

Voxon App Development Overview

The purpose of this document is to outline what a Voxon volumetric application (VX app) is, how it is developed, and some of the core concepts behind developing a Voxon application. The target audience for this document is software developers and general users of the Voxon hardware who want some insight into what the system is doing when a Voxon application is launched.

What is a Voxon Application?

A Voxon application is a volumetric software application developed with Voxon runtime libraries (always using *voxiebox.dll* but in some instances additional files). A VX app is intended for use with Voxon's volumetric display technology. While VX applications are intended to be used on Voxon's volumetric hardware, all Voxon VX apps can run on a Windows machine using the software simulator / emulator built into the *voxiebox.dll*.

A VX app is a standard Windows based executable file. All VX apps require access to the *voxiebox.dll* to run. Upon execution, If hardware is detected the application will render onto the volumetric display, if no hardware is found the application will execute into simulator / emulator mode.

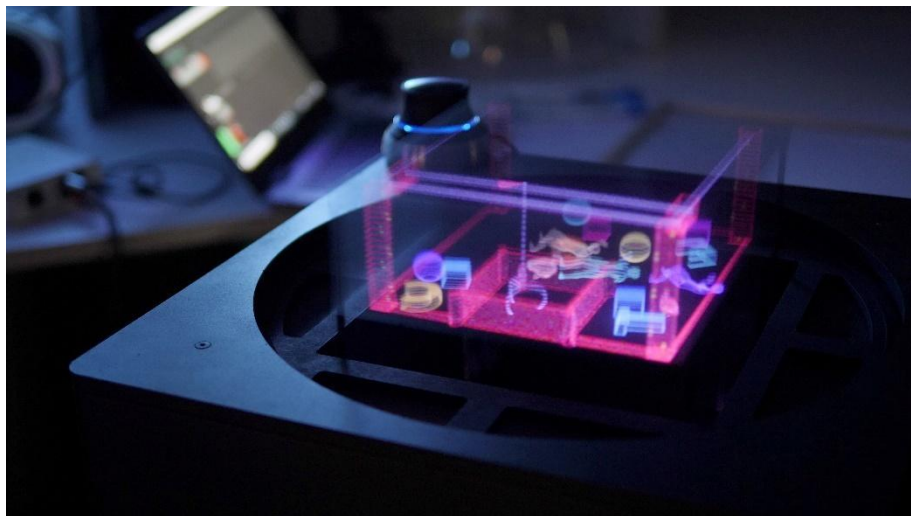


Figure 1 Voxon Claw Game - running on Voxon hardware.



Figure 2 Voxon Claw Game - running on Windows 10 PC via the Voxon hardware simulator

How VX apps are developed

VX applications are developed on 64-bit Windows based machines using the Voxon Developers Kit (obtainable from our website at www.voxon.co).

The Voxon Developers Kit contains the development files required to create VX executables. All VX applications are derived from the *voxiebox.dll* library which is written in C. The *voxiebox.dll* could be wrapped to work with other computing languages.

At the current time, Voxon officially supports three methods of creating VX applications:

- 1) Writing a VX application in C directly using *voxiebox.dll* and *voxiebox.h* files. Programs can be easily compiled with a Visual Studio command prompt or GCC. (GNU.)
- 2) The 'VX++ Framework' is a C++ wrapper and framework, which interfaces with the *voxiebox.dll* via the *VoxieBox* class. This framework includes various quality-of-life changes and helper functions to assist with development. It is also made to integrate with Visual Studio easily.
- 3) Using *Unity* (a popular 3D engine) using the Voxon official plugin. Unity's graphic engine and 'component based' scripting makes it an attractive and unique entry point for VX app development.

Having access to Voxon hardware is not needed for VX app development as the simulator is extremely accurate in rendering how our volumetric display looks and how the hardware behaves on a traditional 2D screen.

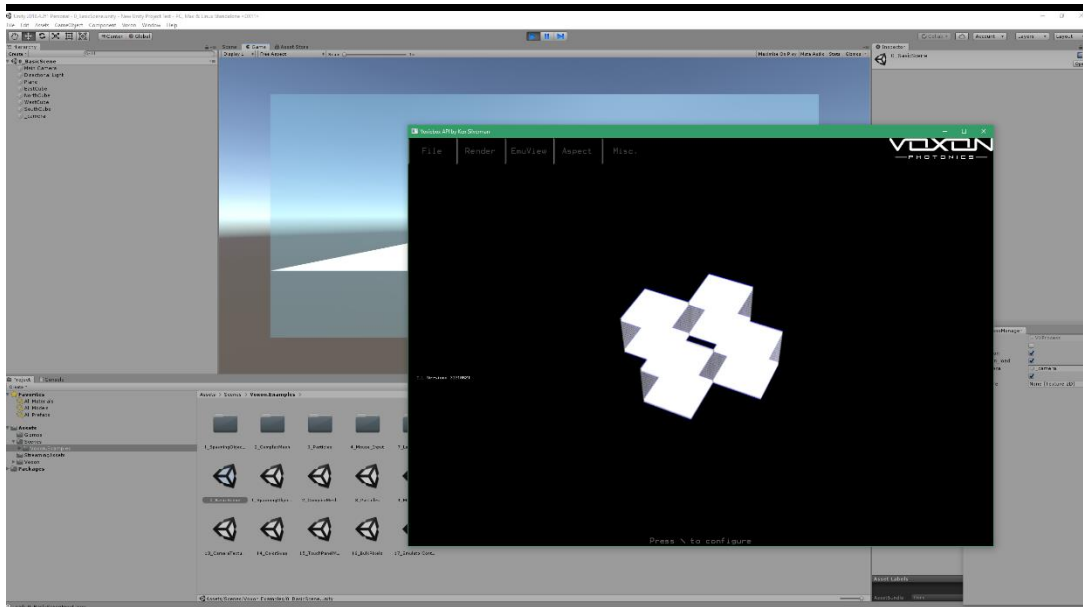


Figure 3 a VX app being developed with Unity on a Windows PC

Using a VX app

Volumetric applications offer a full 360 degree viewing experience. The display is also fully transparent. These two factors are wildly different from any other digital medium and come into play when designing your app.

VX apps can also make use of a secondary touch display located on the front of the Voxon volumetric unit. This display is used to change settings, navigate menus and it can also be used as an additional interactive screen for your app.

The 3D Space Mouse can be a great tool in helping users navigate VX applications in combination with more traditional input devices such as keyboards, mice, and game controllers. Since the VX1 system and runtime are Windows based, it is easy to develop and experiment with different types of input devices.

VX app development is new, exciting, but also challenging!

How a VX app works

A VX app works by using the Voxon runtime library. At the heart of the library is the *voxiebox.dll*. Other libraries may be needed depending on your app. (VX++ apps require *VxCpp.dll*, Unity developed apps require the C# bridge files.)

Despite the required files, all VX apps have a similar core structure and require certain function calls defined in *voxiebox.dll*.

Observe the life cycle of a VX app:

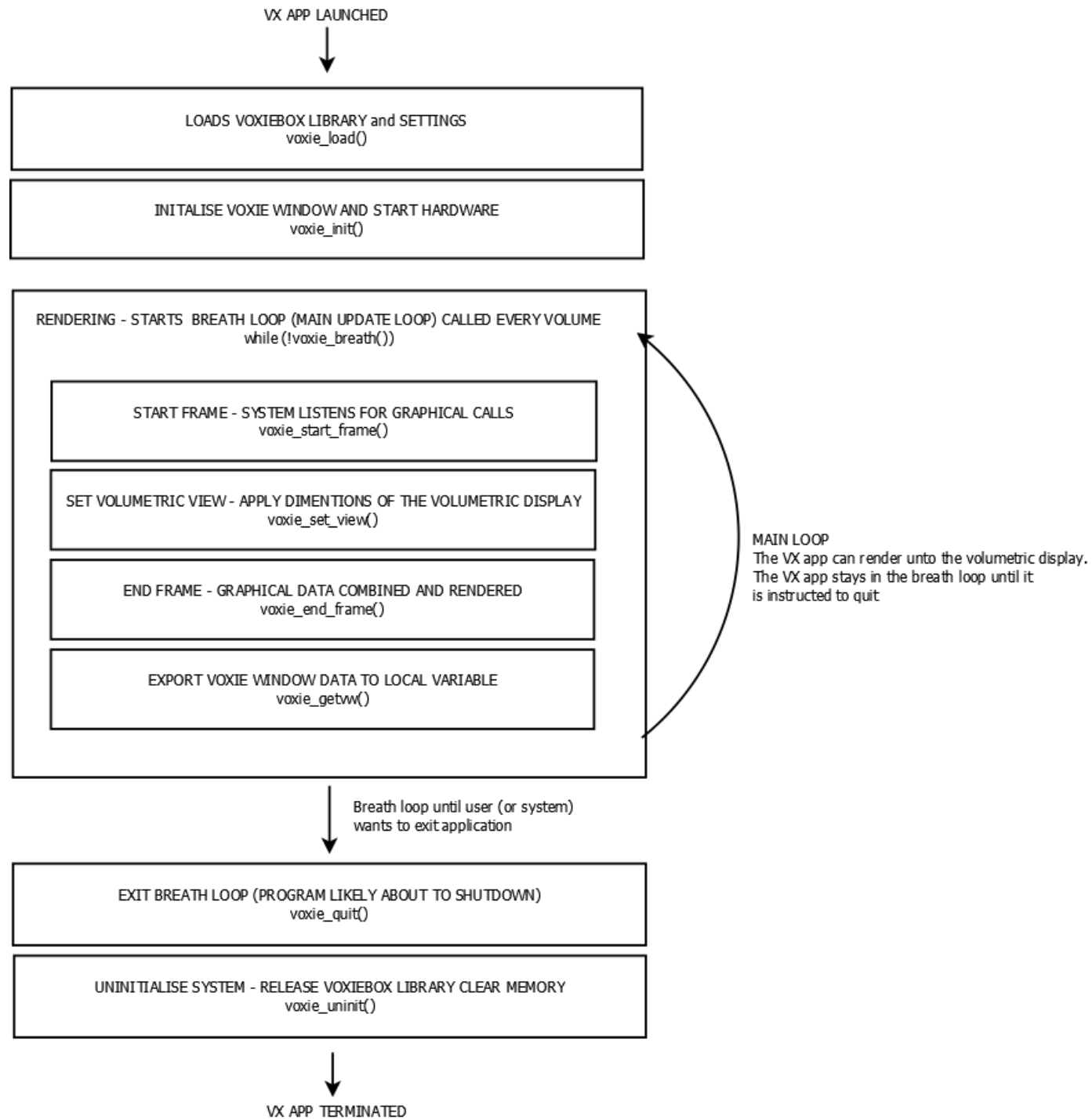


Figure 4 Diagram explaining the lifecycle of a VX app

01. **voxie_load() is called** - Loads in the *voxiebox.dll* into memory and settings from *voxiebox.ini* and *voxie_menu_0.ini*.
02. **voxie_init() is called** – imports and sets the *voxie_wind_t* struct ('voxie window'), which holds all the settings for the volumetric display and starts the hardware (if applicable)
03. **voxie_breath() loops starts** – Main update loop for the program. Called every volumetric sweep. A VX application spends most of its runtime within the 'breath loop'
04. **voxie_start_frame() called within breath loop** - readies the *voxie_frame_t* struct to listen for graphical calls.
05. **voxie_set_view() called within the breath loop** - the dimensions (X,Y,Z) of the volumetric display are set.
06. **voxie_end_frame() called within the breath loop** - all volumetric graphical calls are interpreted and prepared. The *voxiebox.dll* renders the summed graphical data to the display.
07. **voxie_getvw() called within the breath loop** – exports the Voxie Window data in memory to the local 'vw' variable. Used to pass window data to and from the *voxiebox.dll*
08. **voxie_quit() called once within the breath loop to exit loop** - used to quit out of the breath loop when the application is terminating.
09. **voxie_uninit() is called** - closes the *voxiebox.dll* handle and frees *voxiebox.dll* from memory.

Understanding of how these functions work can be seen in the many example programs or the coding reference manuals provided.

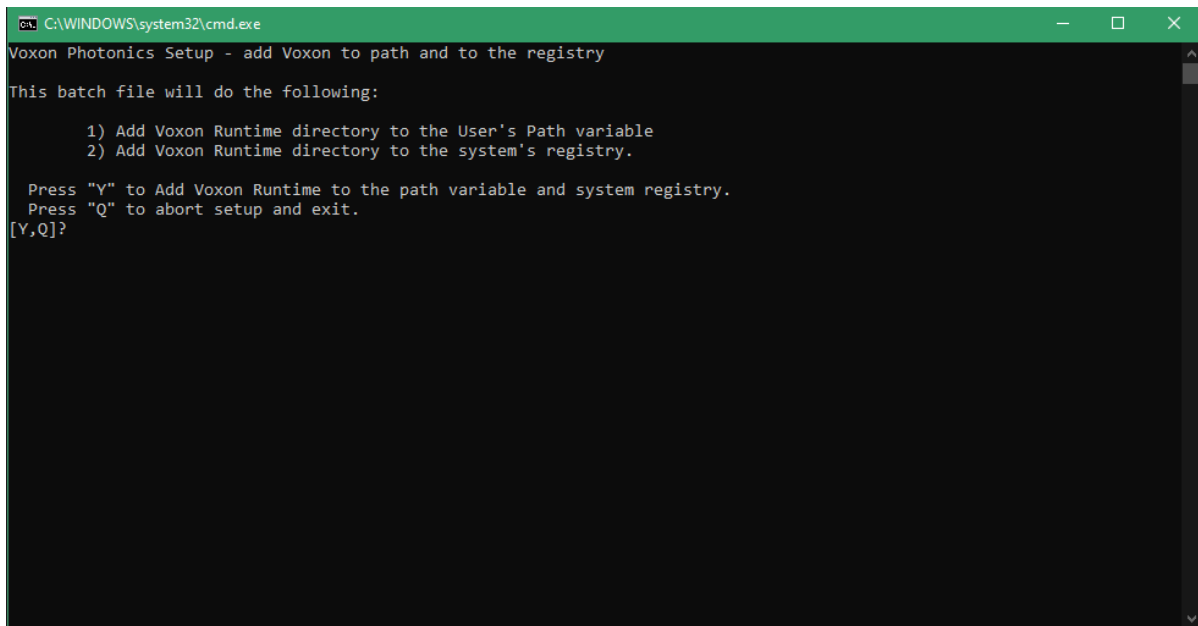
The Voxon Runtime

By default, the Voxon Runtime files are located at 'C:\Voxon\System\Runtime'. These are the files needed to run the Voxon environment.

| | |
|--------------------------------|---|
| <i>C#-bridge-interface.dll</i> | The interface library for the C# bridge – used by the <i>Unity plugin</i> |
| <i>C#-Runtime.dll</i> | C# bridge which works which wraps the <i>voxiebox.dll</i> |
| <i>grab.flac</i> | Sound file when using the Voxie Menu |
| <i>typeKey.flac</i> | Sound file when using the Voxie Menu |
| <i>voxiebox.dll</i> | The core library. Essential to running any VX software |
| <i>voxiebox.ini</i> | <i>voxiebox.dll</i> 's System settings configuration file – human editable |
| <i>voxiebox_menu_0.ini</i> | <i>voxiebox.dll</i> 's User settings configuration |
| <i>voxiebox_menu_1.ini</i> | Backup of default settings for a VX1 with Keynote projector |
| <i>voxiebox_menu_2.ini</i> | Backup of default settings for a VX1 with Anhua projector |
| <i>runtimeinfo.json</i> | Human readable file which contains the version numbers of files. Used by Vertex |
| <i>voxiebox.log</i> | Log file amended to whenever the <i>voxiebox.dll</i> is accessed |
| <i>VxCpp.dll</i> | Library for VX apps using the VX++ framework. |
| <i>voxonRuntime.txt</i> | Text file giving a summary of the runtime, it files and reference of .ini file settings |

On Voxon hardware systems (and recommended on Developers Kit installs), a user environment path variable is set up to give path access to the runtime files. For the *Unity plugin* to correctly work some registry keys need to be created. To set up your path or registry, run the batch file 'Set Path and Registry.bat' located under '..\Voxon\System' which will set these settings for you.

Note: The path and registry values are set when you run the Voxon Installer.



```
C:\WINDOWS\system32\cmd.exe
Voxon Photonics Setup - add Voxon to path and to the registry

This batch file will do the following:

    1) Add Voxon Runtime directory to the User's Path variable
    2) Add Voxon Runtime directory to the system's registry.

Press "Y" to Add Voxon Runtime to the path variable and system registry.
Press "Q" to abort setup and exit.
[Y,Q]?
```

Figure 5 The 'Set Path and Registry.bat' file can set the path and registry values for easy access to the Voxon Runtime files.

The Core Elements

voxiebox.ini

The *voxiebox.ini* is a human readable / editable configuration file. The *voxiebox.ini* contains system and hardware settings which define the voxie window and hardware setup. In general, the settings won't need to be reconfigured unless there is a system or hardware change.

voxiebox_menu_0.ini

The *voxie_menu_0.ini* file is a human readable / editable configuration file. It contains most of the user settings. An interesting setting to note is that the keystone configuration is stored in this file. If *voxiebox_menu_0.ini* is absent from the runtime nothing will display on the hardware when a VX app is launched. When 'Save Settings' is pressed from the 'File' tab in the voxie menu a new *voxie_menu_0.ini* is written with the new settings.

Keystone Settings & Keystone Calibration

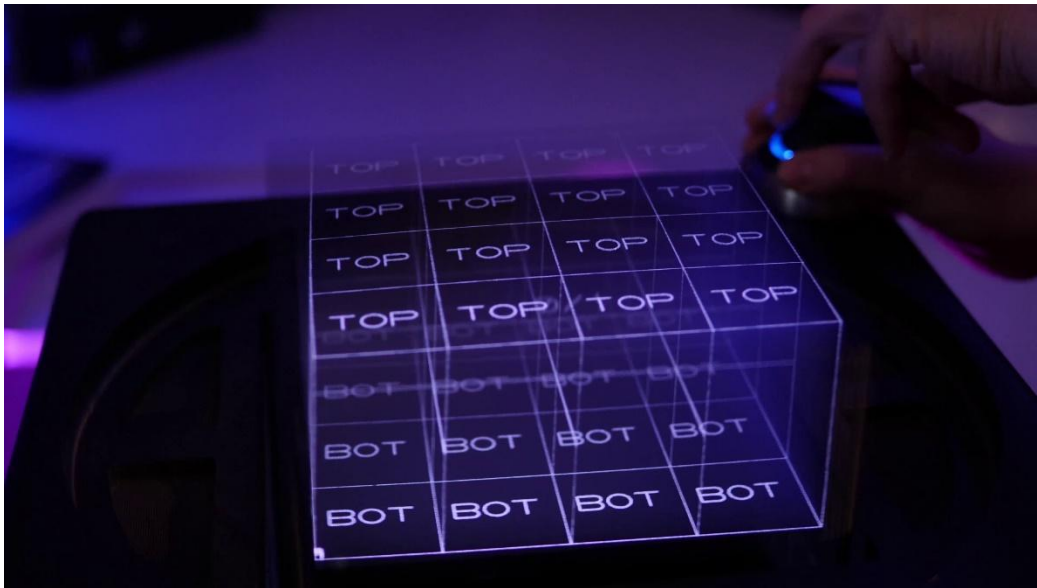


Figure 6 the keystone being calibration on a VX12

The VX1 achieves its three-dimensional imagery by projecting a stack of two-dimensional image frames onto a reciprocating screen. As the screen physically moves further away from the projector, the two-dimensional image will spread out. This technical problem is solved by using a software 'keystone' calibration. The keystone can be adjusted by manually adjusting the vertices of the cubic volume using a mouse or space navigator. This allows the software to scale the voxels within the volume proportionally and prevents distortion.

This process is called 'keystone calibration' and can be conducted on your VX1 by choosing 'keystone calibration' located under the 'aspect' tab in Vertex's Settings panel. Alternatively, it can be run by loading *Voxiedemo* and choosing 'Keystone Cal'.

The Voxel Window

The voxel window (*voxie_wind_t*) is a data struct that contains variables associated with managing the volumetric display.

Many of the settings for the initial voxel window are defined by the two voxelbox related *.ini* files. During execution a VX application can also change the settings using the 'voxie_init()' function.

A system without a voxel window settings wouldn't know where to render the volumetric image or how the hardware is to behave.

The Voxel Frame

The voxel frame (*voxie_frame_t*) is a data struct that contains all the graphical data for the volumetric display (and secondary touch screen display). The frame buffer contains the x,y,z positions for each voxel to render on the volumetric display. It also prepares graphical output for the secondary (touch) screen. All graphical draw calls between 'voxie_start_frame()' and 'voxie_end_frame()' are sent to the *voxie_frame_t* struct to be prepared for rasterization.

Loading the DLL and Initialisation

A VX app needs to load in the *voxiebox.dll* to make use of its library. This is done by using the 'voxie_load()' function. This creates a handle for the DLL and puts it into memory. The function 'voxie_init()' initialises the window. The first time this function is called it initialises the voxie window and starts the hardware. Any subsequent calls updates the voxie window.

The Breath Loop

A 'breath' encapsulates a single graphical sweep of the volume, similar to a single frame of a traditional display. The 'breath loop' is thus the main loop that VX apps cycle through during runtime. A while loop is used to create the voxie_breath function ('while (!voxie_breath())') and will return 0 if the system hasn't been instructed to exit the loop (done so by the 'voxie_quit()' function).

The Set View function

The 'voxie_set_view()' function sets the coordinates of the volume to render. It can be called any number of times during runtime. Typically you'll want this function to match the aspect ratio set out in the voxie menu (vw.aspx, vw.aspy, vw.aspz).

The Start and End Frame functions

These two functions prepare the voxel buffer which is to be rendered by the volumetric display. The 'voxie_frame_start()' function starts, the voxel buffer and any graphical calls after it will be loaded into the voxel buffer. When 'voxie_frame_end()' is called, the buffer is processed and sent to the display for rendering.

Drawing Graphics

The *voxiebox.dll* supports many types of graphical draw calls. 3D model file types such as '.obj', '.stl', '.kv6', and '.ply' can be rendered using the 'voxie_drawspr()' function. Raw vertex data (and textures) can be rendered using the 'voxie_drawmeshtex()' or 'voxie_drawpol()' function.

Heightmaps (2D images with a heightmap stored alpha channel) are supported using the 'voxie_drawheimap()' function.

Simple shapes, single voxels, and a line-based text (font) format are also supported and have their own relevant function names.

Managing Input and other non-graphical updates

The *voxie_inputs_t* struct manages mouse inputs and is updated when 'voxie_breath()' is called. The functions 'voxie_keystat()' and 'voxie_keyread()' manage keyboard presses. Note the scancodes are a little different, use Ken's *KeyView.exe* ('...\Voxon\Developers Kit\App Development\Development Tools\') application to get the required scan code or see the VX++ framework reference manual. Space Nav and USB game controllers use *voxie_nav_t* and *voxie_xbox_t* structs to manage their various states. See the coding examples provided to further understand the use of these functions.

Ending the application and correct shutdown procedure

Only one instance of the *voxiebox.dll* can be running as it controls the Voxon hardware. It is very important that a VX application terminates appropriately, otherwise it can lose sync with the Voxon hardware and cause malfunctions. For a VX App to terminate properly, the 'voxie_quitloop()' function is called during the 'voxie_breath()' loop. This ends the loop and informs *voxiebox.dll* that the process has ended. 'voxie_uninit(0)' is then called which closes the *voxiebox.dll* handle and frees *voxiebox.dll* from memory.

Changing System Settings During A VX App's Execution

Modification of the voxie window during the runtime of a VX app is possible. A VX app will have two versions of the *voxie_wind_t* struct. A local version, and a version which is being used in memory by the *voxiebox.dll*. The 'voxie_getvw()' function copies the DLL's voxie window state to the local instance.

Any changes you make to the local instance of the *voxie_wind_t* struct has to be passed onto the *voxie_wind_t* struct being used by the *voxiebox.dll*. This is done by using the 'voxie_init()' function.

For example, this code will change the color mode of the display to RGB interlaced mode.

```
vw.usecol = 1; // set the local voxie_wind_t usecol value to 1
voxie_init(&vw); // send the local voxie_wind_t to the DLL and update
settings
```

For more information about the attributes of the *voxie_wind_t* struct reference see the 'VX++ framework' manual which is located in the '...Voxon\Developers Kit\App Development\C++ (VxCpp) Development\Documentation' folder.