS0103: The name '_PublicFirst' does not exist in the current context
error XS0103: The name '_PublicNext' does not exist in the current context
error XS0103: The name '_MAXAS' does not exist in the current context
error XS0103: The name '_PrivateFirst' does not exist in the current context
error XS0103: The name '_PrivateNext' does not exist in the current context

These errors are caused again my my macro evaluation code:

```
function IsMemVar( symName as symbol ) as logic
    local lFound        as logic
    local symVar        as symbol

    lFound      := false
    symVar      := _PublicFirst()
    while symVar != null_symbol
        if symVar == symName
            lFound      := true
            exit
        endif
        symVar      := _PublicNext()
    enddo
    if ! lFound
        symVar      := _PrivateFirst(_MAXAS())
        while symVar != null_symbol
            if symVar == symName
                lFound      := true
                exit
            endif
            symVar      := _PrivateNext()
        enddo
    endif

    return lFound
```

Simple and effective change since there are no memvars:

```
function IsMemVar( symName as symbol ) as logic
    local lFound        as logic
    local symVar        as symbol

    lFound      := false
    #ifndef __XSHARP__
    symVar      := _PublicFirst()
```

So the X# version returns simply „false“.