Setting up Visual Studio for Voxon Application Development

Visual Studio is a powerful integrated development environment (IDE) and is widely used for software development especially for Windows based applications. This guide will demonstrate how to set up *Visual Studio* to be used for Voxon application development.

This guide was written for *Visual Studio* 2017 and 2019. It should work with older or new versions of *Visual Studio* just watch these common 'gotchas': Your project must have access to the Voxon libraries (.dll files) and header files (.h files) and your project's 'Character Set' settings need to be set to 'Use Multi-Byte Character Set'.

Note: Voxon volumetric applications (a "VX app") can only be developed on Windows based machines.

Step 1: Installing Visual Studio, the Voxon Developers Kit and setting the path variable to include voxiebox.dll

Before you begin developing C/C++ Voxon applications you will need two pieces of software installed on your system:

1) *Visual Studio* (not *Visual Studio Code*) with the workload 'Desktop development with C++' installed. (Download here: https://visualstudio.microsoft.com/.)
2) The Voxon Developers Kit (Download here: https://github.com/Voxon-Photonics/Content-Developers-Kit)

1.1 Installing Visual Studio

Download *Visual Studio* from https://visualstudio.microsoft.com/. Note ***Visual Studio*** and ***Visual Studio Code*** are two different programs. The community version of *Visual Studio* is free. When you install *Visual Studio* make sure the workload 'Desktop development with C++' is installed. This can also be added to an already installed version of *Visual Studio* by running the ***Visual Studio Installer*** program.

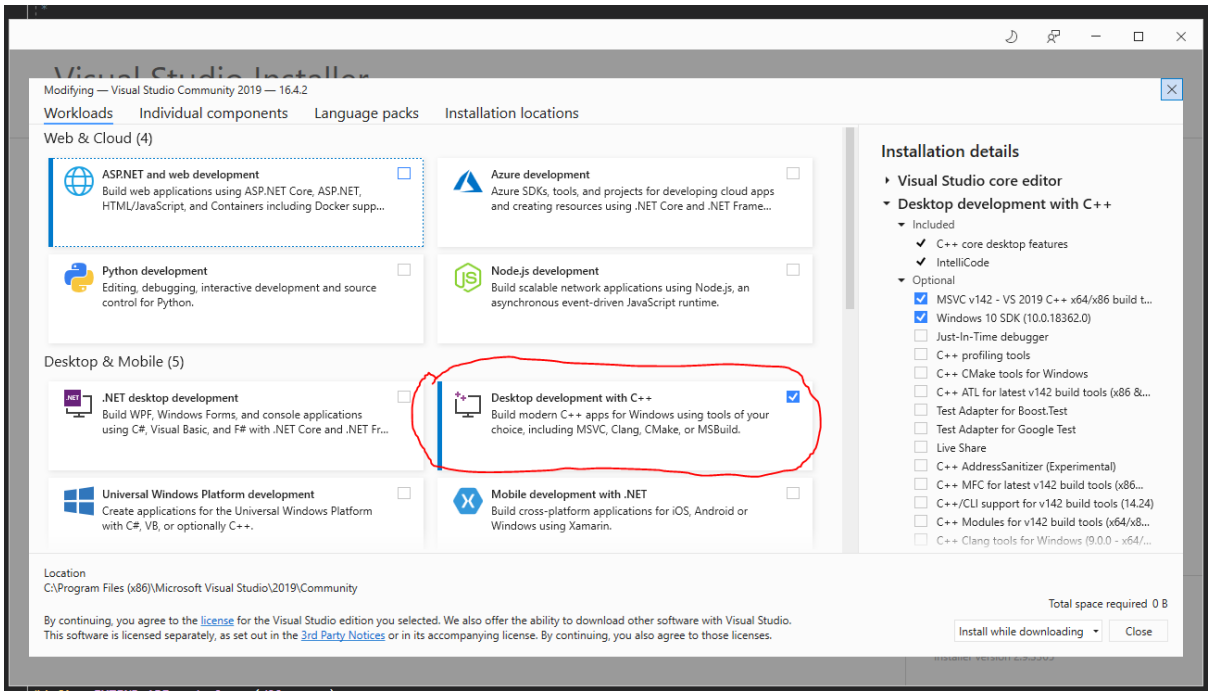

Figure 1 Desktop Development with C++ needs to be installed in Visual Studio for Voxon Development to work.

1.2 Installing The Voxon Developers Kit

The Voxon Developers Kit can be downloaded from Voxon Photonics website. For the **most recent stable release** the Developers Kit can be downloaded via the Voxon Installer which can be downloaded from here: https://voxon.co/sdk-unity/

For more experienced users, the very latest build can be obtained at Voxon's GitHub page: https://github.com/Voxon-Photonics/Content-Developers-Kit.

We recommend extracting the Voxon Developers Kit to C:\Voxon\ but you can install it anywhere.

1.3 Ensuring the path variables are set on your machine (if you side loaded via GitHub)

The Voxon Installer adds the Voxon runtime files to the environment path variable. However, if you installed the Developers Kit from GitHub, you will need to add the Voxon runtime files to the environment path variable. If your path variable is not set up, VX applications will only work if the *voxiebox.dll* is in the local directory.

To set up the path variables and registry see the '…/System/Setup' folder there you will find the files to help you set up these settings.

To check that the path environment variable has been set you can check by typing 'Edit The System Environment Variables' in the start button's search. load up the System Properties and click 'Environment Variables' and ensure that the Path variable has access to the *voxiebox.dll* (by default C:\Voxon\System\Runtime).

If it's not there you can edit the path variable to include the directory where *voxiebox.dll* is located.

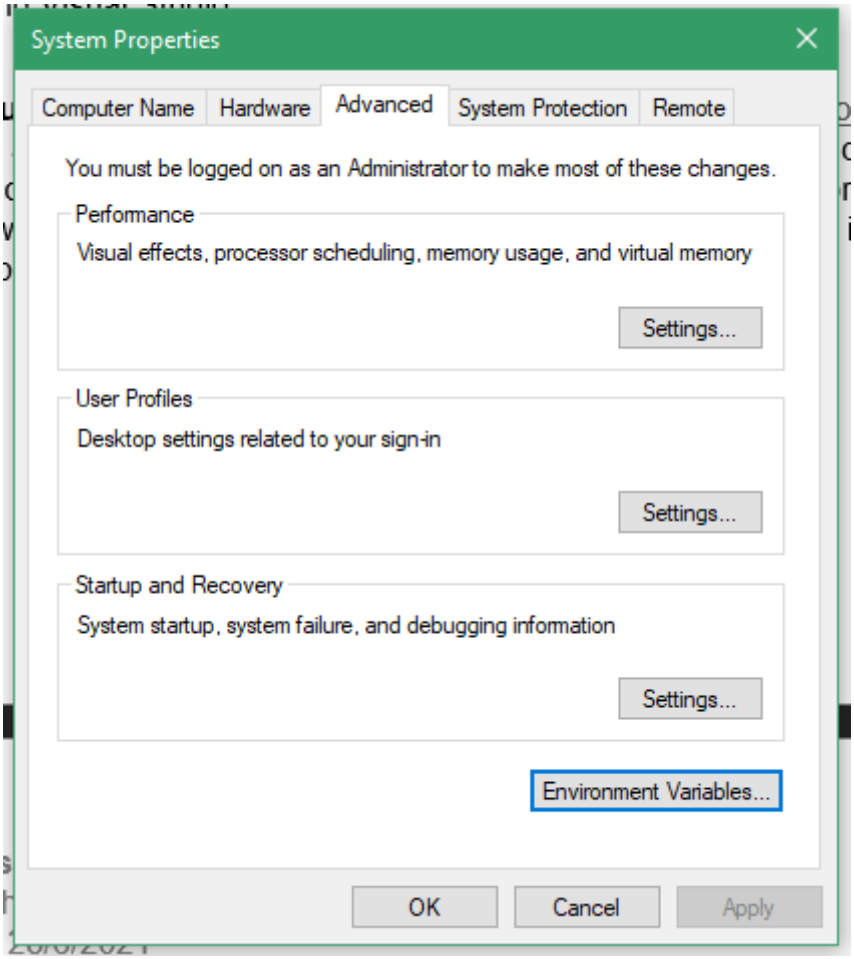

Figure 2 The System Properties from Control Panel.

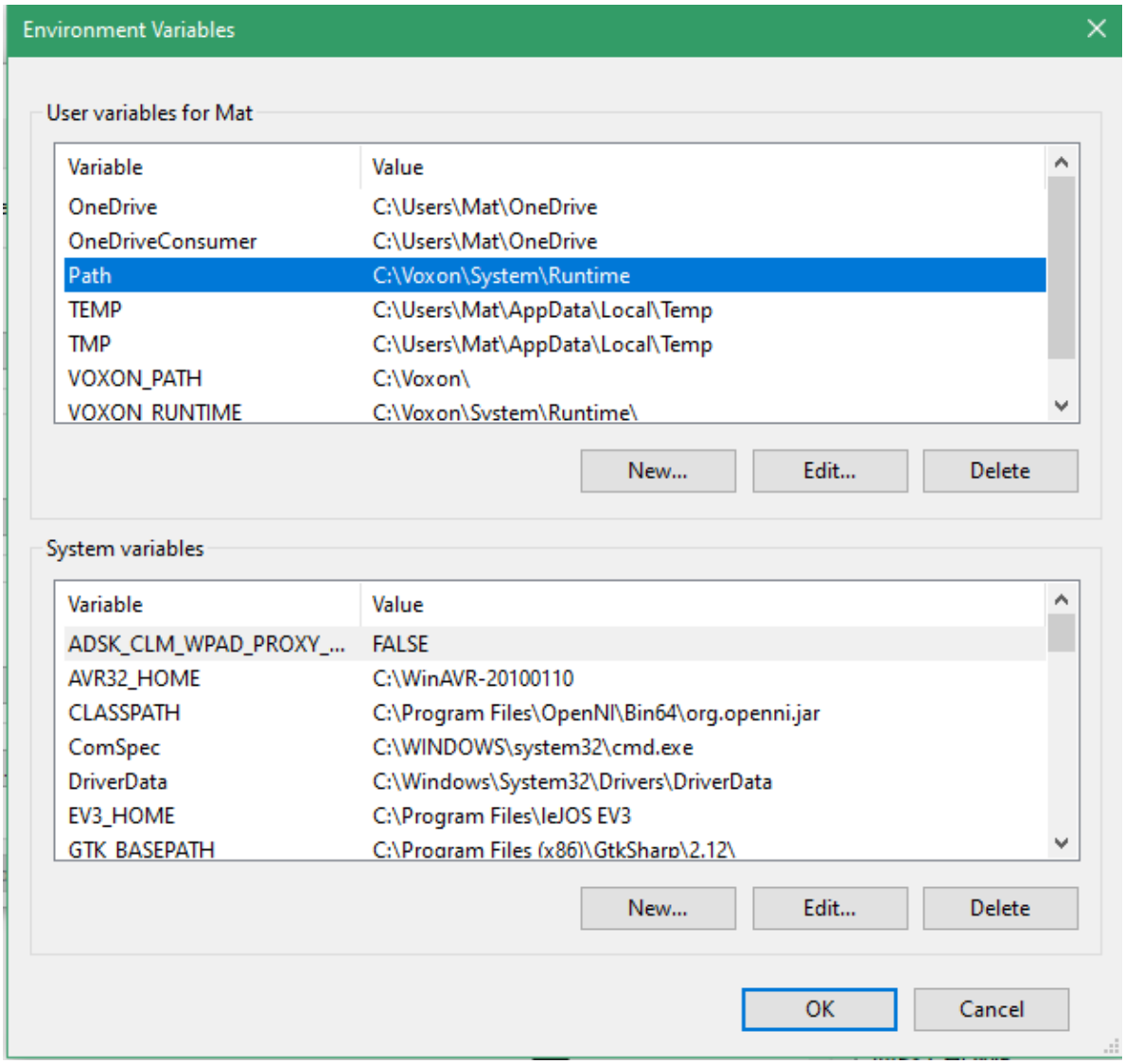

Figure 3 Make sure the Path variable has the Voxon Runtime directory included.

Step 2: Setting up a Visual Studio Project for Voxon Development.

Now that *Visual Studio* and The Voxon Developers Kit (and the environment path variable has been set). You are ready to start setting up *Visual Studio* to develop your own VX applications.

2.1 Choosing the right Voxon Development Framework

There are two Voxon frameworks for Voxon development, the C++ class approachm "VX++ framework' and the original C Voxon "VX framework".

Method A: "VX++ framework" Using a C++ class approach. (Recommended)

A newer framework to make development for Voxon Applications easier. The VX++ framework uses C++ and works by creating a 'VoxieBox' object to access the core Voxon library. The VoxieBox object includes all the functions from the Voxon API, extra helper functions and quality-of-life changes.

This approach requires *VxCpp.h*, *VxCpp.dll*, *VxInputTypes.h*, *VxInterfaces.h*, and *vxDataTypes.h* (all these files are included in the Voxon Developers Kit).

Recommended for most developers as the quality-of-life changes and ability to use object orientated design principles makes it the more practical approach for most use cases.

A VX++ app requires *VxCpp.dll* and *voxiebox.dll* to run. (Both files are part of the Voxon Runtime).

Method B: Original "VX framework" uses native C approach.

Ken Silverman's original Voxon API. This is a pure C approach to developing VX applications.

This approach only requires the *voxiebox.h* file.

For developers who want to write in pure C and want the purest form of the Voxon API

Please note developing this way can lead to some problems when expanding your project to multiple files as including the *voxiebox.h* into multiple files can cause some compiling / linking issues.

A VX application only requires *voxiebox.dll* to run.

Note: if you wish to learn how to set up Visual Studio for the original VX framework skip down a few pages below.

2.2 (Method A) Step by step of setting up Visual Studio using the VX++ Framework.

2.2.1 Create a new Visual C++ project

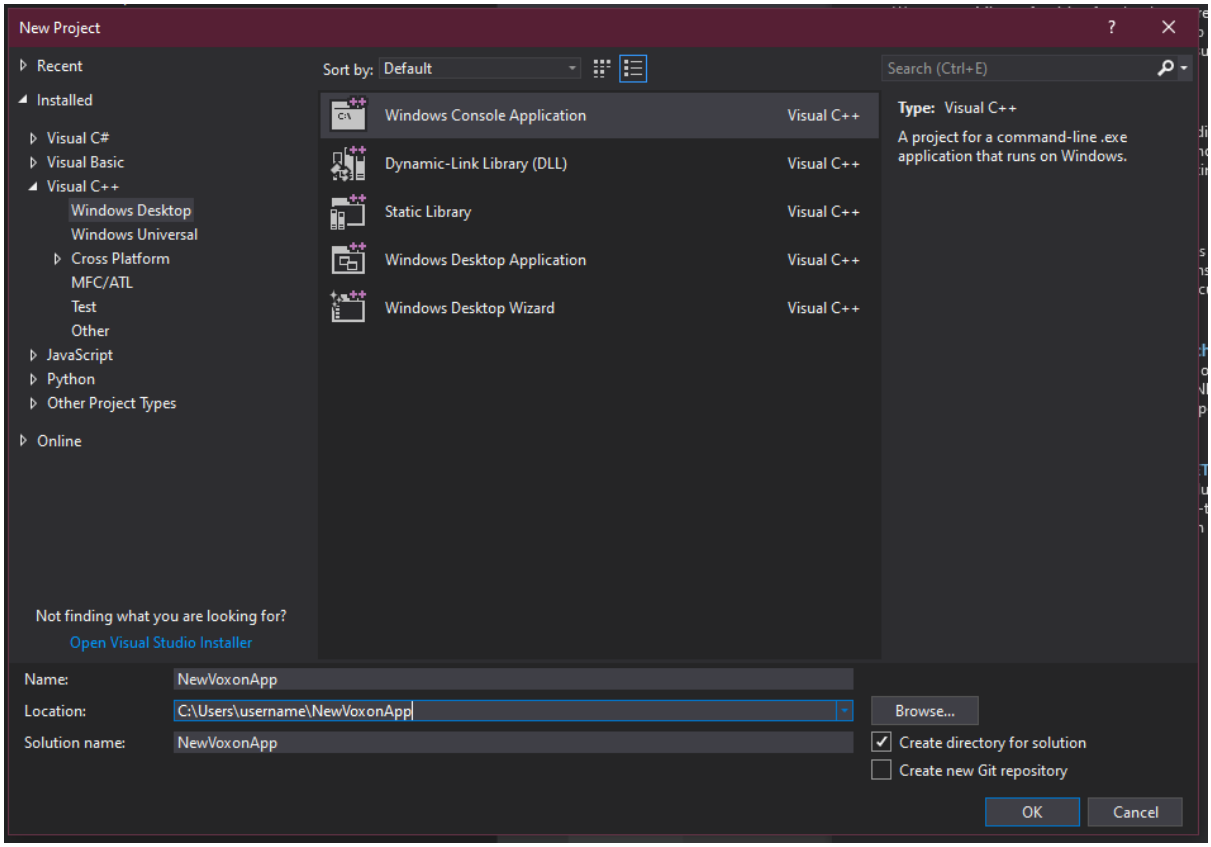

Figure 4 Creating a Visual C++ Project in Visual Studio

In *Visual Studio* create a new project and select a Visual C++ template. (If Visual C++ templates are not available you need to run the *Visual Studio* Installer and install the C++ development workload.)

You can name the project whatever you like. But keep in mind it can be difficult to change later. A ‘Windows Console Application’, ‘Console App’, or an ‘Empty Project’ are all fine. *Visual Studio* slightly changes the names with every release.

2.2.2 Set the target platform to a 64-bit application.

By Default, *Visual Studio* wants to build an x86 (32 bit application). Change the target platform to x64 (64 bit application) as the Voxon API can only build 64 bit applications.

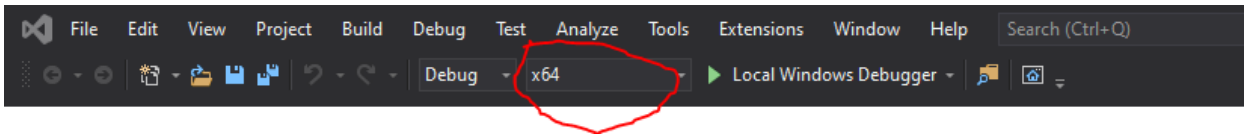

Figure 5 Ensure your target platform is x64

2.2.3 Create or replace the main source file with the VX++ Hello World text

Depending on which template you started from. Either create a new *main.cpp* (call the file whatever you want…) file or edit the existing main source file.

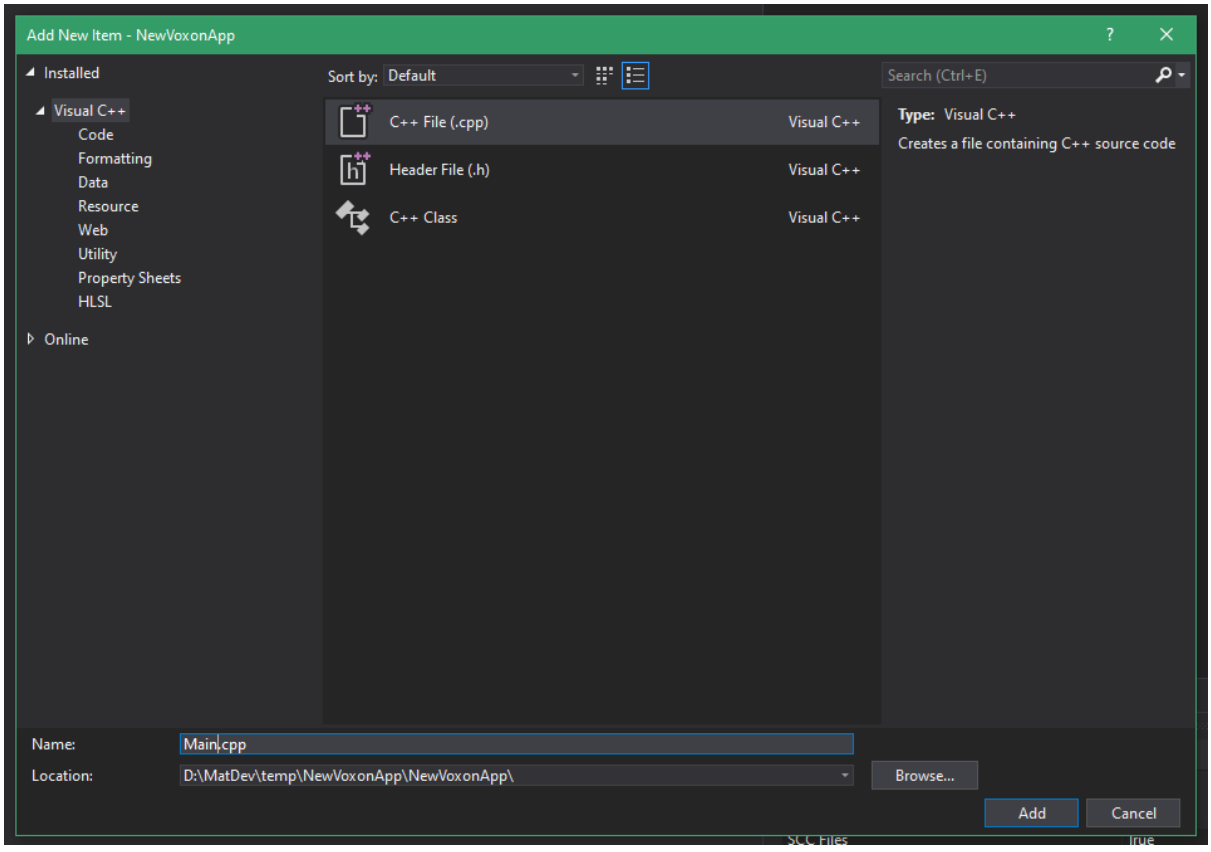

Figure 6 If no source files exist in your project make a new Main.cpp

As a starting point paste in the VX++ App Hello World Example.

```
// VX++ APP HELLO WORLD EXAMPLE
#include "vxCPP.h"
// You may use this as a template to start creating your own volumetric
applications.
int main(HINSTANCE hinst, HINSTANCE hpinst, LPSTR cmdline, int ncmdshow)
{

	// load in the vxCpp.dll so we can create the VoxieBox object...
	HINSTANCE _Notnull_ hVoxieDLL = LoadLibrary("vxCpp.dll");
	if (hVoxieDLL == NULL) return 1;
```

```
	// Access and call "CreateVoxieBoxObject" function from vxCpp.dll. The
CreateVoxieBoxObject function creates a new VoxieBox object.
	CREATE_VOXIEBOX pEntryFunction = (CREATE_VOXIEBOX)GetProcAddress(hVoxieDLL,
"CreateVoxieBoxObject");

	// Set a new IVoxieBox pointer to point to the entry function (the voxiebox
object) now 'voxie' get access to the VoxieBox class
	IVoxieBox* voxie = pEntryFunction();

	// variables to setup text positions to display 'Hello World' -- feel free to
delete this just
	point3d textPos{ -0.5, 0, 0 };	// text postions x,y,z values
	point3d textWidth{ 0.1,  0, 0 };	// text rVector  x,y,z ... the x value
determines the width of the text the other values deal with rotating text
	point3d textHeight{ 0, 0.15, 0 }; // text dVector  x,y,z ... the y value
determines the height of the text the other values deal with rotating text

	voxie->setBorder(true); // if true draws a border around the perimeter of the
display

	// Update loop for program -- breath() -- is a complete volume sweep. Called
once per volume.
	while (voxie->breath())
	{
		voxie->startFrame();	// The start of drawing the Voxieframe (all
voxie draw calls need to be within this and the endFrame() function)
		voxie->drawText(&textPos, &textWidth, &textHeight, 0xffffff, "Hello
World"); // draw text onto the volumetric display.
		textPos.z = cos(voxie->getTime()) / 5; // move the text's Z position
over time using cos (creates a moving up and down effect)
		voxie->debugText(35, 100, 0xffffff, -1, "Hello World On the Touch
Screen!"); // draw text onto the secondary (touch) screen.
		voxie->showVPS(); //show VPS data and VXCPP.DLL version unto the touch
screen.
		voxie->endFrame(); // the end of drawing to the volumetric display.
	}

	voxie->quitLoop(); // quitLoop() stops the hardware from physically moving
and ends the breath() loop
	delete voxie;
	return 0;	// After the program quits the deconstructor for voxiebox frees
the DLLs from memory if you wanted to do this manually call the voxie->Shutdown()
}
```

2.2.4 Set up references (or add local copies) of VX++ Runtime development files.

You may notice that there will be a red zigzag line under the #include *vxCPP.h* line this is because the *Visual Studio project* doesn't know how to locate the *vxCpp.h* file. The *vxCpp.h*

is the main header needed to make VX++ applications.

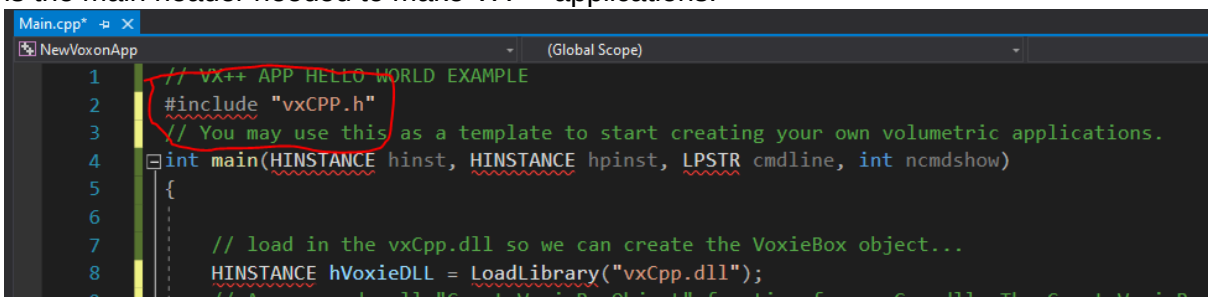

Figure 7 Additional include directory needs to be set in the project's settings otherwise Visual Studio can't find the VX++ development files

For a VX++ application development you will need to setup the paths to the following files.

VxCpp.h	The header file for VxCpp.dll
VxCpp.dll	The library which contains the VoxieBox and IVoxieBox classes
vxDataTypes.h	Header file for various data types used by volumetric applications
vxInputTypes.h	Header file for definitions for various input types for volumetric applications
vxInterfaces.h	Header file for definitions for various interfaces used for volumetric applications.

You can either copy / add these files into the project or you can set up an additional include directory in the project's settings. To edit your project's settings, right click on the project in the solution explorer and select 'Properties'.

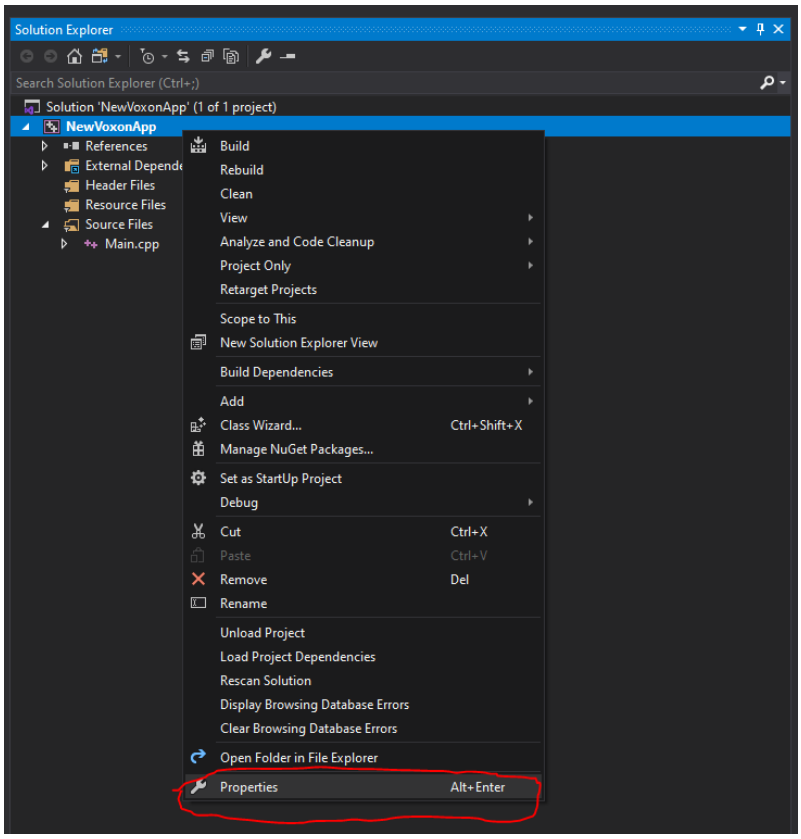

Figure 8 Right click on the Project and select 'Properties'

Look under the C/C++ tab and add a directory pointing to the *VxCPP.dll* and header files located in the **'Headers and DLLs'** folder. The file is located in the Developer's Kit path:

...\ Developers Kit \ App Development \ C++ (VxCpp) Development

This can be an absolute path or relative.

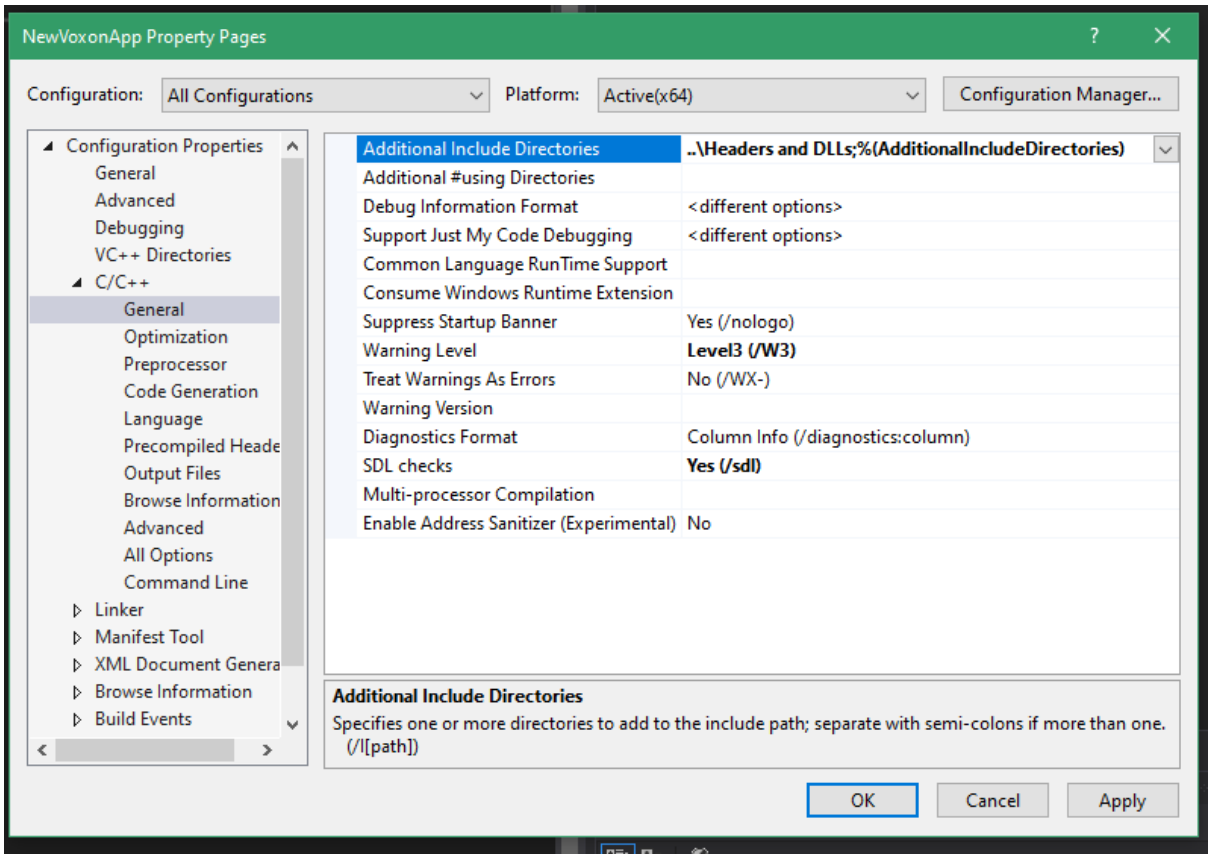

Figure 9 Make sure the VxCpp development files are included in your VS project

Copying the development files into the project would make the project more portable but you would have to copy the development files in again when an update is released. Having setup the additional Include directories makes it easier to update multiple projects at once (if they all are set to the same include path).

Note: Make sure when you make this adjustment 'Configuration:' in the top left corner is set to 'All Configurations' or it has been setup for both Debug and Release

2.2.5. In Project Settings set Character Set to ‘Use Multi-Byte Character Set’

By default, *Visual Studio* sets the Character Set to be ‘Use Unicode Character Set’ this will cause issues for Voxon development and needs to be changed to “Use Multi-Byte Character Set”.

In *Visual Studio 2017* this option can be found under the ‘General’ tab. In *Visual Studio 2019* this option can be found under the ‘Advanced’ tab.

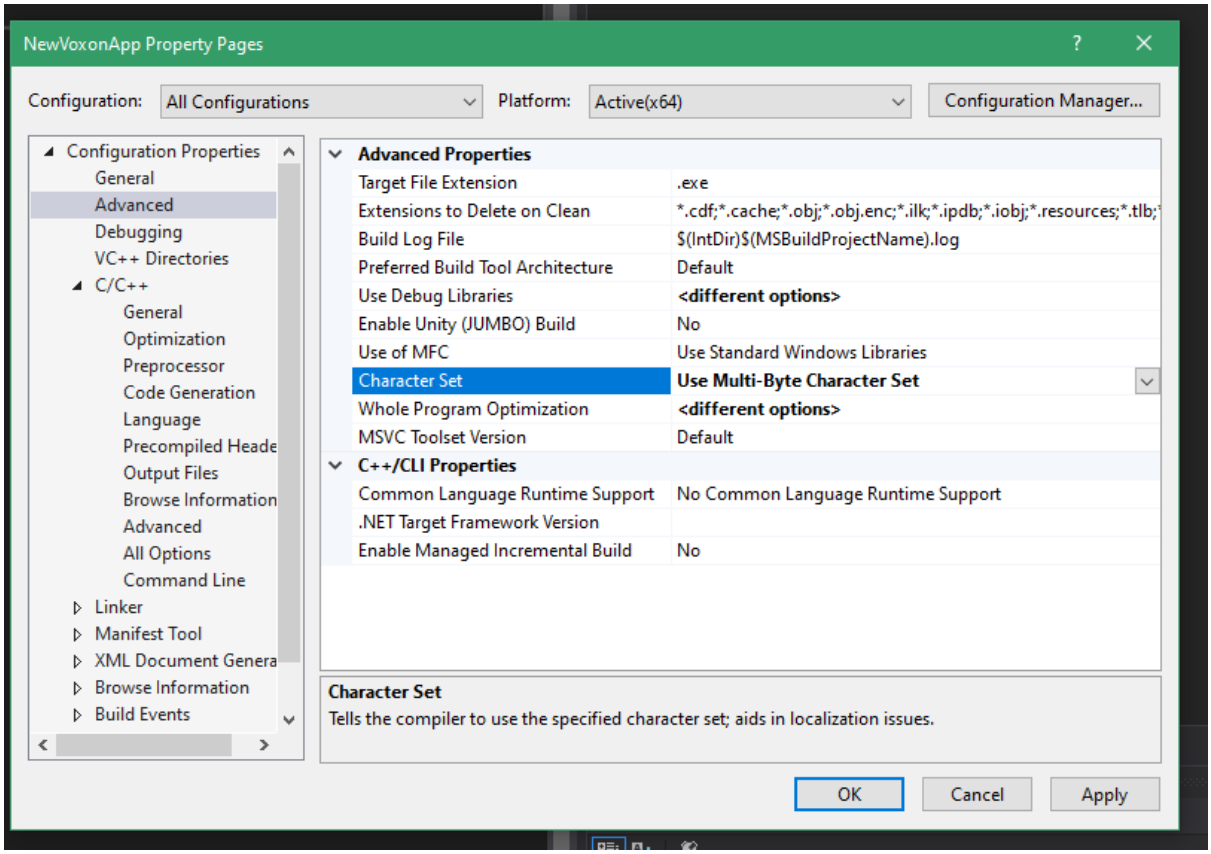

Figure 10 Voxon Development requires ‘Character Set’ to be set to 'Use Multi-Byte Character Set’.

2.2.6. Change the Runtime library to “Multi-threaded (/MT) to make the app more portable

By default, programs compiled with *Visual Studio* will require VS runtime files on the end user’s system. (*MSVP140.dll* etc) you can embed these files into your application, so your users won’t require the *Visual Studio Runtime* files to be installed.

To change this, go to the project's properties and under C/C++ Code Generation. Change the runtime library to Multi-threaded (/MT) for a Release configuration and Multi-threaded Debug (/MTd) for the Debug configuration.

This results in a slightly larger ‘.exe’ file but the benefit is you won't have to worry about people having the *Visual Studio Runtime* files installed on their device.

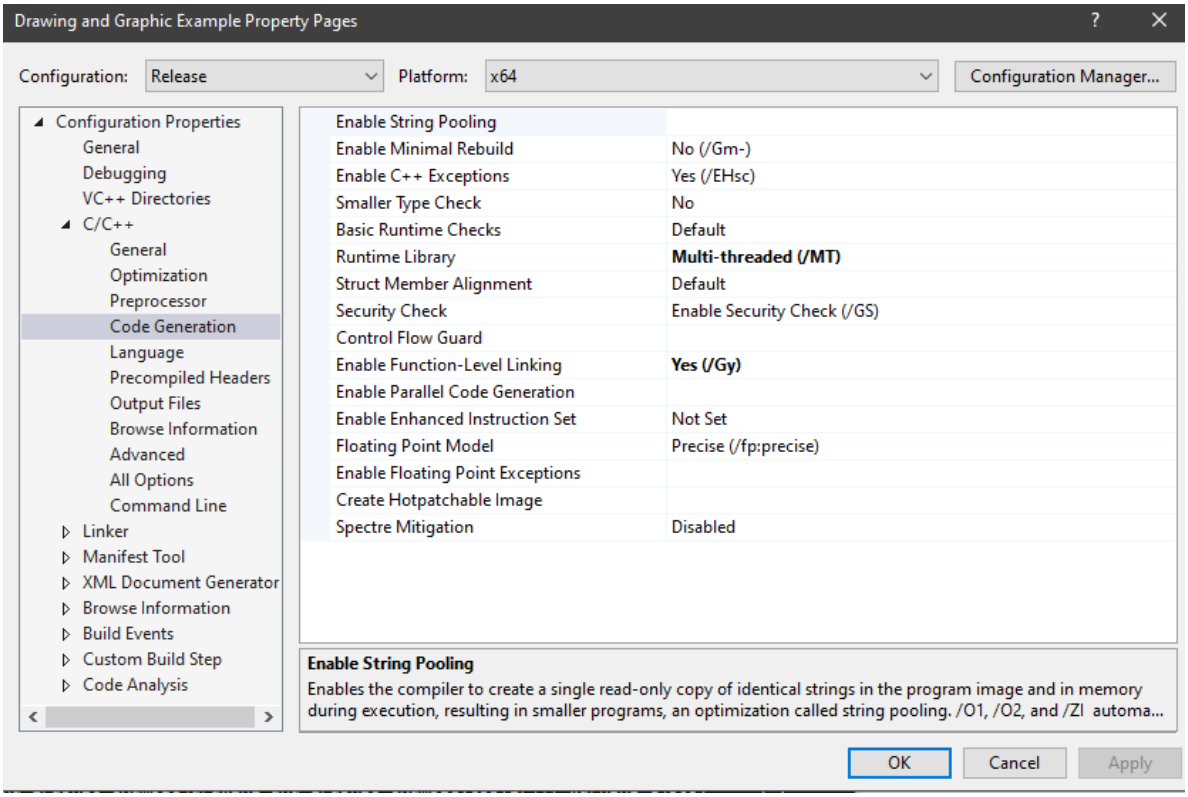

Figure 11 Set the 'Runtime Library' To Multi-threaded (or with debug) so end users won't need VS runtime installed.

2.2.7. Test that your App can be compiled.

Your VX++ App is ready to be launched. Click the Green arrow to run the application in the local debugger.

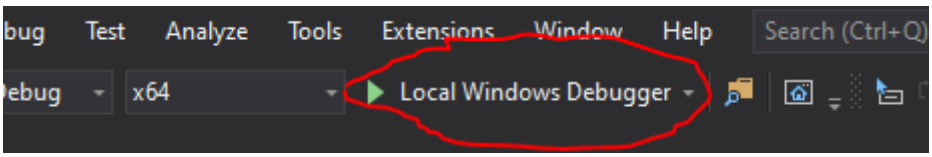

Figure 12 The green arrow launches your application in the debugger.

The VX++ Hello World application should open…

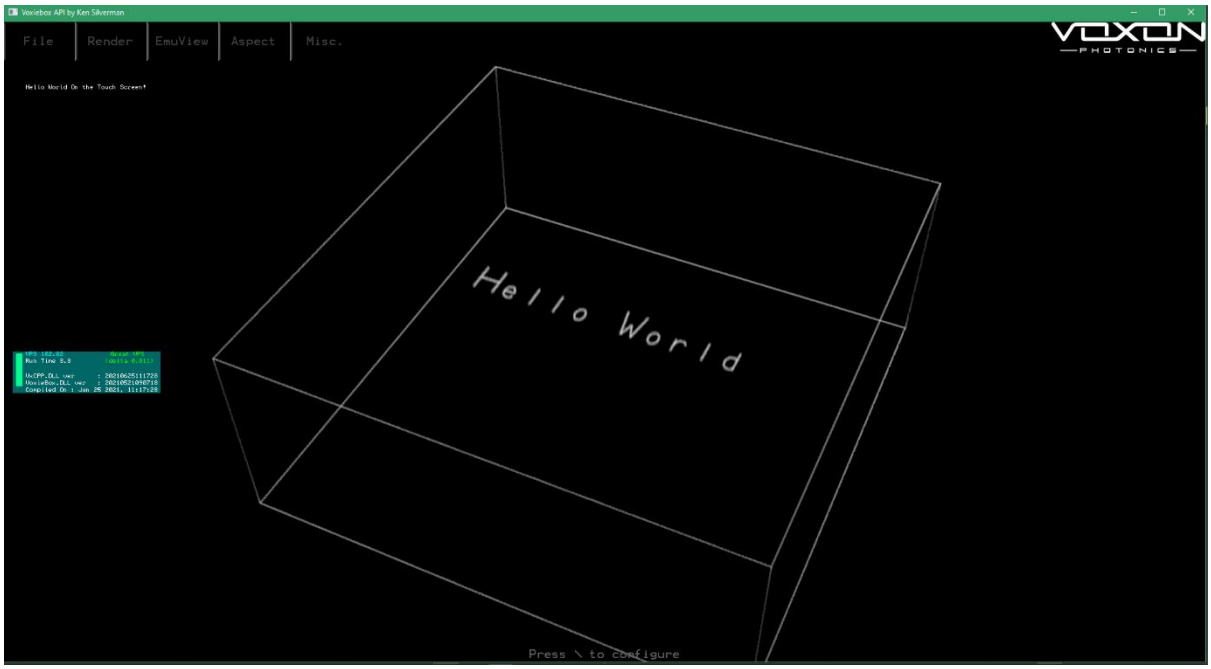

Figure 13 A successful VX++ Hello World Application

Any applications built using VX++ (*VxCpp.dll*) will require access to both *voxiebox.dll* and *VxCpp.dll* to run.

You can either distribute your app with *voxiebox.dll* and *VxCpp.dll* in the local directory or assume the end user will by running the app on Voxon hardware (or have the Voxon Developers Kit installed on their system).

For function and reference documentation about the VX++ Framework *'*.../Developers Kit/App Development\C++ (VxCpp) Development\Documentation*'.*

2.3. Method B: Step by step of setting up Visual Studio using the VX Framework.

Note: setting up the VX Framework requires fewer development files but a few extra settings need to be adjusted.

2.3.1 Create a new Visual C++ project

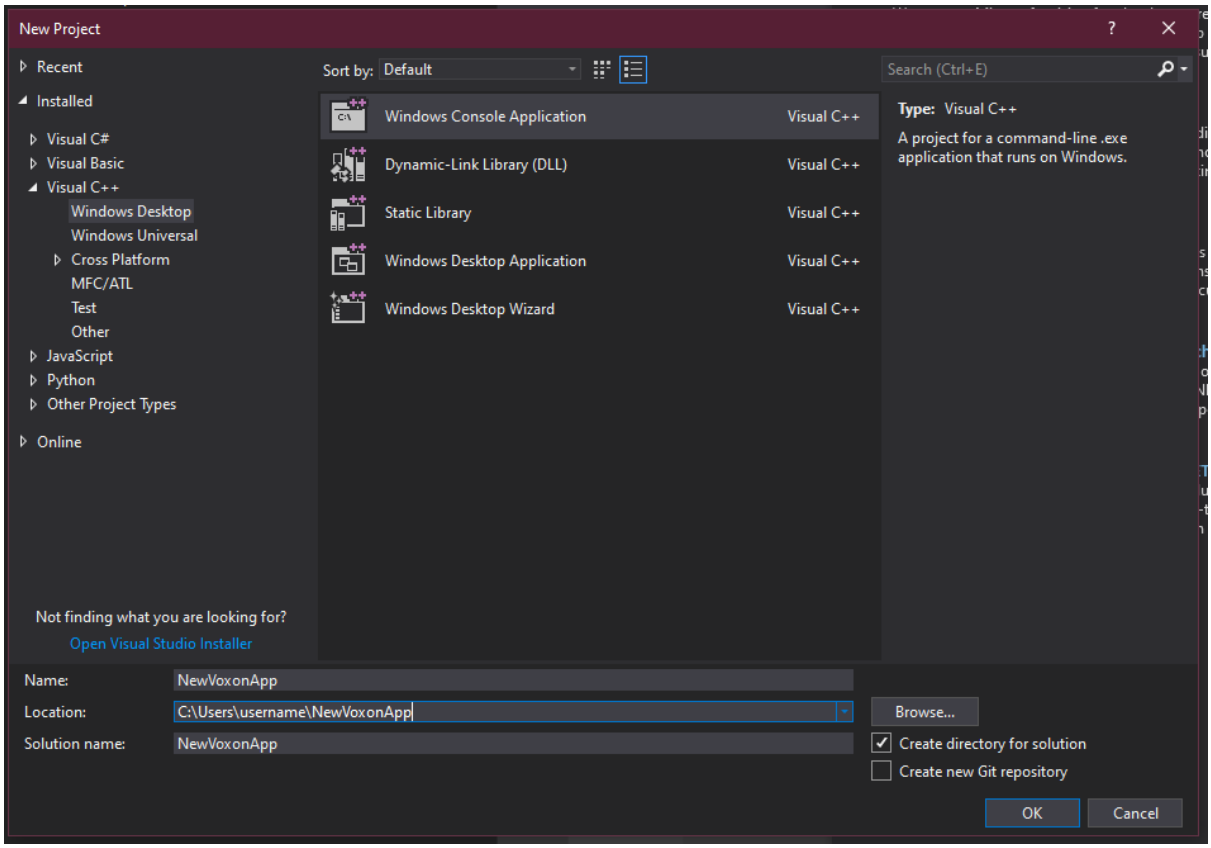

Figure 14 Creating a Visual C++ Project in Visual Studio

In *Visual Studio* create a new project and select a Visual C++ template. (If Visual C++ templates are not available you need to run the *Visual Studio* Installer and install the C++ development workload.)

You can name the project whatever you like. But keep in mind it can be difficult to change later. A 'Windows Console Application", "Console App", or "Empty Project" are all fine. *Visual Studio* slightly changes the names with every release.

2.3.2 Set the target platform to a 64-bit application.

By default, *Visual Studio* wants to build a x86 (32 bit application). Change the target platform to x64 (64 bit application) as the Voxon API can only build 64 bit applications.

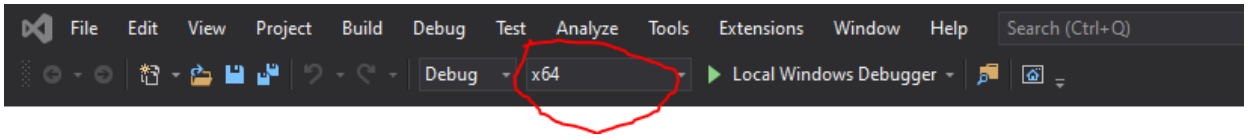

Figure 15 Ensure your target platform is x64

2.3.3 Create or replace the main source file with the VX++ Hello World text

Depending on which template you started from. Either create a new *main.cpp* file or edit the existing main source file.

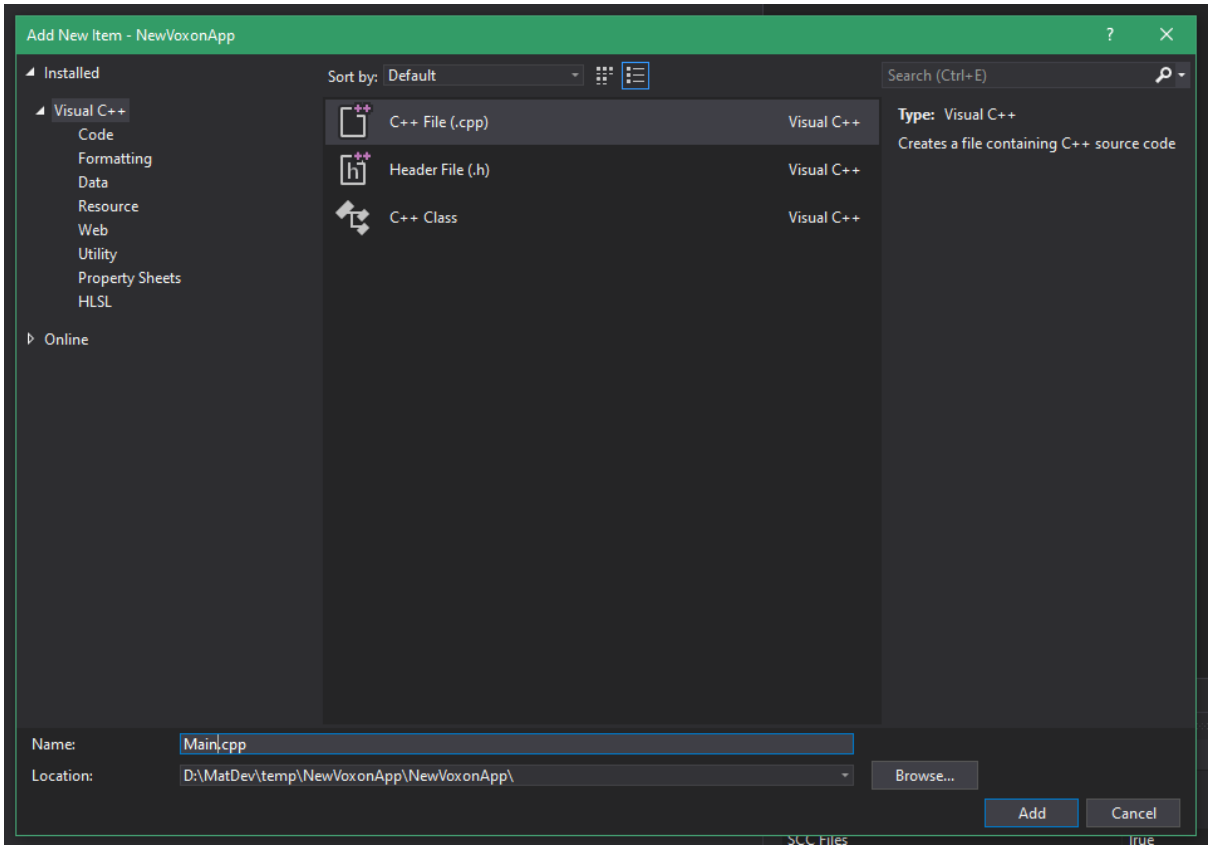

Figure 16 If no source files exist in your project make a new Main.cpp

Paste in the VX Hello World Example App as a starting point.

```
// VX APP C HELLO WORLD EXAMPLE
#include "voxiebox.h"

#define PI 3.14159265358979323
#include <math.h>

int main ()
{
	voxie_wind_t vw;                                                    // The voxie window struct to manage the volumetric display - most display settings exist in this struct
	voxie_frame_t vf;                                                   // struct to manage the voxie frame all graphical data is loaded into the voxie_frame_t struct
	voxie_inputs_t in;                                                  // mouse input struct
	double tim = 0.0, otim, dtim, avgdtim = 0.0;     // various timers
	voxie_xbox_t vx[4];                                                 // USB game controller structs
	int ovxbut[4], vxnplays;                                        // old USB game controller struct and number of USB controllers detected
	int i;
	// reusable variable
```

```
	if (voxie_load(&vw) < 0) //Load voxiebox.dll and get settings from
voxiebox.ini. May override settings in vw structure here if desired.
	{
		MessageBox(0, "Error: can't load voxiebox.dll", "", MB_OK); // if
there is an error this will create a standard Windows message box
		return (-1);
	}
	if (voxie_init(&vw) < 0) // first initalise of the (&vw) voxie_wind_t
activates settings from voxiebox.ini and voxie_menu_0.ini
	{
		return (-1);
	}

	// anything before the voxie_breath while loop is called once
	// use this space to setup your program
	// Hello world text variables - safe to remove these variables
	point3d textPos = { -.5,0,0 };
	point3d textWidth = { .1, 0,0 };
	point3d textHeight = { 0, .15, 0 };
	int textcolor = 0xffffff;

	while (!voxie_breath(&in)) // a breath is a complete volume sweep. a whole
volume is rendered in a single breath
	{
		/**********INPUT & TIMERS UPDATE**********/

		// update the timers. tim is current runtime and dtim is delta time.
Presented in seconds.
		otim = tim;
		tim = voxie_klock();
		dtim = tim - otim;

		// check through the USB game controllers
		for (vxnplays = 0; vxnplays < 4; vxnplays++)
		{
			ovxbut[vxnplays] = vx[vxnplays].but; // replace the 'old voxie
xbox button' values with the new ones..
			if (!voxie_xbox_read(vxnplays, &vx[vxnplays]))
			break;
		}

		// check for a few keyboard presses...
		if (voxie_keystat(0x1)) // esc key closes ap
		{
			voxie_quitloop(); // quitloop() is used to exit the main loop
of the program
		}

		i = (voxie_keystat(0x1b) & 1) - (voxie_keystat(0x1a) & 1); // keys '['
and ']'
		if (i)
		{
			if (voxie_keystat(0x2a) | voxie_keystat(0x36))
				vw.emuvang = min(max(vw.emuvang + (float)i * dtim * 2.0,
-PI * .5), 0.1268); //Shift+[,]
			else if (voxie_keystat(0x1d) | voxie_keystat(0x9d))
				vw.emudist = max(vw.emudist - (float)i * dtim * 2048.0,
400.0); //Ctrl+[,]
			else
				vw.emuhang += (float)i * dtim * 2.0; //[,]
			voxie_init(&vw);
		}
```

```
            /**********DRAW***********/

            voxie_frame_start(&vf); // start the voxie frame all graphical calls
must be between frame_start and frame_end
            voxie_setview(&vf, -vw.aspx, -vw.aspy, -vw.aspz, vw.aspx, vw.aspy,
vw.aspz); // set the view - the 'camera' position to capture volumetric content
(usually the confines of the aspect ration set in the voxie_window

            textPos.z = cos(tim) / 5; // move the text's Z position over time
using cos (a moving up and down effect)
            voxie_printalph(&vf, &textPos, &textWidth, &textHeight, textcolor,
(char*)"Hello World"); // actually draw / render the text

            //draw wireframe box around the edge of the screen
            voxie_drawbox(&vf, -vw.aspx + 1e-3, -vw.aspy + 1e-3, -vw.aspz,
+vw.aspx - 1e-3, +vw.aspy - 1e-3, +vw.aspz, 1, 0xffffff);

            //display VPS
            avgdtim += (dtim - avgdtim) * .1;
            voxie_debug_print6x8_(30, 68, 0xffc080, -1, "VPS %5.1f", 1.0 /
avgdtim);

            voxie_frame_end(); // end of the voxie frame
            voxie_getvw(&vw); // update the local voxie_window data with the one
being used by the DLL
        }

        voxie_uninit(0); //Close window and unload voxiebox.dll
        return (0);
}
```

2.3.4. Set up references (or add local copies) of VX Runtime development files.

You may notice that there will be a red zigzag line under the #include *voxiebox.h* line. This is because the *Visual Studio* project doesn't know how to find the *voxiebox.h* file. This is the main header needed to make VX applications.

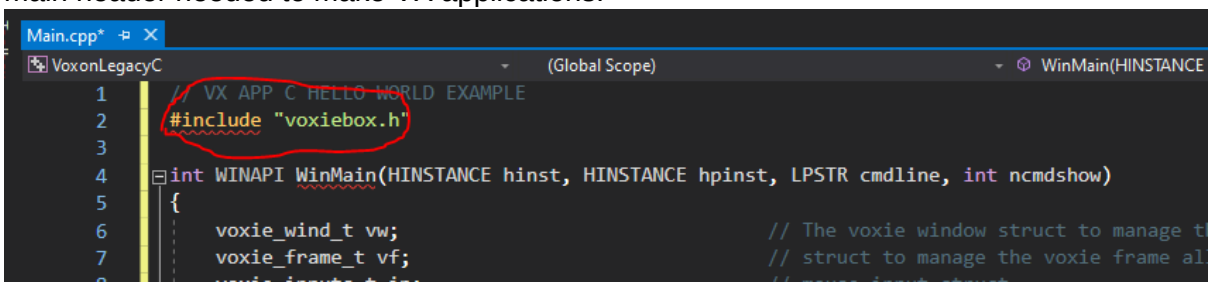

Figure 17 An Additional include directory needs to be set in the project's settings otherwise Visual Studio can't find the VX development files (voxiebox.h and voxiebox.dll)

For a VX application development you will need to set up the paths to the following files.

voxiebox.h	The header file for voxiebox.dll

You can either copy or add these files into the project, or you can set up an additional include directory in the project's settings. To edit your project's settings right click on the project in the solution explorer and select 'Properties'.

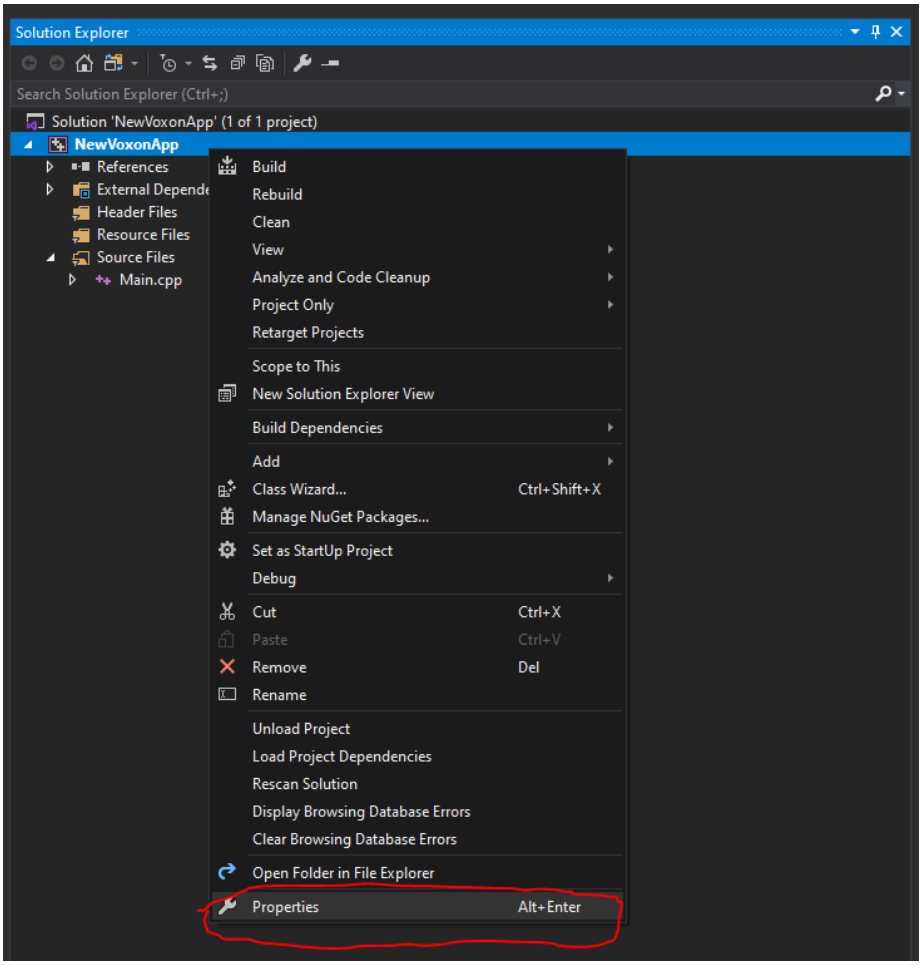

Figure 18 Right click on the Project and select 'Properties'

Look under the C/C++ tab and add a directory pointing to the *voxiebox.h* and *voxiebox.dll* and header files located in the **'Headers and DLLs'** folder. The file is located in the Developer's Kit path:

'...\Developers Kit\App Development\C Voxon SDK Development'

This can be an absolute path or relative.

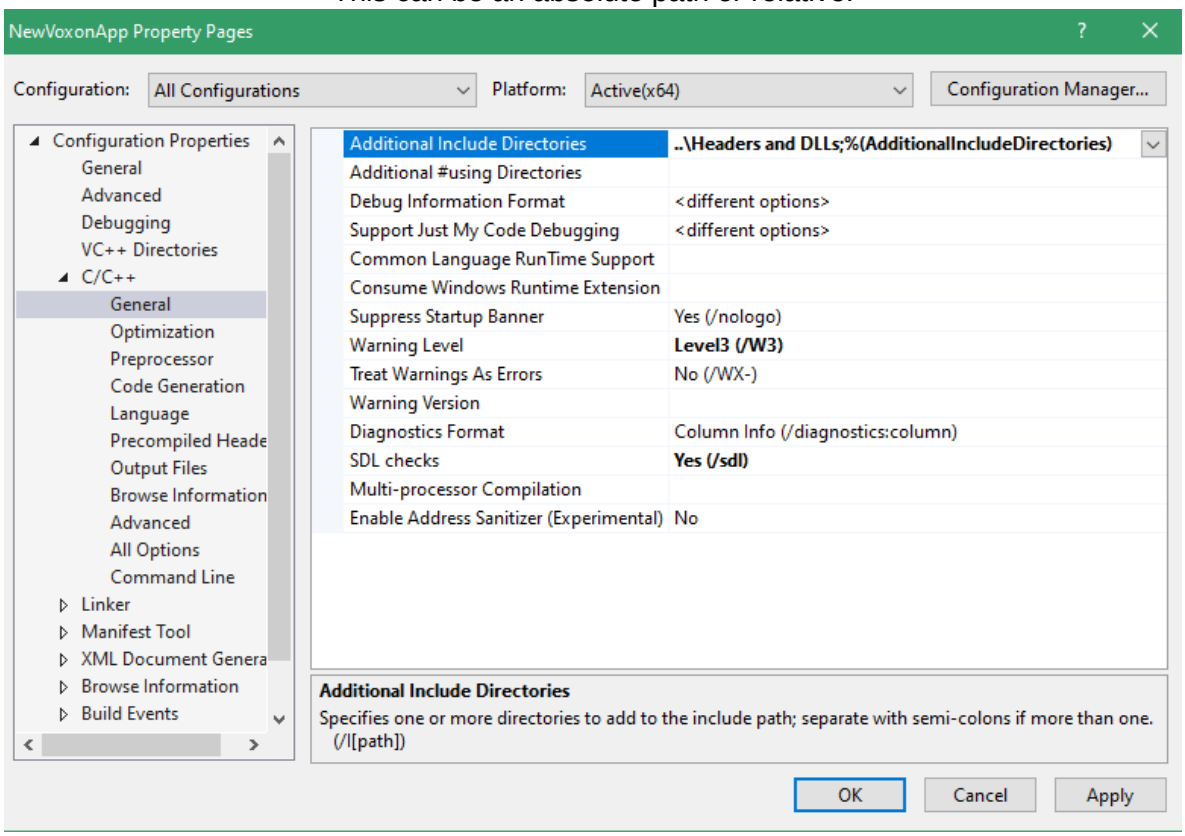

Figure 19 Make sure the voxiebox.h development file are included in your VS project

Copying the development files into the project would make the project more portable but you would have to copy the development files in again when an update is released.

Having setup the additional Include directories makes it easier to update multiple projects at once (if they all are set to the same include path).

Note: Make sure when you make this adjustment 'Configuration:' in the top left corner is set to 'All Configurations' or it has been setup for both Debug and Release

2.3.5 In Project Settings set Character Set to 'Use Multi-Byte Character Set'

By default *Visual Studio* sets the 'Character Set' to be 'Use Unicode Character Set' this will cause issues for Voxon Development and needs to be changed to "Use Multi-Byte Character Set'

In *Visual Studio 2017* this option can be found under the 'General' tab. In *Visual Studio 2019* this option can be found under the 'Advanced' tab.

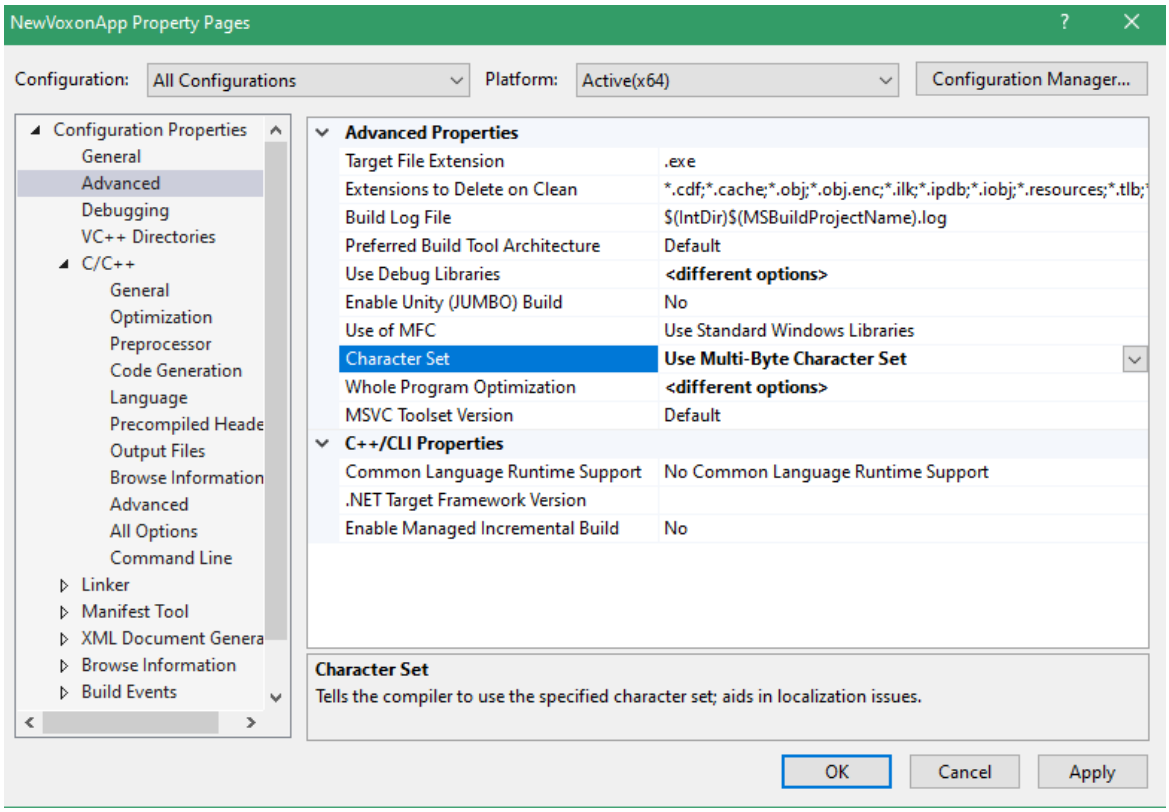

Figure 20 Voxon Development requires Character Set to be set to 'Use Multi-Byte Character Set'

2.3.6. Change the Runtime library to “Multi-threaded (/MT) to make the app more portable.

By default, programs compiled with *Visual Studio* will require VS runtime files on the end user’s system. (*MSVP140.dll* etc) you can embed these files into your application so your users won’t require *Visual Studio Runtime* files to be installed.

To change this, go to the project's properties and under C/C++ Code Generation. Change the runtime library to Multi-threaded (/MT) for a Release configuration and Multi-threaded Debug (/MTd) for the Debug configuration.

This change results in a slightly larger ‘.exe’ file but the benefit is you won't have to worry about people having the *Visual Studio Runtime* files installed on their device.

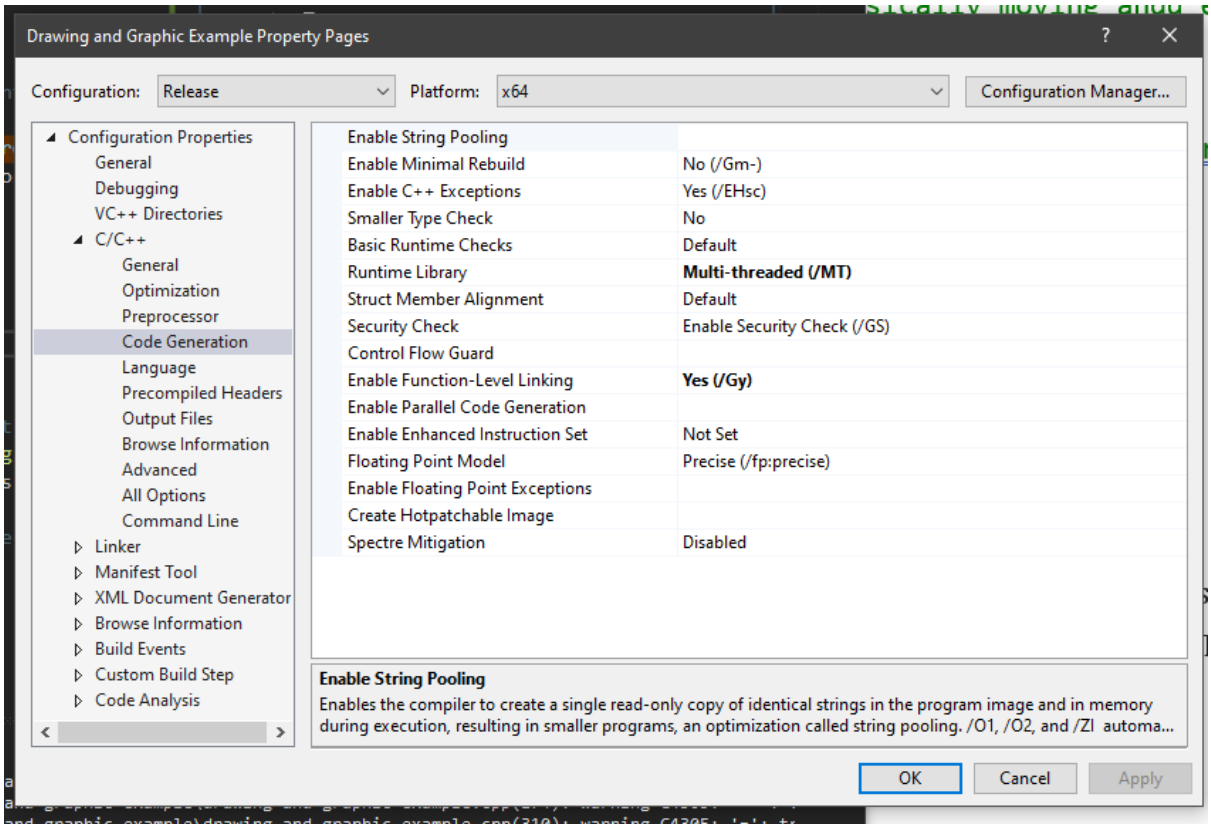

Figure 21 Set the 'Runtime Library' To Multi-threaded (or with debug) so end users won't need VS runtime installed.

2.3.7. Add _CRT_SECURE_NO_WARNINGS to the preprocessor definitions.

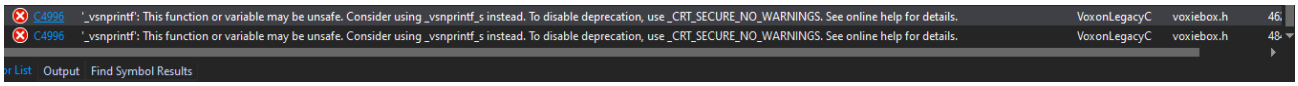

Figure 22 The error message you get when _CRT_SECURE_NO_WARNINGS is not defined

One of the *voxiebox.h* functions ‘voxie_printalph_()’ uses a ‘_vsprintf()’ function which Visual Studio flags as potentially unsafe. To get around this error (rather than editing the voxiebox.h file), edit the project’s properties and look under the **Preprocessor tab** nested under the C/C++ branch. Edit the Preprocessor Definitions **and add the text: “_CRT_SECURE_NO_WARNINGS”.**

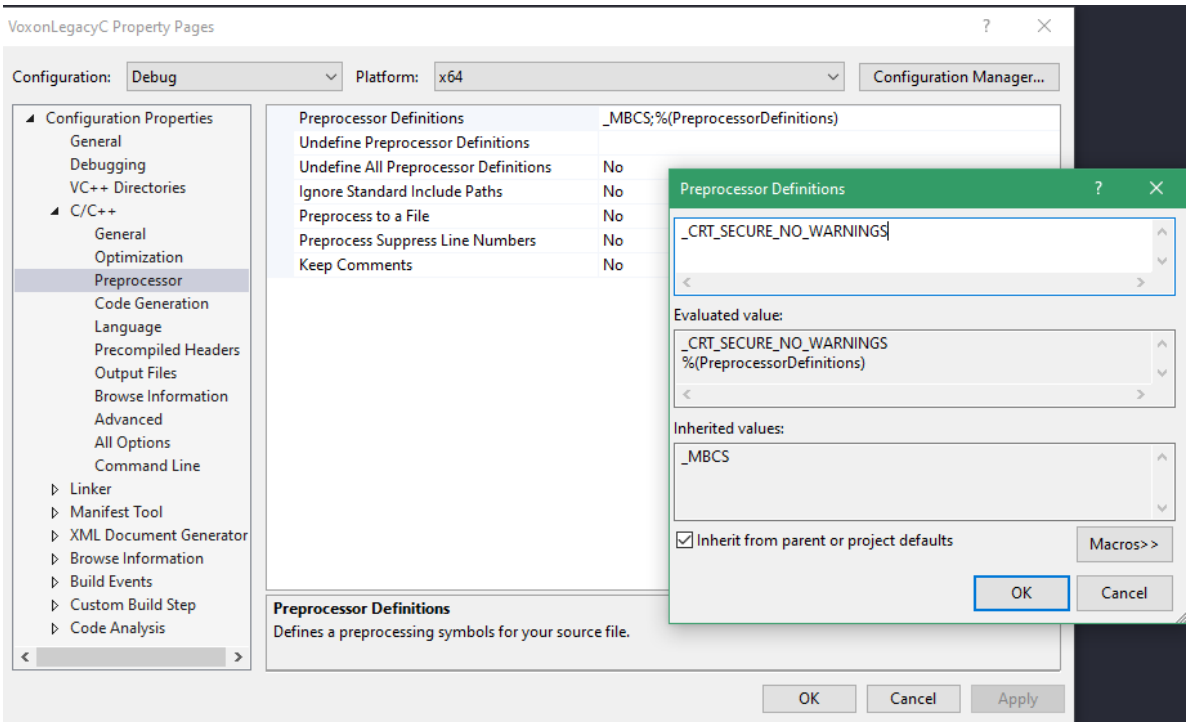

Figure 23 Adding the _CRT_SECURE_NO_WARNINGS definition is needed when making a VX application.

2.3.8. Test that your App can be compiled.

Your VX App is ready to be launched. Click the Green arrow to run the application in the local debugger.

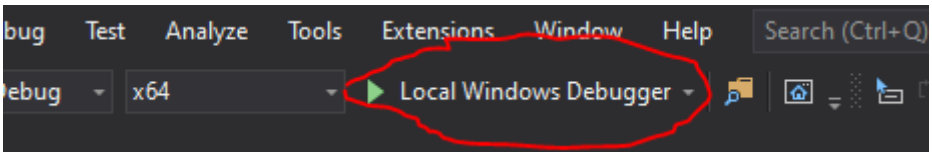

Figure 24 The green arrow launches your application in the debugger.

The VX Hello World application should if everything is successful it should look like this:

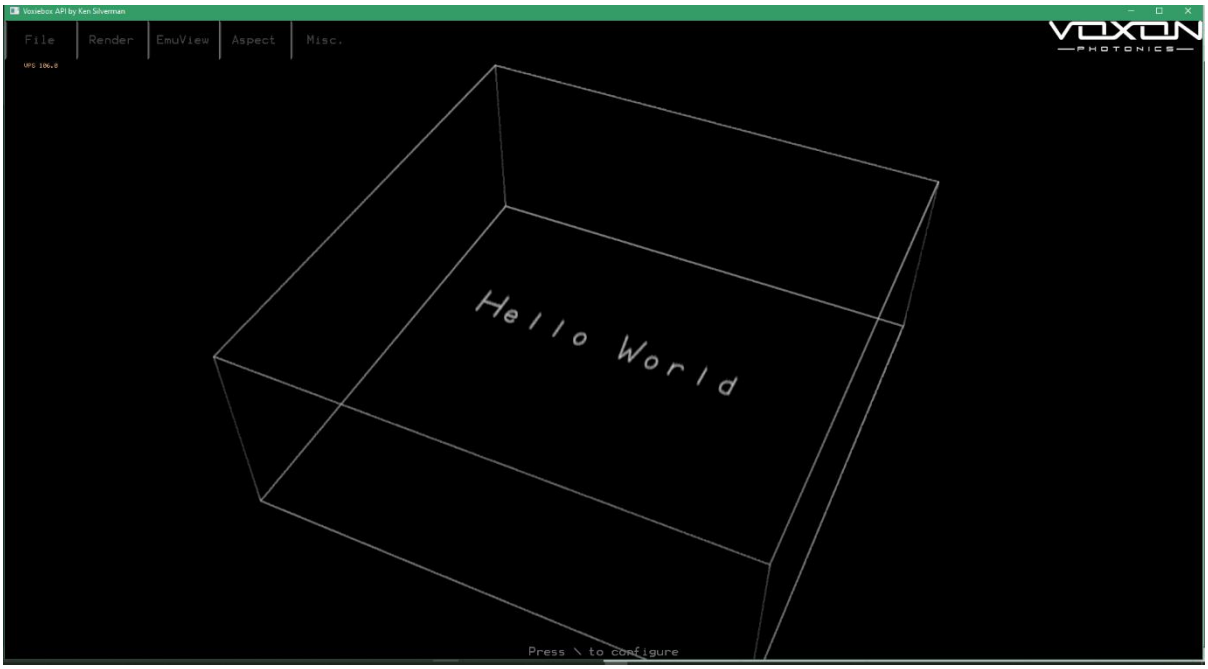

Figure 25 A successful VX Hello World Application

Any applications built using *voxiebox.h* will require access to *voxiebox.dll*. You can either distribute your app with *voxiebox.dll* in the local directory or assume the end user will by running the app on Voxon hardware (or have the Voxon Developers Kit installed on their system.

For function and reference documentation check out the *Voxiebox.txt* documentation which is located at '...\App Development\C Voxon SDK Development\Documentation'