Getting started with VSTGUI4 and RackAFX

Will Pirkle

Before you begin to understand and use the RackAFX GUI API, you need to familiarize yourself with a few basics of VSTGUI4 as well as the special RackAFX structures and messages that handle the customized GUI API. In the rest of the Advanced GUI Design modules, you will need to be familiar with these basic ideas to make the coding easier. In this module we will cover:

- VSTGUI4 Introduction
- Installing VSTGUI4
- graphics and control-tags in XML
- Adding the advanced GUI Files to RackAFX
- the *showGUI()* function in RackAFX

VSTGUI4

VSTGUI4 is a free GUI library that is designed specifically for plug-in GUIs, although it can be used for any GUI that requires controls like knobs, sliders, switches, buttons and the like. VSTGUI was originally designed at Steinberg, LLC for the VST plug-in API. The library is platform independent so your GUIs will show up identically on WinOS or MacOS. VSTGUI4 is the latest incarnation, the previous version is VSTGUI3 and is still available, though it is retired and the designers do not update it any longer. In this and following documents the term “VSTGUI” is synonymous with “VSTGUI4.”

Simply stated, VSTGUI is a library of C++ objects that encapsulate GUI controls, views and behaviors. The objects are created just like any other C++ objects and have member variables called “attributes” and member functions that are called during the course of operation. In this respect VSTGUI is no different that any other C++ library. The underlying details of operation that allow the library to be platform independent are fairly well hidden from the user and in general you will never need to deal with that code, but the information is there if you need it. VSTGUI is NOT a static (.lib) or dynamic (.dll) library. You do not need to link to anything to use it - VSTGUI is really just a set of folders full of C++ objects and helper code.

One major advantage to using VSTGUI4 is that while the GUI itself is created using the usual C++ operations of object instantiation, placement and attribute setting, the GUI can be described in an XML file. The XML file description method of packaging the GUI is optional but it is also very powerful and simple. The GUI Designer in RackAFX writes the XML file for you based on the controls you drag and drop into the interface. In fact, RackAFX could be used as a standalone VSTGUI4 design package.The name of the XML file is ***RackAFX.uidesc*** where “uidesc” is short for “ui description.” We will examine some parts of this file later on in this module, and then again in future modules.

Installing VSTGUI4

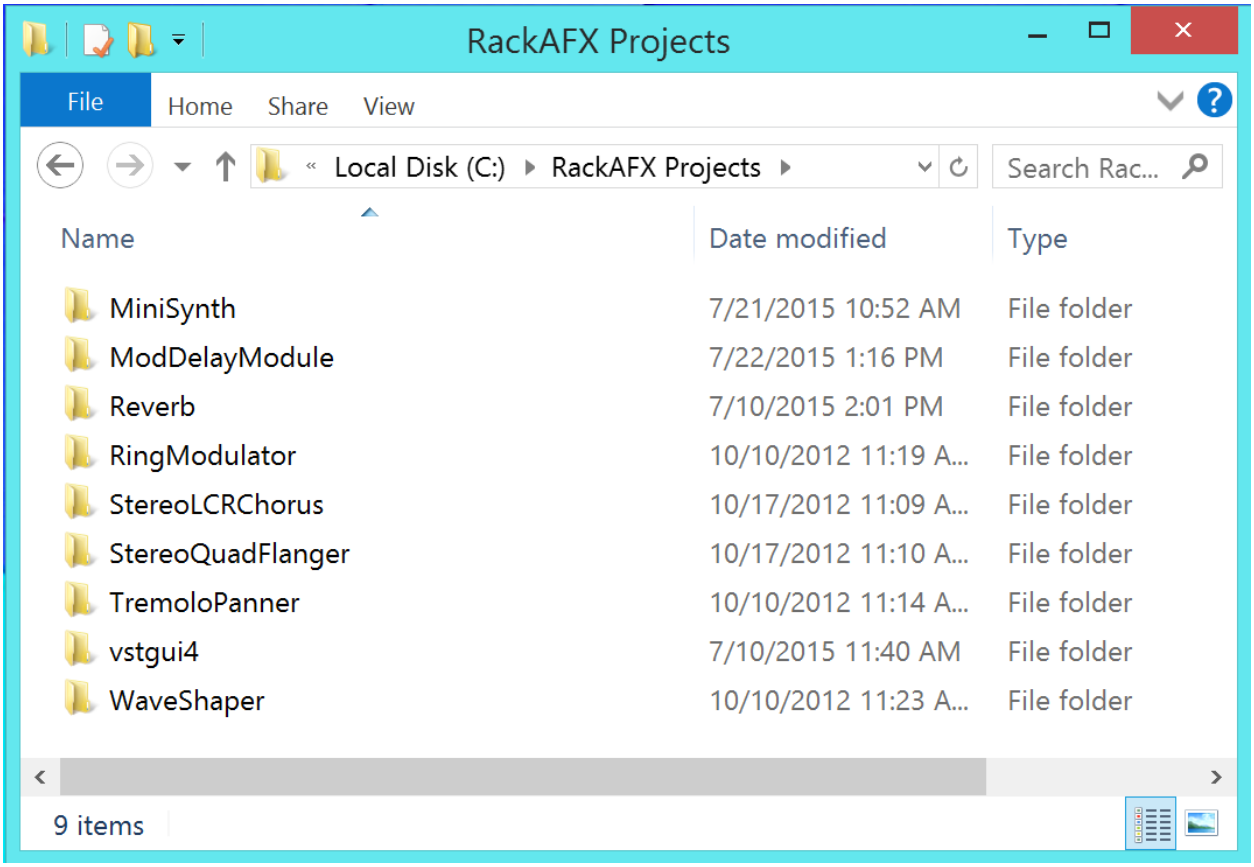
The easiest way to install VTSGUI4 is with RackAFX itself. Choose *Utilities->Install VSTGUI4* to install the library. The installation process simply copies the VSTGUI4 folder (and subfolders) into your RackAFX Project directory that you set with *View->Preferences* and does not alter your registry or install any executables or DLLs. RackAFX installs an abbreviated version of the library that omits the tutorials, tests, and Doxygen (auto documentation). These are not needed for using the library.

You can install the full package manually by downloading the library at

http://sourceforge.net/projects/vstgui/

and then copying it into your RackAFX Projects folder. RackAFX v6.6 uses and installs VSTGUI 4.2 though newer versions are available. Since the library is really just a folder, it is easy to upgrade or replace - just copy the new version (or old version if you are going backwards). My experience is that VSTGUI 4.2 is the most stable and tested version. If you install a newer version and experience bugs or visual issues that were not present previously, just remove the new folder and copy the older one back into the RackAFX Projects folder.

It is important that the VSTGUI4 folder be located in the same folder as your RackAFX Projects in order to use the supplied custom VSTGUI objects that allow you to take advanced control over your GUI. If you have some compelling reason NOT to install the folder in your RackAFX Projects folder, then you will need to alter the #include paths and/or your Visual Studio project's *Additional Include Directories* attribute. After installation, your folder hierarchy should look something like this (where my project folder is named "RackAFX Projects") — note the **vstgui4** folder:



Open the vstgui4 folder and have a look at its subfolders. The documentation for the VSTGUI package is located in

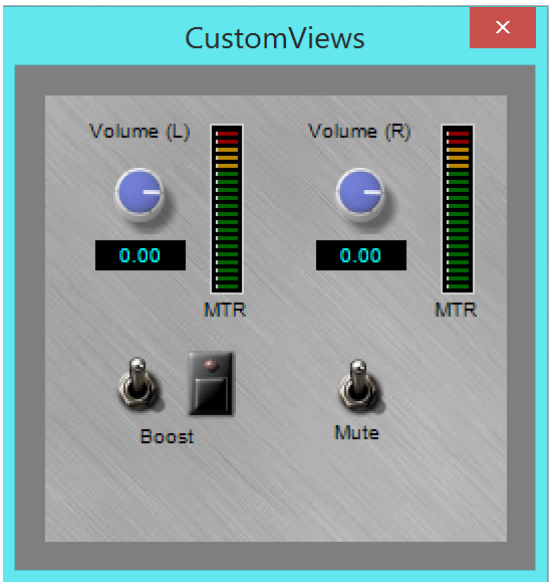
..\vstgui4\vstgui\Documentation\Index.html

You can also find a ton of information in the archives of the VSTGUI mailing list located at the VSTGUI website above. With the VSTGUI folder installed parallel to your project files, we are ready to begin!

VSTGUI Objects and Attributes

The best way to get started is to dive right in and have a look at the XML file that RackAFX creates for you in the GUI Designer. **Although we are going to be poking around in the XML code, don't fear — you do not have to write any XML code!** However, you do need to understand how the XML relates to the C++ objects, and on occasion you may need to find attribute values in this file. We will get into the details of the objects in the next module.

Open the project that accompanies this module named **CustomViews** and examine the XML file in Visual Studio. The file is located in the Resources filter and is named *RackAFX.uidesc*. The GUI for the project looks like this:



You can see that all the controls are embedded in a CViewContainer object with a lightgreymetal background. There are 2 RackAFX Knob Groups, 2 LED meters, 2 Boost buttons and 1 Mute button.

Bitmaps (PNG file graphics)

Before we look at the XML code, have a look around the .uidesc file and you will find that it is broken into "chunks." Locate the <bitmaps> chunk and you will see all the bitmaps that are available in your GUI Designer:

```
<bitmaps>
        <bitmap name="darkgreyvinyl"
                ninepartiled-offsets="0, 0, 0, 0"
                path="darkgreyvinyl.png"
                rafxtype="backgnd" />
        <bitmap name="bluemetal"
                ninepartiled-offsets="0, 0, 0, 0"
                path="bluemetal.png"
                rafxtype="backgnd" />
        <bitmap name="greenmetal"
                ninepartiled-offsets="0, 0, 0, 0"
                path="greenmetal.png"
                rafxtype="backgnd" />

etc…
```

Each bitmap has several attributes:
- name - the name you see in the GUI Designer
- path - the full filename with extension
- nineparttiled-offsets - offsets used for the nine-part-tiled bitmap (generally we avoid anything but 0,0,0,0)
- rafxtype - a custom attribute that RackAFX uses to filter the graphics in the GUI Designer

Control-Tags
More importantly, find the chunk marked <control-tags>:

```
<control-tags>
        <control-tag name="Volume (L)" tag="0" />
        <control-tag name="Volume (R)" tag="1" />
        <control-tag name="Boost" tag="2" />
        <control-tag name="Mute" tag="3" />
        <control-tag name="L" tag="4" />
        <control-tag name="R" tag="5" />
```

etc…

This is a key concept in VSTGUI programming — each control is connected to the underlying plugin variable with a control-tag which consists of a name (string) and tag (integer) value.

You will notice that the VSTGUI tag values are NOT the same as the RackAFX Control ID values in this file. This is because I have used a technique called "control mapping" to create a zero-indexed set of tags for your plug-in. I decided to do this mapping to make it easier to integrate VSTGUI with some of the existing RackAFX functions. If you design your own pure-custom GUI (in module 5), you do not need to do this - you may simply use the RackAFX Control ID values as your tags.

When you create custom views, you may need to reference the bitmap names as well as the control-tag values in this file.

Additional RackAFX Files
To use the advanced GUI features, you will need to add two files to your RackAFX project. In order to keep the projects lean, RackAFX does not automatically add the files into your Visual Studio solution but it does copy them into your Project folder. You need to add these files to any RackAFX project that uses the advanced GUI API. The files are:

GUIViewAttributes.h
GUIViewAttributes.cpp

In Visual Studio, right-click on your Solution and choose *Add->Existing Item…* and then browse your project folder and select the two files above.

The *pluginconstants.h* file has changed in this revision for the new GUI API. Near the top of this file you will find the GUI message enumeration:

```
// --- messages
enum {GUI_DID_OPEN,
      GUI_WILL_CLOSE,
      GUI_TIMER_PING,
      GUI_CUSTOMVIEW,
      GUI_SUBCONTROLLER,      /* CURRENTLY NOT USED */
      GUI_HAS_USER_CUSTOM,
      GUI_USER_CUSTOM_OPEN,
      GUI_USER_CUSTOM_CLOSE,
      GUI_USER_CUSTOM_SIZE};  /* CURRENTLY NOT USED */
      GUI_RAFX_OPEN,
      GUI_RAFX_CLOSE,
      GUI_RAFX_INIT,
      GUI_RAFX_SYNC};
```

This **bold** items are the currently supported messages that will be delivered to your plug-in when the GUI loads or unloads.

GUI_DID_OPEN
This message is delivered after the GUI has been created. It allows you to do any custom post-creation stuff. We will use this message when we cache custom view pointers in module 4. NOTE: this message is NOT called for pure-custom GUIs.

GUI_WILL_CLOSE
This message is delivered after the GUI just before the GUI is destroyed. It gives you the chance to clean up variables or do any other pre-destruction chores. We will use this message when we cache custom view pointers in module 4. NOTE: this message is NOT called for pure-custom GUIs.

GUI_TIMER_PING
VSTGUI4 uses a timer notification to update the GUI in order to implement preset selections (the GUI controls move when you select a preset) as well as for animation of the LED meters and other custom views you may implement. RackAFX uses a similar system to update its GUI and show the LED meters in action. The GUI_TIMER_PING message will occur every 50 milliseconds, which is the timer interval. You will only need to respond to this message if you have a custom control that needs repainting/updating. We will use this message when we create a cool scrolling waveform view in module 4 — the waveform moves across the window and this timer notification makes the animation work.

GUI_CUSTOMVIEW
This message will be delivered once for each Custom View that you implement. You respond to the message by instantiating your custom view and then passing a pointer to the view object back to the caller. We will use this message in the next module where we learn to subclass VSTGUI objects.

GUI_HAS_USER_CUSTOM
This is a query-message from the GUI host asking you if your plug-in has a full custom GUI, that is, a GUI that you code completely on your own, without the use of the RackAFX GUI Designer. We will use this message in module 5 when we implement a full custom GUI.

GUI_USER_CUSTOM_OPEN
This message tells you to create your full custom GUI. We will use it in module 6.

GUI_USER_CUSTOM_CLOSE
This message tells you to destroy your full custom GUI. We will use it in module 6.

You will also see the messages GUI_RAFX_OPEN, GUI_RAFX_CLOSE, GUI_RAFX_INIT and GUI_RAFX_SYNC — these are used for the RAFX generated GUI another change that happened with version v6.6. You can see where they are used by examining the showGUI() function in the plugin.cpp file. The messages trigger the GUI operations via the Sock2VST3 library your plugins are always linked to. You don't need to worry about these messages or deal with them in any way.

VSTGUI_VIEW_INFO

Below the message enumeration (in *pluginconstants.h*) is a **VSTGUI_VIEW_INFO** structure definition. Do not be intimidated - you really only need to use a few of the attributes in the structure. And right now we only need to focus on one of them:

unsigned int message; // type of info message

The RackAFX *showGUI( )* function and VSTGUI_VIEW_INFO

In order to use these advanced GUI programming techniques, you will need to add code to the RackAFX function *showGUI()*. The showGUI() function is the only function that is used for all advanced GUI stuff. The prototype for the function is:

virtual void* __stdcall showGUI(void* pInfo);

If you are creating a new project with RackAFX v6.6 or later, this function is already declared in your derived class object's .h file. If you are modifying a project built in RackAFX v6.5 or earlier, you will need to override this function manually by adding the line above to your <project>.h file and then the implementation below to your <project>.cpp file. The most important thing about this function is to notice that its argument is a **void*** and it returns a **void*** variable. The use of void*'s to cloak arguments and return variables is an old C++ trick that allows for future expansion without breaking old code.

When *showGUI()* is called, the void* **pInfo** that is passed into it is actually a **VSTGUI_VIEW_INFO*** and it contains all the information you need to accomplish any of the advanced GUI features. To use it, you need to uncloak the pointer and then decode the message. That is easily done in the **bold** code below. A switch/case statement is used to decode the message.

Notice that all messages return NULL except for the GUI_CUSTOMVIEW message which returns a pointer to the view object cloaked as a void*. Also, notice the call to the base class at the top of the function - this if for future updates; currently the base class function generates your RAFX GUI when you use RackAFX.

```
void* __stdcall CCustomViews::showGUI(void* pInfo)
{
	// --- ALWAYS try base class first in case of future updates
	void* result = CPlugIn::showGUI(pInfo);
	if(result)
		return result;

	// --- uncloak the info struct
	VSTGUI_VIEW_INFO* info = (VSTGUI_VIEW_INFO*)pInfo;
	if(!info) return NULL;

	switch(info->message)
	{
		case GUI_DID_OPEN:
		{
			// called after GUI opens
```

```
                return NULL;
            }

            case GUI_WILL_CLOSE:
            {
                // called before GUI closes
                return NULL;
            }

            case GUI_CUSTOMVIEW:
            {
                // --- create custom view, return a CView* cloaked as void*
                return NULL; // return NULL if not supported
            }

            case GUI_HAS_USER_CUSTOM:
            {
                // --- set to true if you have a custom GUI
                info->bHasUserCustomView = false;
                return NULL;
            }

            // --- create your custom GUI
            case GUI_USER_CUSTOM_OPEN:
            {
                return NULL;
            }

            // --- destroy your custom GUI
            case GUI_USER_CUSTOM_CLOSE:
            {
                return NULL;
            }

            // --- handle paint-specific timer stuff
            case GUI_TIMER_PING:
            {
                return NULL;
            }
        }

        return NULL;
}
```

This completes this basic information module. In the next module, we will implement a Custom View and write the code that deals with the GUI_CUSTOMVIEW message. In module 4 we will create a custom control that needs the GUI timer ping and in module 5 we will create a full custom GUI in which you will write the code for the GUI and design it programmatically instead of using the RackAFX GUI Designer.

You now know how to:

- install VSTGUI4
- poke around in the *RackAFX.uidesc* file to see how the GUI is described in XML
- add the advanced GUI files to your RackAFX Project
- uncloak the **VSTGUI_VIEW_INFO*** that is passed to the *showGUI()* method
- decode the *message* member of the **VSTGUI_VIEW_INFO** structure in the *showGUI()* method

You also know that each GUI control is linked to an underlying plugin variable with the **control-tag** attribute.

References:

VSTGUI4 Files and Documentation: http://sourceforge.net/projects/vstgui/