Migration Guide:

From Visual Objects to X#

Preliminary note

This short guide should help migrate your Visual Objects code to X#.

Before starting, you should remember that the X# compiler is very compatible to the Visual Objects compiler, but since a program compiled with X# will run under the .NET environment it has to follow some rules. One of these rules is for example that there cannot be an access/assign with the same name as a method. This will make changes in your code necessary.

Another rule is that an access/assign must have the same type (because it is translated to a property): you cannot have a int in the access and a dword in the assign.

And I will add another limitation: in Visual Objects you could extend an existing class with new methods or access/assign. .NET does not allow this, and I will give you an example how to solve this with extension methods and properties.

Tools

At the time of writing X# was available as RC5 of the final version (only to subscribers of the FoX program – that I would recommend you to join if you are making money with your X# software).

For the migration I have used the VO-xPorter tools (Chris is very helpful and has added many features I requested).

As for Visual Objects, I have used the latest available version, 2.8 SP4b (Build 2838), but any 2.7 or 2.8 version should work. Maybe there will work also a 2.5 or 2.6 version, but these versions are very old and the class libraries that come with them may have no source or be too buggy (GrafX has done a lot of fixes during the development of Vulcan.NET).

Runtime libraries

At the time of writing, the X# project had not released any runtime library. To migrate a VO application, you will need these two types of libraries:

- The runtime libraries that contain the functions and the macro compiler (the VO equivalent is the VO28Run.dll – in Vulcan they are VulcanRT.dll and VulcanRTFuncs.dll). These libraries are distributed in both Vulcan and Visual Objects in a compiled-only form, so the X# team needs to build them from scratch
- The class libraries, like GUI classes, RDD classes and so forth. These libraries are distributed in both compiled and source format, so you can transport them to X# yourself (or use the Vulcan compiled ones). At the time of writing, the X# team recommends to use the Vulcan versions). And since this code is copyrighted by CA and GrafX, the X# team cannot distribute it.

Most likely you will also need the RDDs (the DLLs with a .RDD extension) that permit to access DBF data files. These files are distributed in a binary format also, so currently you need the Vulcan versions since the X# versions are not yet ready (again code that needs to be written by the X# team)

The need to rewrite such a lot of code of course takes time, but has at least two big advantages:

- Newly written code can be AnyCPU code without being limited to the X86 (32 Bit) platform as the Vulcan libraries
- Newly written code can be released with open source licenses, giving you more safety and the possibility to collaborate in development and bugfixing, and to have deep insights into the code

3rd Party Tools and Libraries

You should have a source code version of your tool, or better, an X# or Vulcan compiled version of it.

For the **bBrowser**, there is a Vulcan version, the bBrowser.NET. You should upgrade your existing bBrowser and use the Vulcan compiled assemblies or recompile them in X#. There should be a few compile errors to fix – maybe later Joachim Bieler will release an X# version.

If you are using **ReportPro 2**, you need to buy the Vulcan version of it and use the Vulcan assemblies. The X# team states that ReportPro compiles also in X#, but personally I don't use it so I have no experiences. Anyway, the ReportPro 2 version for Vulcan comes with full sources and contains fixes that the VO version not contains.

For **ReportPro 3** there is a Vulcan version that already compiles also in X#. Again, currently I have no experiences as I don't use it. If you like to continue to use ReportPro 3 and would like to move to X#, you should buy the Vulcan version from GrafX.

For **VO2ADO** there is a Vulcan version, and I have no doubts that Robert v.d. Hulst as main X# developer has promised to deliver an X# compiled version when the X# runtme is available. Since I'm using VO2ADO myself, in later releases of this document I will have more details.

IDE

Personally I prefer the excellent XIDE written by Chris Pyrgas in X# itself, because it is much more like the VO IDE and has several features I like (and need), for example an autoexport function or the possibility to export applications or projects into a single file. And it has plugin system – I myself have written a plugin with a few functions that even the XIDE author uses.

Therefore my description will rely heavily on XIDE – but most things will be usable also in Visual Studio.

Migration method

Personally, I cannot afford to stop my VO development during the migration process (this process will take at least a year or two for me, as I have more than 2 millions of lines of code to move), and therefore I will have to continue development in VO and transport my code very often to X#. This means that all corrections have to be made in the VO code and not on the X# side, until the VO code

is definitely dead. This method was developed and shown by Meinhard Schnoor and Dieter Crispien, so the credits go to them. And if you need some to help you in the migration process: contact them as they have a lot of experience with such migration processes. Having one of them in your project will cost some money, but save a lot of time and errors.

Migration step by step means also that you need to move your code on an application basis, starting with the most basic one, and only if this application works, you can start the migration of the next one. And please don't forget: make all changes in the VO code!! I'll let you see some solutions how to do that.

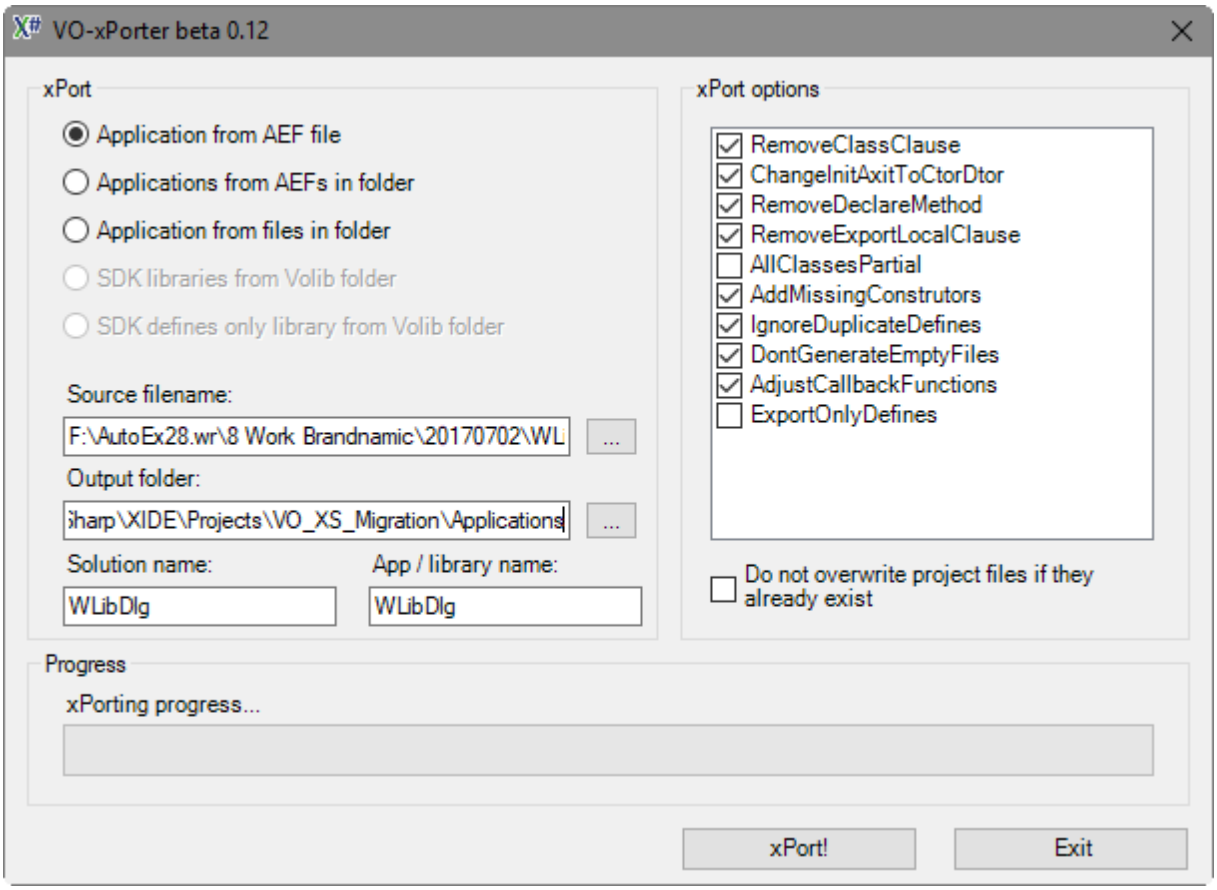
How to proceed

Export the first application/library you need to transport to X# to a folder you like

Start up XIDE and create a new project.

Select „Open Folder" from the project context menu and copy the path into your clipboard

Open the XPorter VO utility and paste the path of the applications subdirectory to the output folder control (for example: C:\XSharp\XIDE\Projects\VO_XS_Migration\Applications) – removing the application name from the path.



Select the AEF from the first step

Click on „xPort!"

(Leaving the XPorter open)

Return to XIDE and click „Add application“ from the Project context menu, go into the Applications subdirectory and then the subdirectory of the application you have created, and select the VIAPP file. The application is then added to your project.

You can then compile this application, make the changes you need and repeat the migration process as often as you need. XPorter will not touch your application configuration anymore, and will replace only the prg files. This is very important and a big time saver.

In my basic library, I have moved incompatible code (incompatible in the entity declaration) to a separate module, and removed this from the X# application. On the other side, I have added a separate prg file that contains code that is only needed on the X# side. Repeated migrations will not touch this prg file, and will not add the removed prg file, so you can repeat the migration over and over.

And if this application is complete, you can add the next application to the same project and continue your work.