RHMG Audio-Video API Library

Software Developers Guide - MSCompact

RHMG Software Tools Library

11/16/2018

Authors: Bill and Scott Werba
Developer: Scott Werba

Document ID: 3000-000
Revision: v3.0.0
Publication Date: Nov 15, 2018
Publication Status: **Release**

1.0 Introduction – w/3D Scope Tools

Update: we have added a set of 3D scope functions to our standard scope library. See our *MultiscopeCompact* User Guide for the 3D scope operational descriptions. The 3D DLL programming details can be viewed in this document.

Our developers kit offers each instrument in standard Windows DLL form. As a result, our audio/video scope functions set can be easily integrated into your own video project, using standard software development IDE's and practices. We have used Visual Studio 2013 as our IDE.

The colour space conversion for each instrument follows industry SD and HD video standards ITU-R BT.601 for SD and ITU-R BT.709 for HD (depending on mode set). Originating colour space is defined as pure 24bit RGB in full gamut and is converted\translated to the limiting YUV or YCrCb gamut descriptions depending on the user setting for each instrument, with appropriate 8 bit scaling and IRE level limits.

We do not yet support BT-2020, 4K 10 bit video, but is in our plans.

Source data for each instrument is found from inputs of standard uncompressed RGB based bitmap files or from RGB values originating from a pixel data array in memory. RGB order must first be defined. Only pure RGB images are supported. Any other image format must be converted to BMP before they can be processed by our API functions. This and further details are found later in this document.

Our software library also comes with a number of other important ancillary DLL functions that allows rendered output to be displayed in OpenTK GLControl. This API tool set version of our release is *.Net Framework 3.5 and 2 based.

Multiple instances can also be incorporated if required, each with its own data source and process and GL Control for render, assuming each instance has been setup independently with its own variables.

The software requires Framework 2.0 and Framework 3.5 for these DLLs and your application to operate. The only hardware consideration is that there is a graphics card in your computer system that supports OpenGL v1.2 at a minimum. That's it.

Each software audio and video instrument in our software kit offers various setup and modes of operation that are described in this document. The C# sample code will show how to setup and code the DLL's into a typical application.

When rendering a 1920 x 1080 HD image for example, and accounting for all of the pixels and lines in the picture frame, the data size will be quite large and will slow down the computation and display process. So not all of the lines need be rendered for workable results. There is the ability to decimate the rendered image to allow a recovery in refresh speed. This can be done automatically (see LOD function) or manually. The quality of the output of a decimated display will only be evident if the decimation factor is set too high. Decimation factor is an integer value which affects the number of lines output vertically. Lines are skipped uniformly in factors of 2 and output for display. The more the decimation factor the less accurate the display and perhaps the more rough looking output. In return of course the display refresh speed increases, if you are looking for approximate expectations or signal trends.

DLL System Configuration
The DLL internal operation is to decode the input arguments, to act on them and to calculate the necessary data output to be displayed. They in of themselves have no display capabilities. That is done outside the DLL using a GLControl on a form. After the relevant input data has been processed, the resulting data is rendered internally to meet the selected instruments display output properties. The DLL will then supply an output map of the rendered data to a GLControl.

2.0 Installation of SDK into Visual Studio

1) Open the SDK .rar file to a directory of your choice. Upon extraction the directory will contain a number of DLLs, a readme file, an OpenTK license and a small sample project with source code.

2) In Visual Studio import the DLLs in the Solution explorer and add in the References; OpenTK, OpenTK.GLControl, QuickFont, and ScopeTools, those using 3D scopes will also need ScopeTools3D as a reference.

3) Add; using OpenTK, using OpenTK.GLControl, using ScopeTools and using BitmapHeader to the form that will be using the Scope Tools. If using 3D scopes you'll also be adding using ScopeGLTools3D as this is the namespace holding the 3D scopes and the 3D supporting an upgraded Modular Scope Display 3D

4) Declare a ModularScopeDisplay or a Scope_xxx Variable type

5) On initialization have the variable assigned to a new of its type with a ref of the GLControl on the form, and ScopeMode if it has one.

6) Setup the Variable's Init, SetupProjection and SelectConfig Functions with the proper data. And when ready to render, use the Render function to render

Important note:
You must include all of the DLLs, the readme file and the OpenTK license when you distribute the product.

3.0 Video Toolset Description

3.1 Waveform Monitor

The *Waveform Monitor* is an industry standard video calibration instrument used to measure the video luminance and chroma components in an image or a video signal. Video is represented as digital data.

Newer WFM technology does not use analog inputs anymore but use digital SDI type serial input or HDMI for SD and HD video display work. In *Figure xx* above, shows a typical luminance output of an RGB based image frame. The RGB values of the input file or array are converted to YCrCb where only the Y component is used for display purposes. Y calculations are based on either SD or HD selected colour spaces.

There is an optional chrominance overlay on top of the luminance white trace and adds a combined chrominance value where chrominance displayed is the square root of U squared plus V squared.

This option is off by default.

The equations used within this application are in *Appendix C*:

*SD Ref: en.wikipedia.org/wiki/ycbcr
*HD Ref: Charles Poyton Errata - Digital Video and HDTV Algorithms and Interfaces

The smallest window or picture box this function can be displayed in is 342 by 240 pixels, before any annotated text in the rendered display will disappear.

3.1 VectorScope

The Vector scope is a video tool used to see the hue and saturation levels of a color image source. Saturation is represented in vector length or magnitude and hue a vector angle. It is a great tool for indicating white balance of an input image for example, as shifts in hue are readily shown when offset from the white and black center of the scope.

The chrominance component of an image can be isolated when converting the RGB colour space to YCrCb and only using the CrCb component and calculating the necessary vector (angle and magnitude) from the colour data.

RGB data can be sourced from a file or from an array of data which is user selectable.

The user can also select the rendered result in direct colour association or in standard instrument green colour. In addition, the traces can be displayed as dots or lines. The default of these being colored dots.

The smallest window or picture box that can be programmed for this instrument is 280 by 280 pixels. Have the aspect ratio be as close to 1:1 as possible. Using a 4:3 AR is the next best standard to use if necessary.

The targets for the primary colors and their complements are to specification. The mini-targets are at 2.5 IRE and at 2.5 degrees – 100% saturation. The outer targets are at 20 IRE and 10 degrees. The layout rings are there to help the user with finding the value of the visual representation plotted relative to the center of the display which is white/black levels (255 and 0).

Flesh tone angle of 33 degrees is also part of the display face.

3.2 RGB Histogram

The Histogram is a simple statistical tool used to view the number of occurrences of a particular pixel value in an image. Pixel depth is from 0 to 255 (8 bit image systems) and noted in the x-axis and the number of occurrences for any one pixel is normalized in the Y axis.

The Histogram accepts 24-bit RGB bitmaps, and RGB, RGBA, BGR and BGRA data sourced pointers. The system can display the data only in Percent right now, and finds the most common and least common color value in the image.

The smallest window or picture box this tool can use is 350 by 160 pixels.

The numbers on the right side of each colour strip of the histogram display the value of the color component found that is the maximum at the top and the minimum found just below the maximum.

3.3 RGB Parade

The RGB parade is a tool used to see the individual RGB color values and intensities in an image.

The RGB parade accepts 24-bit RGB bitmap files, and RGB, RGBA, BGR and BGRA data pointers. The system can display the data in Percent, IRE and RGB. The user can set the tool to only show RGB (Tri-mode) or show luminance and RGB (Quad-Mode).

The smallest window or picture box this tool can use is 342 by 240 pixels.

We have added a Blended RGB mode which will combine all three channels in overlay mode to show channel differences – helpful to adjust image colour shifts.

3.4 YCrCb Parade

The YCrCb parade is a tool used to see the separate luminance and chrominance values in an image.

The parade accepts 24-bit RGB bitmaps, and RGB, RGBA, BGR and BGRA data pointers. The system can display the data in Percent, IRE and RGB.

The smallest window or picture box this tool can use is 342 by 240 pixels.

3.5 Audio 2Ch Stereo Oscilloscope

The oscilloscope is a 2 channel audio real-time instrument used see the waveform of incoming audio stream. The stream is buffered and prints the properties of the wave form found; ie: sample rate, time in seconds, bit depth, and the size of the buffer. The oscilloscope scope can render up to 4 channels of audio, each being a different color; green, red, blue, and cyan. The channels can be painted in either lines or dots – default, to display the waveform.

3.6 Audio PPM (Peak Program Meter – SMPTE Digital Model)

The PPM is an audio instrument used to show the audio peak levels in real-time of an audio stream. Full quantization of 16bits is used but only the upper 8 bits (48db worth) is metered. The PPM function will buffer a user set stream size and seek out the peak and display that instance. There is no averaging. The scale is in db and can process either a single or dual channel audio buffer.

3.7 Phase Meter

The Phase Meter is a tool used to see the phase difference between the left and right channels of a stereo audio signal. The meter traces can render in either dots (default) or lines. The trace color rendered shows something about its correlation between the channels. Green traces show a strong correlation (around 1); red traces show strong uncorrelated signals (around -1) and yellow traces covering anything between -1 and 1.

3.8 3D-Histo Vectorscope

The Histo-Vector scope represents color and hue data in its standard form along with the added feature of showing the hue/color distributions in 3D planes.

Rotational, pan and zoom controls are available for ease of viewing axis and data of interest. Plot traces are in the colors captured and processed. Planes are mapped in saturation, hue and chrominance distribution.

Access to all plot properties are available and shown in the applicable function in *Section 4.0*.

The histo-vector accepts 24-bit RGB bitmaps, and RGB, RGBA, BGR and BGRA data pointers.

The window or picture box for this scope can be scaled any size.

3.9 3D-Histo Waveform Monitor

The Histo-Waveform scope represents image luminance and color overlay data in its standard form along with the added feature of showing the sum total of its color of any point in 3D space.

Rotational, pan and zoom controls are available for ease of viewing axis and data of interest. Plot color traces are in the relevant data that was captured and processed. Planes are mapped in lumenance and chrominance distributions.

Access to all plot properties are available and shown in the applicable function in *Section 4.0*.

The histo-waveform scope accepts 24-bit RGB bitmaps, and RGB, RGBA, BGR and BGRA data pointers.

The display window or picture box for this scope can be scaled to any size.

3.10 3D-Histo RGB Parade

The Histo-RGB Parade scope represents image RGB overlay data in its standard 3 column form along with the added feature of showing the color distributions in 3D planes.

Rotational, pan and zoom controls are available for ease of viewing axis and data of interest. Plot color traces are in the relevant data that was captured and processed. Planes are mapped in lumenance and chrominance distributions.

Access to all plot properties are available and shown in the applicable function in *Section 4.0*.

The histo-RGB parade scope accepts 24-bit RGB bitmaps, and RGB, RGBA, BGR and BGRA data pointers.

The display window or picture box for this scope can be scaled to any size.

3.11 3D-Histo YCrCb

The Histo-YCrCb Parade scope represents image Y Cr and Cb data in its standard 3 column form along with the added feature of showing the color distributions in 3D planes.

Rotational, pan and zoom controls are available for ease of viewing axis and data of interest. Plot color traces are in the relevant data that was captured and processed. Planes are mapped in lumenance and chrominance distributions.

Access to all plot properties are available and shown in the applicable function in *Section 4.0*.

The histo-YCrCb parade scope accepts 24-bit RGB bitmaps, and RGB, RGBA, BGR and BGRA data pointers.

The display window or picture box for this scope can be scaled to any size.

3.12 RGB Cube - Demo

The RGB cube presents the full RGB gamut of a video frame. There is no color space restrictions in its display. There are 3 axis showing the prime, complementary colors and shades of white to black axis (gray scale). There is an internal outline showing the 91% or RGB 235 saturated value, as specified in 601 and 709 color space presentations along with a outline showing an RGB value of 16.

Rendering of pure RGB captured frames can be displayed in the cube. Full rotational, pan and zoom controls are available. No out of gamut indication is possible with is cube as there is no color space equation applied. RGB cube presentation is a very nice way of showing color distributions.

3.13 YCrCb or Color Gamut Cube

The gamut cube offers color distributions with color space restrictions applied. Out of gamut colors can be made clearly upon rotation and panning of the cube.

Data input is 24 bit RGB. Color space equations are applied to show the outer boundaries of allowed colors for specified video applications. YCrCb equations in the color space of interest determines the color boundaries. The Y value is luminance, the Z axis is Cr and the X axis is Cb with necessary offsets for proper display.

All prime and secondary colors are plotted along with the reference white to black axis

The scope renders uncompressed image data with it's color and color system selected and can either display the result with dots or lines. All vectors shown will be rendered in the color processed as it is being plotted.

4.0 DLL Description

4.1 Introduction

There are a series of distinct function calls in this DLL set. Most are mandatory and must be run in the order shown in *Section 4.4*. Some are optional. The optional calls are meant to be part of tight loop to provide best performance. Using the LOD component(s) in your code for example, will optimize the best decimation value for the images input and thus the highest rendering speed for the computer platform you are using. High performance computer platforms will offer high speed rendering and allow other multitasking operations without or minimal impact on the quality of the render.

4.2 Function Descriptions

Every instance of a scope requires an instance of a GLControl that it is to renders on.

The Contructor of any scope requires a reference to a GLControl that it'll render to, an optional HT_DataPub variable is for use when handling VBO data externally in an event based design. The ModularScopeDisplay and ModularScopeDisplay3D has a second variable which configures it to a specific scope type using an enum ScopeMode

An example of using ScopeBase:

```
sBase = new Scope_WFM(ref GL_Display);

sBase = new Scope_WFM(ref GL_Display, ref HT_DataPub);
```

An example of using ModularScopeDisplay:

```
MSD1 = new ModularScopeDisplay(ref GLDisplay1, ScopeBase.ScopeMode.WFM);

MSD1 = new ModularScopeDisplay(ref GLDisplay1, ref HT_DataPub,
ScopeBase.ScopeMode.WFM);
```

There are two required setup functions; `SetupProjection` and `Init`.

Init requires the size / resolution of the visual data or the SampleRate, number of channels, bits persample and the buffersize of the sound sample that it will be processing to accurately render the results of the information . *SetupProjection* requires the size of the GLControl. It will be rendering to make correct rendering aspect ratio and FOV (*Field of View*). If there is a change to either function, the scope needs to be updated with the new information or it will mis-render the data or worse, crash.

An example of using ScopeBase for a video scope

```
sBase.Init(1920,1080);
sBase.SetupProjection(GLDisplay1.Size.Width, GLDisplay1.Size.Height);
```

explain why there is two versions for the same thing, why use one over the other

An example of using ModularScopeDisplay for a video scope

```
MSD1.Scope.Init(1920,1080);
MSD1.Scope.SetupProjection(GLDisplay1.Size.Width, GLDisplay1.Size.Height);
```

An example of using ScopeBase for a audio scope

```
sBase.Init(sampleRate,channels,bitDepth,Buffersize);
sBase.SetupProjection(GLDisplay1.Size.Width, GLDisplay1.Size.Height);
```

or.....

An example of using ModularScopeDisplay for a audio scope

```
MSD1.scope.Init(sampleRate,channels,bitDepth,Buffersize);
MSD1.scope.SetupProjection(GLDisplay1.Width, GLDisplay1.Height);
```

Please note, the *Buffersize* chosen is critical for Audioscope operation as it will need to be the sample size of one of the channels. So if the HT_AudioBuffer is being supplied with a buffer of [256,2], the buffer should be 256. The buffer size will also have an impact on performance and quality of the render. I recommend an even number and or a power of 2 to ensure scope stability. A buffer size of 256 or 512 would work best.

The all scopes are configured using the *SelectConfig* function as this instructs the scope to be configured in a specific way when it's processing and rendering the result. The '*selection*' requires all 4 basic flags types* ORed together. Each instrument will require its own Config. However, certain scopes have extra flags that can be ORed in with the 4 basic flags types* shown below to give different ways of operating or rendering. See more detail on these flags in the Appendix A.

*The basic flags types are:

GL_DRM_xxx,	Data Read Mode	Where xxx can be BMP
GL_LAC_xxx,	Layout Control	Where xxx can be IRE
GL_LOD_xxx,	Level of Detail	Where xxx can be HI, MED or Low
GL_LUM_xxx	Colour Space	Where xxx can be YUV, HD, SD

An example of using ScopeBase

```
sBase.SelectConfig(GL_DRM_BMP|GL_LAC_IRE|GL_LOD_MED|GL_LUM_YUV);
```

An example of using ModularScopeDisplay

```
MSD1.Scope.SelectConfig(GL_DRM_BMP|GL_LAC_IRE|GL_LOD_MED|GL_LUM_YUV);
```

You can tell the scope to include in the rendering how fast it's going by feeding it a string containing the numerial FPS of how fast it's rendering.

An example of using ScopeBase

```
sBase.SetFPS(FPS.ToString());
```

An example of using ModularScopeDisplay

```
MSD1.Scope.SetFPS(FPS.ToString());
```

Since you may want optimal performance from the scopes you can call the AutoLOD function which will adjust the LOD (*Level Of Detail*) of the rendering so you get as much detail as possible with out the system dropping frames. The function though optional, is very useful if you wish to optimize the rendering speed of your project, since it calculates the best decimation factor to the image being rendered based on the performance level of your computer platform. The range of decimation varies depending on operating mode. If set manually the range is from none to 2048, if in auto mode, none to 16K or better in factors of 2 (for very large images).

AutoLOD must be used in a looping function for it to work best, like in a test/calibration function to find a point of optimum performance and quality for certain parameters. It should be noted however that using the AutoLOD will restrict you to single thread setups as the change in vertex count can cause the scopes to crash when operating in multithreaded or semi-threaded setups.

Using the scopes in a semi-threaded way has restrictions that need to be handled by the SDK developer – eg: one, the LOD should be locked when the scopes are running, Two, certain layout options should also be locked (i.e. RGB_QUAD, VTR_LIN) when the scopes are running. Now RGB_TRI and RGB_BLEND can swap with each other while running, but both do not handle a swap to RGB_QUAD at all. The Audio scope dot and line options can be swapped as the scopes are running.

It should be Noted that Most 3D scopes don't work very well operating in a semi-threaded way and work best when operating using a single thread loop, Though the RGB and YUV cubes do seem to operate with the semi-threaded method.

For example if the decimation factor is set to 2, and you have a 1920 x 1080 image, every other line is skipped. The horizontal pixel count stays intact for this version of the SDK.

Use the applicable function call (C#, VB or C++) based on the coding language you are using.

An example of using ScopeBase

```
sBase.AutoLOD(FPS.ToString());
```

An example of using ModularScopeDisplay

```
MSD1.Scope.AutoLOD(FPS.ToString());
```

The Render Function will take the data and render the results based on the parameters set by the programmer or user. The function renders the bitmap RGB file or RGB data array values once and displays the rendered data according to the configuration and other parameters that were previously set. If there are new renders to run, use the function in a loop. (replaced below).

Rendering is broken into two functions, the first is HT_DataPrep() which processes data given to it via one of three variables (HT_Byte[], HT_Memstr, or HT_File (test use only)) for video and only one variable for audio (HT_Aud_Buffer[sample,channel]). The second function is the HT_Render function, and it will render the prepared data and depending on your setup, will use one of two versions.

The first version has no arguments and is used with a Modular Scope Display not using the PreData_Publisher argument. The second render function requires that a Modular Scope Display use the PreData_Publisher argument and that the main application is setup to handle the PreData_Publisher events itself.

There are no timing constraints between the time a filename is changed to the time the render call is made. The function will merely skip the current render and come back again to run a render when the filename itself is stable. Render is the only function that has variation of data input, in that it'll take Bitmaps by file name, or you can use a variable of Memorystream or Byte Array without a header (replaced below)

The rendering system now allows for a semi-threaded system design. The HT_DataPrep function can be threaded to allow for parallel processing of data for multiple scopes, however given our tests you should not aim for more then 4 scopes running in parallel. The timing of the semi-threaded rendering system will drop data if data processing takes more time then the time it takes to update the data to be processed. Also you should only use one of the three data formats (Byte[], Memstream, file) for the video scopes when processing data.

An example of using ScopeBase

```
sBase.HT_Byte = ByteArray;
        or
sBase.HT_Memstr = MemstreamData;
        or
sBase.HT_File = Filename;

sBase.HT_DataPrep(); // Can be threaded
sBase.HT_Render();
```

An example of using ModularScopeDisplay

```
MSD1.scope.HT_Byte = ByteArray;
        or
MSD1.scope.HT_Memstr = MemstreamData;
        or
MSD1.scope.HT_File = Filename;

MSD1.scope.HT_DataPrep(); // Can be threaded
MSD1.scope.HT_Render();
```

4.3 Flag Description

There are four basic types of flags; A Data Read Mode (DRM), a Layout Control (LAC), a Level Of Detail (LOD), and a Luminance (LUM) flag.

All flags are readily available as public constants from scopebase for creating a configuration to be loaded into SelectConfig().

Refer to Flag *Appendix A* for more detailed info.

The **Data Read Mode** flag controls how to read the image data in and how to process the it. The "GL_DRM_BMP" flag tells the system that it is going to read a 24 or 32-bit bitmap, while flags that begin with "GL_DRM_" and end in"_PNT" informs the system that it will read a data pointer and the order of the values in the data pointer.

The **Layout Control** flag controls the layout and unit of measurement in the display. Some session types (VTR and RGB) have extra layout flags that change how the rendering system displays the data. The "GL_LAC_PER" uses a percent scale, "GL_LAC_RGB"uses the accual value of the data displayed, while "GL_LAC_IRE" displays the values in an IRE scale. An Example of the extra flags in the Vectorscope, if you just use the four basic flags and or includes the "GL_LAC_GRN" (Use green color) and "GL_LAC_PNT" (render Points) flags the display will use green dots to show the value of the data, but if you use "GL_LAC_CLR" (use the color value) and "GL_LAC_LIN" (render lines), while RGB Parade has two optional Flags GL_LAC_TRI (Tri-mode: RGB only) and GL_LAC QUAD (Quad-mode: Luma and RGB) with Tri-mode as a default if the flags are not used. **The new audio scopes use the GL_LAC_DB flag and some of the new audio scope also have extra flags for controling how the data is rendered.**

Level Of Detail is a must if a user does not want to use the AutoLOD function as this flag controls how many vertical lines it will skip, is displayed in the lower right corner. The value of 1 means it does every line, a 2 means every other line, 4 means every 4th line and etc.. All these flags begin with "GL_LOD_", with "GL_LOD_PERF" telling the system to use every vertical line in the image and "GL_LOD_NPERF" doing every other line.

The **Luminance** flag controls the equations used in all calculations involving Luminance, Hue and Saturation that get displayed in screen. In other word this controls the color space equations used. The "GL_LUM_YUV" flag uses the YUVColor space, the "GL_LUM_STD" flag uses the Standard Definition color space, and the "GL_LUM_HID" flag uses the High Definition Color space. **The new audio scopes use the GL_LUM_OFF flag since luminance is not part of audio calculations.**

The Modular Scope Displays can help in managing the scopes in that they help in swapping from one scope to another using the same GLControl display on the form. It's ChangeScope function does the scope swap and keeps the Thread Id with the scope, but requires that the scope gets it's parts configured afterward. The ChangeScopeAdv function does the same, but extracts any current scope setup information, stores it and uses it when a scope is swapped. This does mean some base setup initialization is required at the start, and that one loads the scope configurations into the MSD to keep track of them and use them as scopes swap from one to another.

4.4 Program Order of Functions

The order of the functions are listed below including optional functions.

1) Create a new variable of the scope
2) Initialize it with Init
3) Setup the SetupProjection
4) Enter the SelectConfig

->Loop start

5) HT_DataPrep
6) And HT_Render

->Loop end

The above order is the most optimal setup for configuring and running your program. You can change the order within certain limits to suit your needs, but be sure to use all the setup functions and clean up functions necessary to operate the program to prevent memory leaks.

Once the initialization functions have been executed they need not be run again. Only the Render functions need be called for fresh data to crunch and render in a looping fashion.

5.0 Sample Code

Sample code is provided as part of this SDK package. The sample code for this version is supplied as C# code. Using Visual Studio 2013, the C# syntax and code constructs are near identical.

6.0 Minimum System Requirements

The program can work in any Windows OS with Framework 2.0 and 3.5 installed. The only hardware you need to have is a graphics card with OpenGL 1.2 or better.

7.0 Supported Development Environments

The program was made with VS 2013, OpenTK, and done in C# .

8.0 Support and Contact Information

No technical support will be available for this product at this time. The supplied documentation will describe as much detail as needed for a great percentage of users.

Support will only be provided for pre-sales questions**.**

9.0 License

There is no software license attached to this toolkit or open source restrictions. You can do what you want with it. But you must follow the OpenTK license and restrictions.

I would however, appreciate that you don't share or give away this API kit to others without them paying for it through my web site. It's not that expensive.

Appendix A

Flag Name	Flag Type	Flag Value (Hex)	Function / Purpose
GL_DRM_BMP	Data Read Mode	0x1000	Configure system to read bitmaps
GL_DRM_RGB_PNT	Data Read Mode	0x2000	Configure system to data pointer of RGB format
GL_DRM_BGR_PNT	Data Read Mode	0x3000	Configure system to data pointer of BGR format
GL_DRM_RGBA_PNT	Data Read Mode	0x4000	Configure system to data pointer of RGBA format
GL_DRM_BGRA_PNT	Data Read Mode	0x5000	Configure system to data pointer of BGR format
GL_LAC_PER	Layout Code	0x0100	Set the Layout to Percent
GL_LAC_IRE	Layout Code	0x0200	Set the Layout to IRE scale
GL_LAC_RGB	Layout Code	0x0300	Set the Layout to char scale (0-->255)
GL_LAC_VTR_PNT	Layout Code VTR	0x0000	Configure the Vector scope to use Points
GL_LAC_VTR_LIN	Layout Code VTR	0x0400	Configure the Vector scope to use Lines
GL_LAC_VTR_GRN	Layout Code VTR	0x0000	Configure the Vector scope to use the Color Green
GL_LAC_VTR_CLR	Layout Code VTR	0x0800	Configure the Vector scope to use the color as is.
GL_LAC_RGB_TRI	Layout Code RGB	0x0000	Configure the RGB parade to display only RGB.
GL_LAC_RGB_QUAD	Layout Code RGB	0x0800	Configure the RGB parade to display Luma and RGB
GL_LAC_RGB_BLEND	Layout Code RGB	0x0400	Configure the RGB parade to Have RGB values blend.
GL_LAC_WFM_REG	Layout Code WFM	0x0000	Configure the WFM to display only Luma.
GL_LAC_WFM_CHROM	Layout Code WFM	0x0800	Configure the WFM to display Luma and Chroma.
GL_LAC_RGBC_PNT	Layout Code RGBC	0x0000	Configure the RGB Cube to render with Points
GL_LAC_RGBC_LIN	Layout Code RGBC	0x0400	Configure the RGB Cube to render with Lines
GL_LAC_RGBC_GRN	Layout Code RGBC	0x0800	Configure the RGB Cube to render only with Green
GL_LAC_RGBC_CLR	Layout Code RGBC	0x0000	Configure the RGB Cube to render with Color
GL_LAC_YUVC_PNT	Layout Code YUVC	0x0000	Configure the YUV Cube to render with Points
GL_LAC_YUVC_LIN	Layout Code YUVC	0x0400	Configure the YUV Cube to render with Lines
GL_LAC_HVTR_PNT	Layout Code HVTR	0x0000	Configure the HVTR to render with Points
GL_LAC_HVTR_LIN	Layout Code HVTR	0x0400	Configure the HVTR to render with Lines
GL_LAC_HVTR_TERRAIN	Layout Code HVTR	0x0800	Configure the HVTR to render as Terrain Map (Not in Yet)
GL_LAC_HWFM_PNT	Layout Code HWFM	0x0000	Configure the HWFM to render with Points
GL_LAC_HWFM_LIN	Layout Code HWFM	0x0400	Configure the HWFM to render with Lines
GL_LAC_HWFM_TERRAIN	Layout Code HWFM	0x0800	Configure the HWFM to render as Terrain Map (Not in Yet)
GL_LAC_HRGB_PNT	Layout Code HRGB	0x0000	Configure the HRGB to render with Points
GL_LAC_HRGB_LIN	Layout Code HRGB	0x0400	Configure the HRGB to render with Lines
GL_LAC_HRGB_TERRAIN	Layout Code HRGB	0x0800	Configure the HRGB to render as Terrain Map (Not in Yet)
GL_LAC_HYUV_PNT	Layout Code HYUV	0x0000	Configure the HRGB to render with Points
GL_LAC_HYUV_LIN	Layout Code HYUV	0x0400	Configure the HRGB to render with Lines
GL_LAC_HYUV_TERRAIN	Layout Code HYUV	0x0800	Configure the HRGB to render as Terrain Map (Not in Yet)

GL_LOD_PERF	Level Of Detail	0x0000	Sets the LOD to perfect, no skipping
GL_LOD_NPERF	Level Of Detail	0x0010	Sets the LOD to near perfect, Does every other line
GL_LOD_VHIGH	Level Of Detail	0x0020	Sets the LOD to Very High, Does every 4th line
GL_LOD_HIGH	Level Of Detail	0x0030	Sets the LOD to High, Does every 16th line
GL_LOD_NHIGH	Level Of Detail	0x0040	Sets the LOD to Near High, Does every 32th line
GL_LOD_MEDH	Level Of Detail	0x0050	Sets the LOD to Medium High, Does every 64th line
GL_LOD_MED	Level Of Detail	0x0060	Sets the LOD to Medium, Does every 128th line
GL_LOD_MEDL	Level Of Detail	0x0070	Sets the LOD to Medium Low, Does every 256th line
GL_LOD_NLOW	Level Of Detail	0x0080	Sets the LOD to Near Low, Does every 512th line
GL_LOD_LOW	Level Of Detail	0x0090	Sets the LOD to Low, Does every 1024th line
GL_LOD_VLOW	Level Of Detail	0x00A0	Sets the LOD to Very Low, Does every 2048th line
GL_LUM_YUV	Color Space	0x0001	Configures the Color Space Equation to Digital YUV
GL_LUM_HID	Color Space	0x0002	Configures the Color Space Equation to High Definition
GL_LUM_STD	Color Space	0x0003	Configures the Color Space Equation to Standard Definition
GL_LAC_DB	Audio Layout	0x0200	Set the Audio Layout scale based on DB
GL_LAC_OSC_LIN	Layout Code OSC	0x0800	Configure the oscilloscope to render with lines
GL_LAC_OSC_DOT	Layout Code OSC	0x0000	Configure the oscilloscope to render with dots
GL_LAC_PHA_LIN	Layout Code PHA	0x0800	Configure the Phase Meter to render with lines
GL_LAC_PHA_DOT	Layout Code PHA	0x0000	Configure the Phase Meter to render with dots
GL_LUM_OFF	Audio Space	0x0000	Nulls Lum flag for audio

Values of these can be found in the ScopeBase as Public Const Values of the same name.

Appendix B (Incomplete – sample doc)

ScopeBase:

Namespace: ScopeGLTools

Assembly: ScopeGLTools.dll

Is the abstract Base Class for all ScopeGL scopes in the DLL and would need to be used for making new ones.

C#
```
public abstract class ScopeBase
```

Derived classes: Scope_WFM, Scope_VTR, Scope_RGB, Scope_YUV, Scope_HST, Scope_OSC, Scope_PPM, Scope_PHA, Scope_HWFM, Scope_HVTR, Scope_HRGB, Scope_HYUV, Scope_RGBC, Scope_YUVC

Remarks:

While all ScopeGL scopes made with this abstract base class can be used into a semi-threaded way, Not all scopes made with it can be run in a semi-threaded way. Scopes that do a lot of data processing to prepare data for display should be instead run in a single thread.

Constructors (using derived versions)

```
ScopeBase(ref GL_Display)
Creates a scope that references a OpenTK GL control that it'll display to and will
internally message itself the conversion results of image or sound data into a Vertex Buffer
Object that the Rendering system will use.
```

```
ScopeBase(ref GL_Display, ref HT_DataPub)
Creates a scope that references a OpenTK GL control that it'll display to and will
externally message the conversion results of image or sound data into a Vertex Buffer Object
that the Rendering system will use and will require that VBO to be passed into the Rendering
Function.
```

Properties

DataPrep_Pub
Used for VBO transfer using a PreData_Publisher

_HT_External_Event
For finding out if the scope is set for internal or external handling of the VBO

HT_Byte
Loading a Byte array of image data into the scope for conversion before rendering.

HT_Memstr
Loading a Memorystream of image data into the scope for conversion before rendering.

HT_File
Giving the path and filename of a bitmap file for conversion before rendering.

HT_Aud_Buffer
Loading a Byte array of audio data into the scope for conversion before rendering. [Data, Channels]

_HT_ID
The ID number of a scope to ensure that data is sent to the right scope.

Is3DScope
For finding out if the scope is a 3D scope.