Tutorial : creating a Sensomusic Usine user module on Windows

Version 1.0 (20th April 2018) by Benoit Bouchez (alias iModularSynth)
Version 1.1 (28th April 2018) : typos corrected

Introduction

This document is a short tutorial explaining how to create "from scratch" a user module for Sensomusic Usine using Visual C++ on Windows platform. It is not meant to explain how Usine modules are working, the SDK User Manual made by Martin Fleurent is perfect for that.

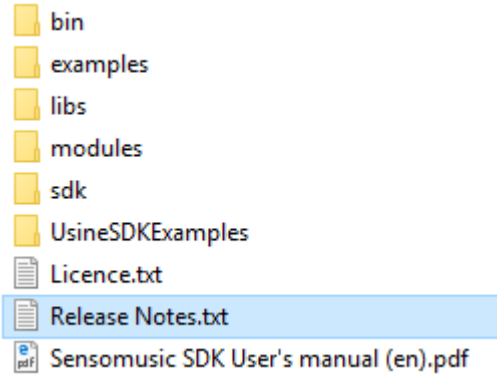
The example given here was created initially for Usine Hollyhock 2, but it should work also directly for Usine Hollyhock 3.

Preparing the computer

First of all, Visual Studio (Express or Developper edition) must be installed on your computer (and you are supposed to know how to use it...).

Download the Usine SDK on Sensomusic website and copy the whole directory on your hard disk. The SDK contains excellent examples to understand how Usine is working and what you have to write to make your own modules. We will concentrate here on the steps to follow to create modules from scratch (you can then copy source code from the projects in the SDK to have a good starting point).

Once the SDK is copied on your hard disk, you should see the following structure in the SDK folder



Technically, Usine user modules are nothing else than DLL, but you need to define some specific properties in the project to make sure that Usine can load and use them.

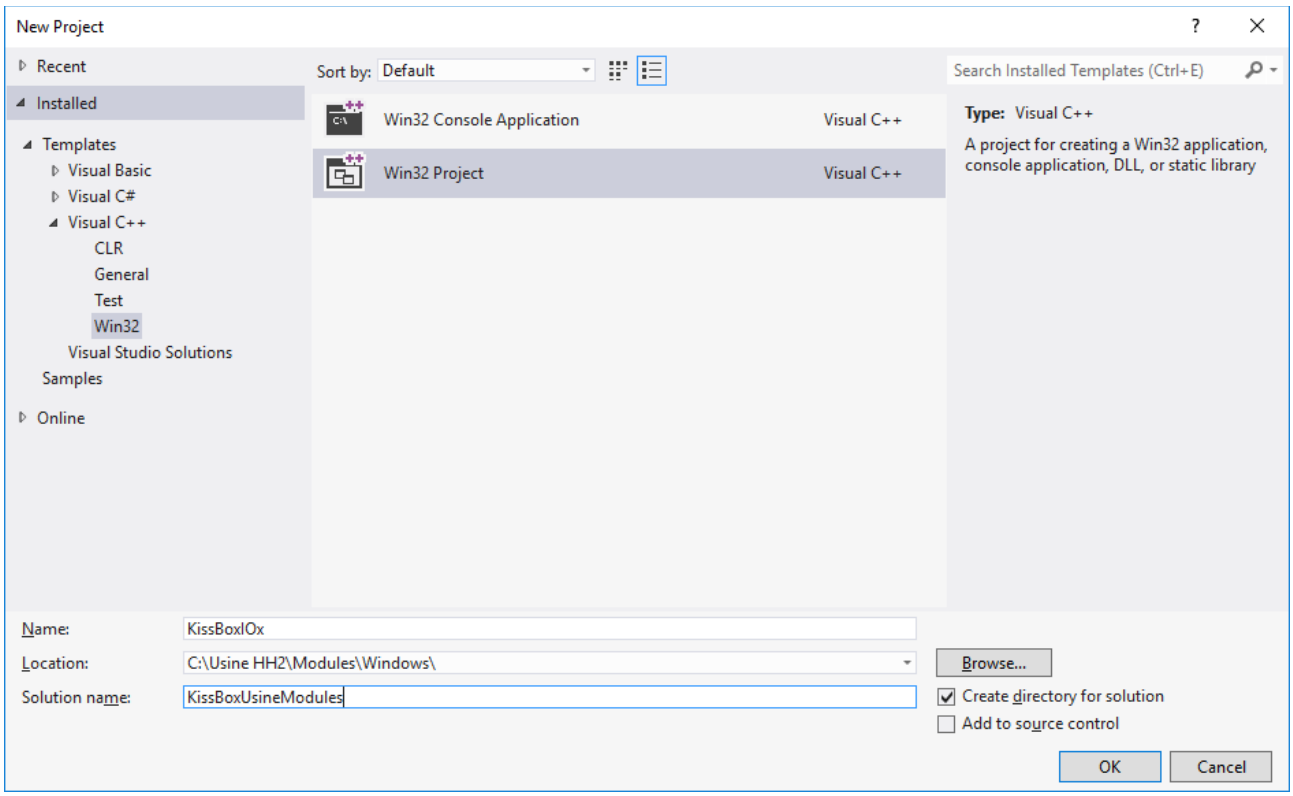
Screenshots in this tutorial are made under Visual Studio Express 2012. Other versions from Visual Studio may look slightly different but the steps remain the same.

Creating the project in Visual Studio

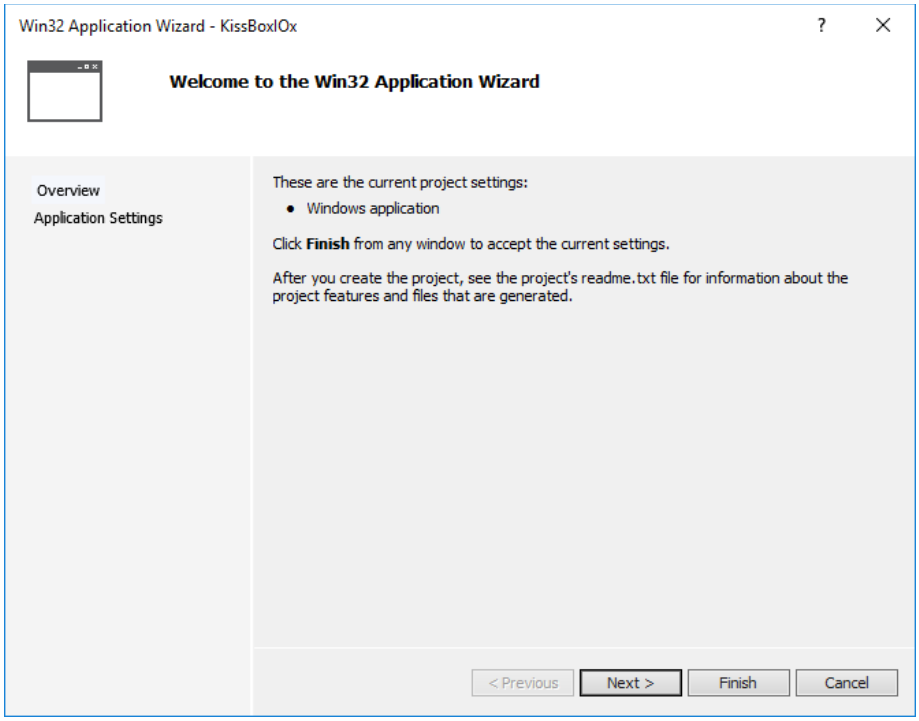
Start Visual Studio. In File menu, select New then Project. Select Win32 project.

Enter the project name. You can use here the name which will be the user module name in Usine, or use a more symbolic name and define the output filename (which will be the module name) later in the project properties.

Solution is the name given by Microsoft to a project group. You can choose to create a new project group, or use an existing one if you want to add your project in an already existing project group.

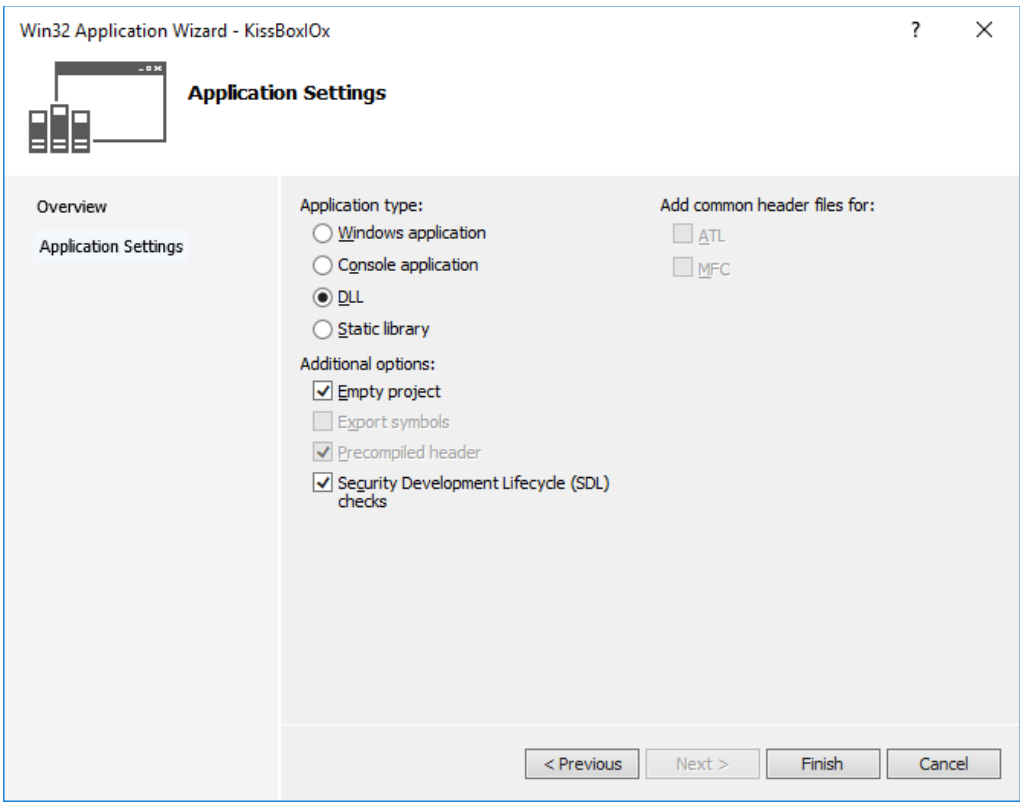


Click OK when done. A new window will now open.

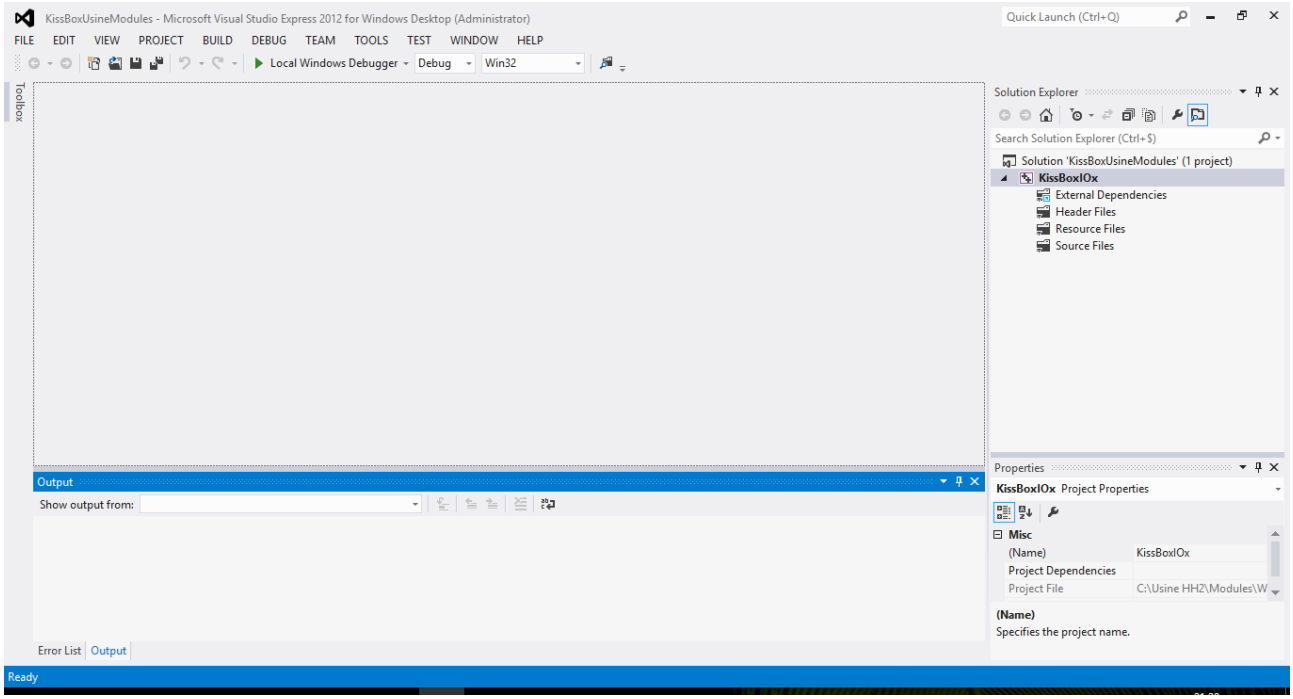


Click Next to access next window. In this new window:

- Select "DLL"
- Check "Empty project"
- You can keep "Security Development Lifecycle" checked or not, depending on your working preferences.



Click "Finish" to create the project. You will then go back to the main Visual Studio window.

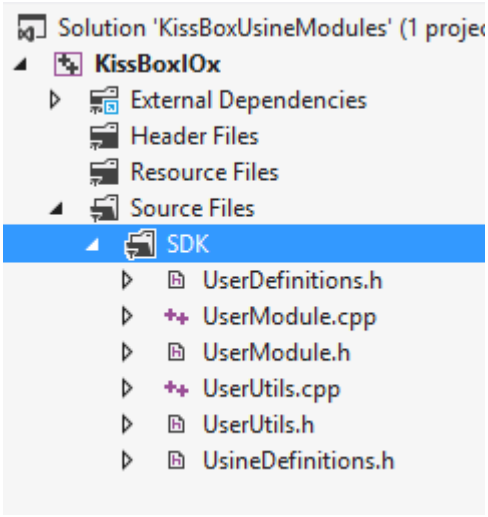


NOTE : Visual Studio puts the files in the directory specified during project creation, which may not fit your personal needs or tastes. In that case, close Visual Studio and rearrange the project directories depending on your needs. When you reopen the solution file, Visual Studio may indicate that it does not find the project anymore. Delete the project you see in the solution explorer (which should be grayed in that case), then right click on the solution and choose Add / Existing project. Select the .vcproj file you moved manually to make it available in the solution.

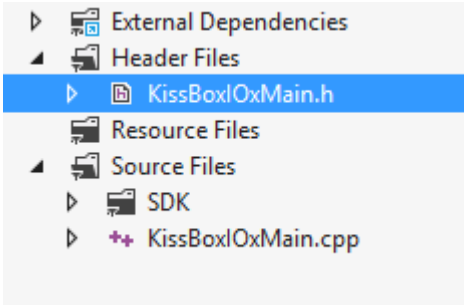
Adding files to the project

First of all, you need to add the SDK files to your project. These files are mandatory to create the user modules. You can put the files wherever you want in your project, as long as the files are part of the project. For our project, the files are added in a specific group, called SDK, placed in Source Files group.

Right click on the group in which to place the files ("SDK" in our example), then select "Add", then "Existing item". Select all the files that are in Sensomusic SDK folder. Your project should then look like the screenshot below (except the project and solution name, of course)



You can now add the project specific source files (the ones that you have to write to make your own module). Detailed explanation about these files are given in SDK User Manual from Sensomusic. Of course, the file names given in the screenshot below are pure examples, your files will be different.



DO NOT TRY TO COMPILE YOUR PROJECT NOW !

The project properties must be set first, and you may experience strange results if you try to compile the module at this step (and the module would be totally unrecogized by Usine and then, it will be useless)

Defining project properties for the Usine modules

VERY IMPORTANT : the steps described hereafter must be repeated for each build configuration you want to use for your modules (32bits/64bits – Debug/Release). In order to keep this document simple, only the 32bits / Debug configuration process is described here.

Open the project properties (Project / Properties or right click on the project in the solution explorer)

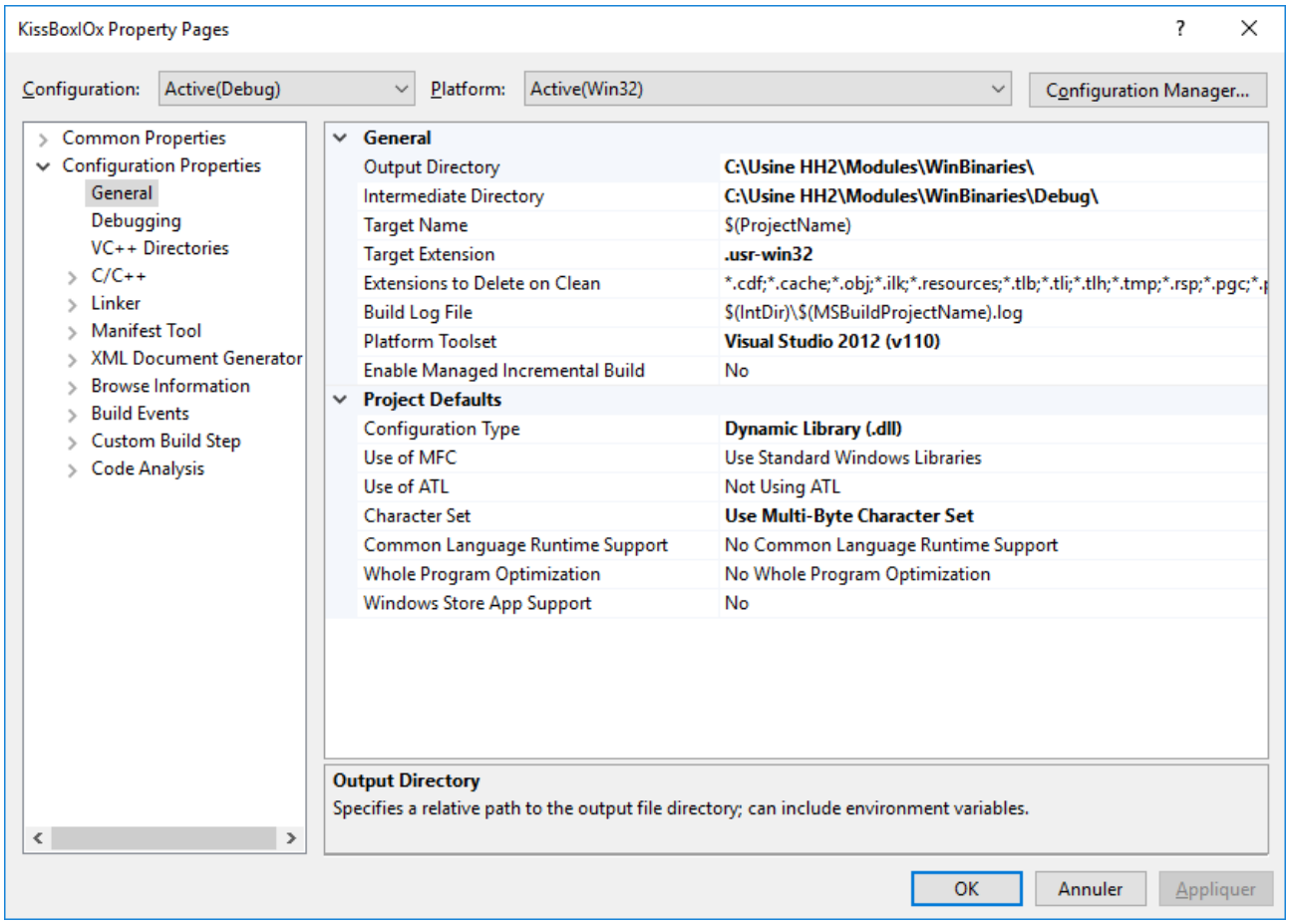
Go to General entry

Select the directory where output and intermediate files will be written by the compiler. I personnally use a specific directory, outside from the project, to avoid to copy the compiled files when I make a project backup. You can declare any folder depending on your needs.

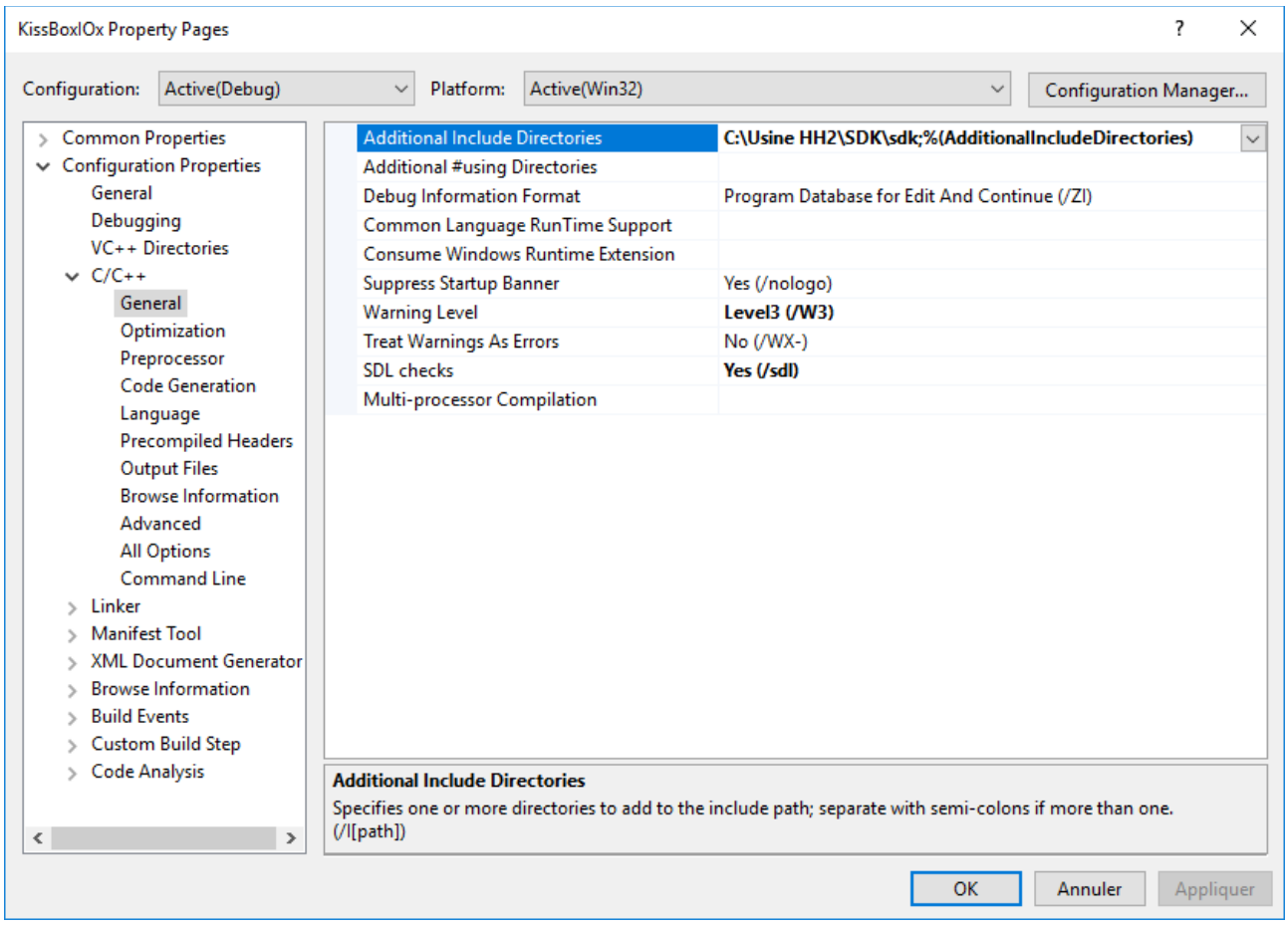
It is also **mandatory** to change the following values :

- Target Extension must be set to **.usr-win32** if you compile a module for Usine 32bits
- Target Extension must be set to **.usr-win64** if you compile a module for Usine 64bits
- Character Set must be set to **Use Multi-Byte Character Set**

If you do not follow these requirements, your module will at least not be recognized by Usine, and it will crash in worst case at the first time it will try to access a string.

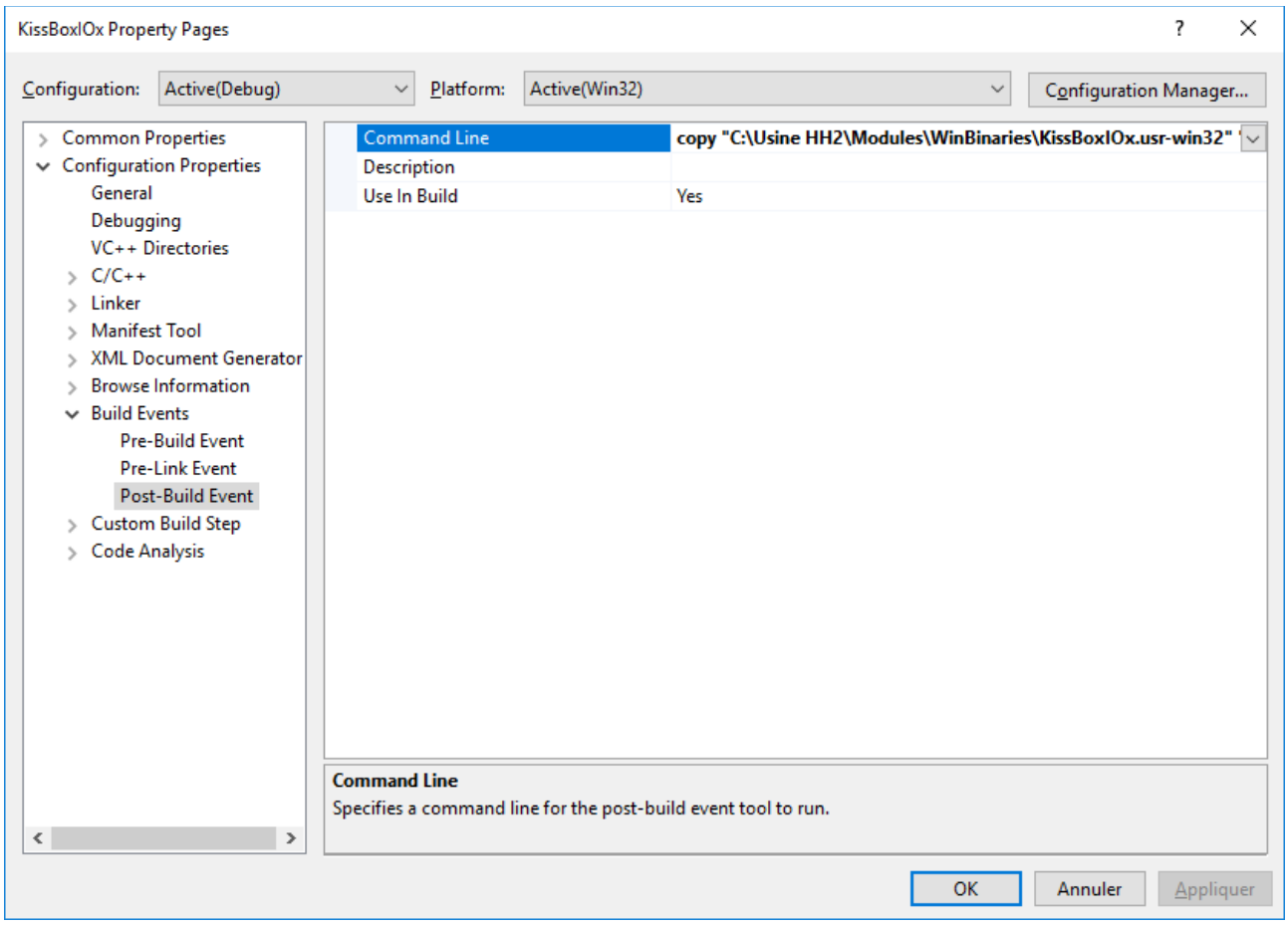


Go in the C/C++ entry, then select "General" page and enter the Usine SDK directory path in "Additional Include Directories"



If your module uses the functions like sprintf function, it is recommended to add _CRT_SECURE_NO_DEPRECATE in the Preprocessor definitions to avoid multiple warnings when compiling.

To finish, I recommend to go in Build Events to copy automatically your module executable in Usine installation directory. You can basically copy the file in any directory as long as you configure Usine to find it. Personnally, I have created a dedicated folder located in Usine\Resources\Modules\All Modules, so Usine can find them automatically when it starts.



You can now compile your module and test it in Usine.

How to test and debug your module

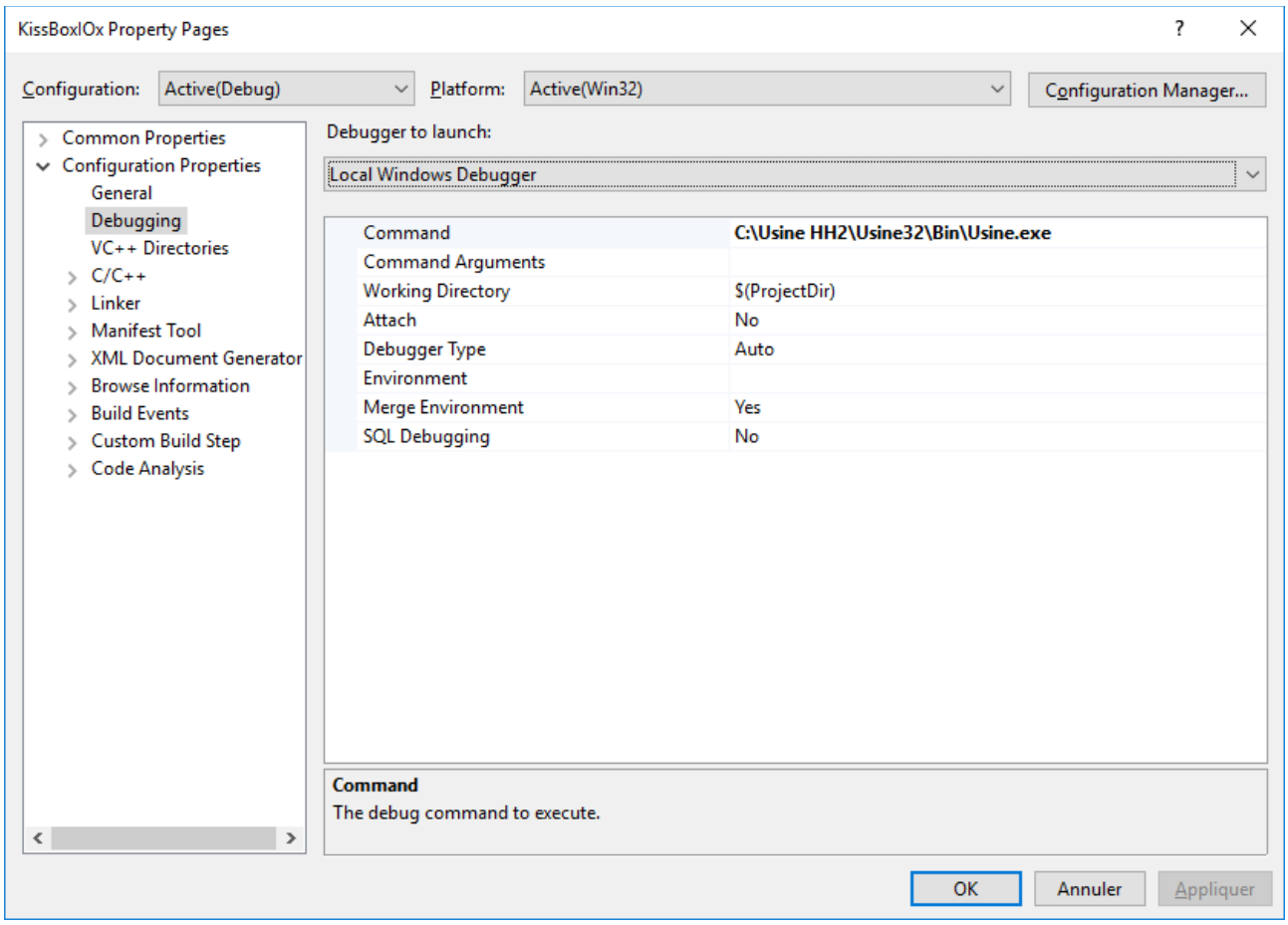
Usine has plenty of cool features, one of them being that it does not rely on a painful copy protection scheme that forces you to use a runtime to debug your creation (some developers will understand me quickly... °-)

You can compile your module and debug it directly in Visual Studio while Usine is running. You can then put breakpoints, go step-by-step, etc... in your code and debug it efficiently.

In order to use Visual Studio debugger, you need to declare the executable used to start your module (don't forget that a user module in Usine is nothing than a DLL, it can't then run by itself)

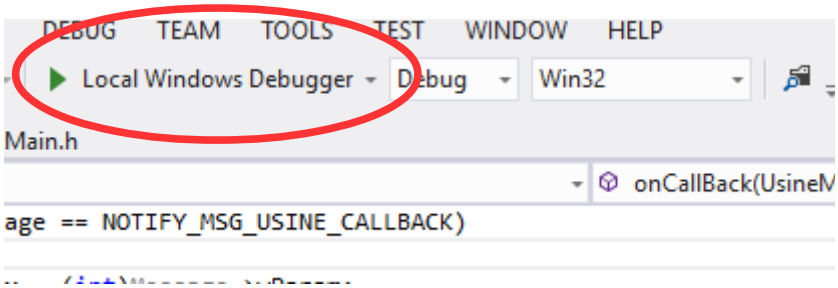
In project properties, go to Debugging tab. Click on Command Line editing area and when the drop down list appears (down arrow on the right), scroll to "Browse". Browse to the Usine executable.

VERY IMPORTANT: do not select the Usine.exe which is located in the Usine root directory, but use the Usine.exe located in the \bin directory otherwise your breakpoints will never work.



Click on OK.

Once your module is compiled in Visual Studio, click on the green arrow "Local Windows Debugger" on toolbar (or you can use Debug / Start Debugging menu)



Usine will now start. If you put a breakpoint in your project code, you will see that they appear as white dots until you put the module in a Usine project. Once your module is instanciated in a Usine project (or when the navigator window opens), you will see the breakpoints becoming active, since the DLL is only loaded when Usine needs it.