Plugin Consultant Guide

Plugin Consultant is a tool for monitoring and logging VST API communication between VST hosts and VST plugins. It started as a debugging code inside another project but when I found out that I needed to repeat this code in many places and also needed more control over what is logged, I decided to create a standalone tool that can useful for every VST developer.

Although Plugin Consultant is a simple to use tool, the following pages will explain it in more detail and will focus on some more advanced issues and tips.

Since logging everything that passes between the host and plugin can sometimes generate be too much information or cause a performance depredation, you can control what is being logged.

How does it work?

Plugin Consultant is a DLL that exports the main() function, making VST hosts believe this is a VST plugin. When a host calls Plugin Consultant's main() function, Plugin Consultant ask the user to select a real VST DLL file. After the VST plugin DLL is selected, all VST API communication between the host and the plugin is proxied by Plugin Consultant. Plugin Consultant does not change anything in the plugin or host behavior but it does watch all the VST API calls for monitoring and logging purposes.

Plugin Consultant also proxies the plugin AEffect structure, meaning that all the plugin information (flags, number of programs, unique ID, etc.) is not changed by Plugin Consultant. The only thing that might look different in the host when using Plugin Consultant, is that the host shows the plugin as 'PluginConsultant' and does not show the original plugin name.

Since the VST plugin implements the editor GUI, Plugin Consultant user interface is displayed in its own separate window but it is still running in the host process.

Plugin Consultant is using the VST C API (defined in AEffect.h and aeffectx.h) for monitoring and logging although most developers use the VST SDK C++ wrapper (defined in AudioEffect.hpp, audioeffectx.h and AEffEditor.hpp). This allows Plugin Consultant to be used with plugins or hosts that are developed in C, Delphi, SynthEdit and other languages or tools. More details about how to translate the C API to C++ will be provided later in this guide.

Using Plugin Consultant

Plugin Consultant DLL Location

For most hosts you should put the Plugin Consultant DLL in the VST folder used by the host so you can later select Plugin Consultant from within the host interface. Some hosts (e.g. energyXT, MiniHost and VSTHost) allow you to load plugins from any folder. Since you probably won't use Plugin Consultant on a daily basis you might want to put outside the VST folder until you need it to avoid some hosts from opening it each time they perform a plugin scan.

Plugin Scan

Some hosts, such as Cubase, perform a plugin scan of the VST folder, usually when they start, and call the main() function of every plugin. If Plugin Consultant is located in the VST folder then it will ask for the VST plugin to log. You should select the same plugin that you are going to log later in the actual session since the host might store information about this plugin and this information should match the plugin that will be logged in the session. You can also see using Plugin Consultant log what the host actually does with the plugin in the plugin scan process. Some hosts will cache the scan information and will not load Plugin Consultant each time they start. In this case you will need to tell the host to re-scan the plugin folder before starting to log a new plugin.

If you are using Cakewalk VST Adapter then the plugin scan happens when the adapter scans your VST plugins and not when a Cakewalk host starts. This mean that before a log session of a new plugin, you need to let the adapter scan Plugin Consultant, select the plugin to log, and only then run the Cakewalk host and use Plugin Consultant to log the same plugin you selected in the scan process.

Selecting a Plugin

When you want to log a plugin you will need to load Plugin Consultant in your host. Plugin Consultant looks to the host as the VST you selected when the host loaded Plugin Consultant in the plugin scan. For example, in Cubase, if you selected an effect in the plugin scan then Plugin Consultant will appear in Cubase as an effect. If you selected an instrument in the plugin scan then Plugin Consultant will appear in Cubase as an instrument.

Some hosts do not do a plugin scan and do not call main() function of the DLLs in the plugin folder. In these hosts the plugins will not be identified as effects or instruments since the main() function needs to be called in order to get this information. In other hosts you can even load a plugin by providing a DLL file name or by dropping the plugin DLL file on the host. With these hosts you can just select to load Plugin Consultant or PluginConsultant.DLL file.

Once Plugin Consultant is loaded by the host it will ask you to choose a VST plugin DLL file to log. After you select a VST plugin the logged plugin will load as usual (although its name will appear as PluginConsultant) and you can use it and use its GUI (if it has one). In addition the Plugin Consultant window will appear as a different application in the Windows taskbar. You cannot close Plugin Consultant window manually, it will close automatically when the logged plugin is unloaded by the host.

Plugin Consultant Window

On the top of the Plugin Consultant window you can see in the title bar the name of the VST DLL being logged. On the top right there is a minimize button for minimizing the Plugin Consultant window, you can always restore the window by clicking on the Plugin Consultant taskbar icon.

On the main part of the window you can view one of the 3 VST API categories (more on that in the Monitoring section) by clicking on categories tabs.

To the left there are several buttons to control the monitoring and logging, a help button to view this guide, a logo that also opens the About dialog and a useful Donate button!

Monitoring

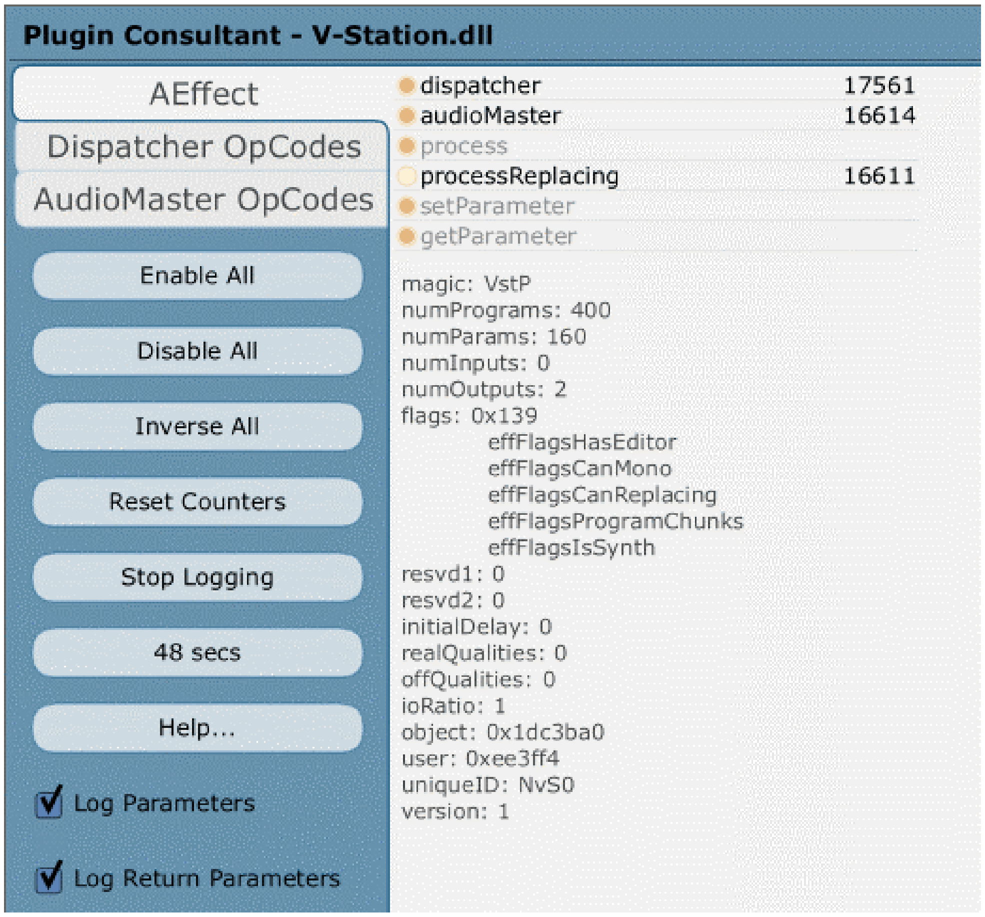
Plugin Consultant monitors all VST API calls between the plugin and the host and displays in real time what functions and opcodes were used in the session and how many times they were called. In addition you can see the log of all the calls, more details on that in the Logging section.

The Reset Counters button resets all the function and opcodes counters and makes all the function and opcodes look like they were not called so far. This is useful when you want to see if a certain function or opcode is called in a certain scenario and you don't care about previous calls.

A Time Display button bellow the Reset Counters shows how much time passed since the logged plugin was loaded, it can be reset back to zero by clicking it.

You can switch between different the VST API categories pages, to monitor different parts of the API, by clicking on the categories tabs.

AEffect Page



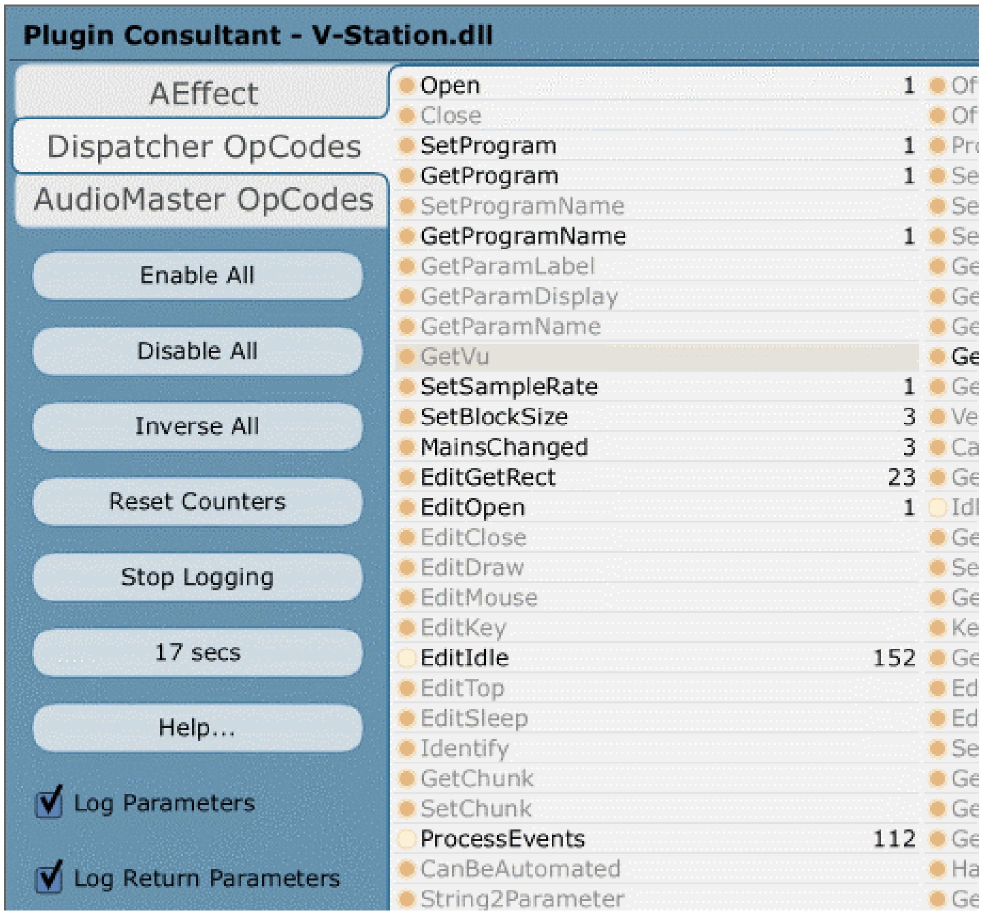
```
dispatcher            17561
audioMaster           16614
process
processReplacing      16611
setParameter
getParameter

magic: VstP
numPrograms: 400
numParams: 160
numInputs: 0
numOutputs: 2
flags: 0x139
        effFlagsHasEditor
        effFlagsCanMono
        effFlagsCanReplacing
        effFlagsProgramChunks
        effFlagsIsSynth
resvd1: 0
resvd2: 0
initialDelay: 0
realQualities: 0
offQualities: 0
ioRatio: 1
object: 0x1dc3ba0
user: 0xee3ff4
uniqueID: NvS0
version: 1
```

This page displays the AEffect structure that is returned to the host from the plugin main() function. The 6 VST C API function are highlighted when called for the first time and a counter shows how many time each function was called. 4 of the C functions are similar to the AEffect class C++ methods (process, processReplacing, setParameter and getParameter) and the other two are used to dispatch opcodes from the host to the plugin (dispatcher) or from the plugin to the host (audioMaster). The AudioEffect and AudioEffectX classes implement these functions for you in the C++ VST SDK and convert the opcodes to C++ methods, usually with names similar to the opcodes names. For more details about opcodes see the Dispatcher and AudioMaster sections.

The logged plugin AEffect structure data members are also displayed in this page The AEffect structure display is refreshed (if changed) every time the plugin calls the host since it might have changed parameters, for example when the plugin calls ioChanged().

Dispatcher OpCodes Page



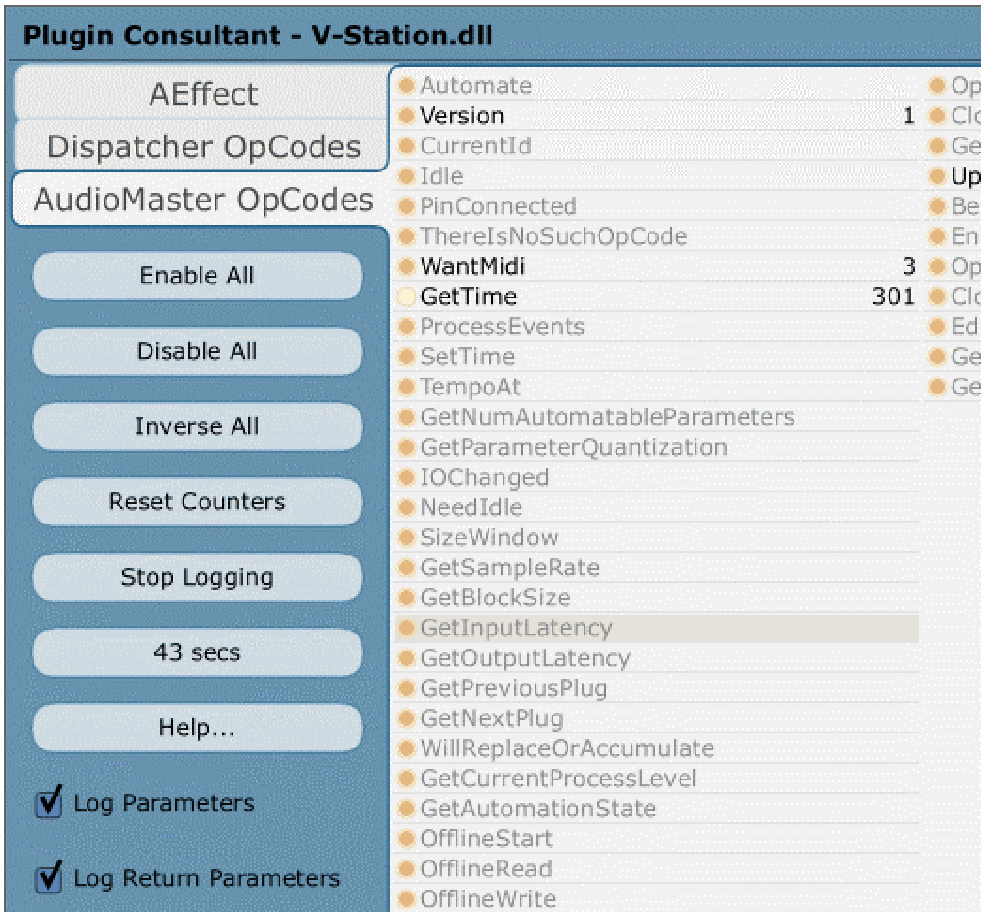
Open	1	Of
Close		Of
SetProgram	1	Pr
GetProgram	1	Se
SetProgramName		Se
GetProgramName	1	Se
GetParamLabel		Ge
GetParamDisplay		Ge
GetParamName		Ge
GetVu		Ge
SetSampleRate	1	Ge
SetBlockSize	3	Ve
MainsChanged	3	Ca
EditGetRect	23	Ge
EditOpen	1	Idl
EditClose		Ge
EditDraw		Se
EditMouse		Ge
EditKey		Ke
EditIdle	152	Ge
EditTop		Ed
EditSleep		Ed
Identify		Se
GetChunk		Ge
SetChunk		Ge
ProcessEvents	112	Ge
CanBeAutomated		Ha
String2Parameter		Ge

This page displays the opcodes that are sent from the host to the plugin via the dispatcher() function. The opcodes are displayed without the 'eff' prefix, for example effOpen is displayed as Open. If you use the C++ SDK then the AudioEffect class invokes the open() method when the effOpen opcode is sent by the host via the dispatcher() function. Usually the effXXX opcodes (displayed as XXX in this page) will invoke a method named XXX in the AudioEffect class. There are some exceptions to this, most important is the effMainsChanged opcode that invokes the suspend() and resume() methods, depending on a parameter sent by the host. You will need to see the log in order to know if the host requires the plugin to suspend or resume.

Opcodes sent for the GUI editor have 'effEdit' as a prefix, for example effEditOpen will invoke the open() method of the AEffEditor class and will be displayed in this page as EditOpen.

The opcode names are ordered in the same order they appear in the AEffect.h and affectx.h.

AudioMaster OpCodes Page



Automate	
Version	1
CurrentId	
Idle	
PinConnected	
ThereIsNoSuchOpCode	
WantMidi	3
GetTime	301
ProcessEvents	
SetTime	
TempoAt	
GetNumAutomatableParameters	
GetParameterQuantization	
IOChanged	
NeedIdle	
SizeWindow	
GetSampleRate	
GetBlockSize	
GetInputLatency	
GetOutputLatency	
GetPreviousPlug	
GetNextPlug	
WillReplaceOrAccumulate	
GetCurrentProcessLevel	
GetAutomationState	
OfflineStart	
OfflineRead	
OfflineWrite	

This page displays the opcodes that are sent from the plugin to the host via the audioMaster callback. The opcodes are displayed without the 'audioMaster' prefix, for example audioMasterVersion is displayed as Version. If you use the C++ SDK then the AudioEffect and AudioEffectX classes sometimes provide method that hide these details from the programmer. For example the getMasterVersion() will send the audioMasterVersion opcode to the host.

The opcode names are ordered in the same order they appear in the AEffect.h and affectx.h.

Logging

Plugin Consultant logs all the communication between the host and the plugin using the Win32 OutputDebugString function. In order to view the log you will need to an application that displays Windows debug output, or view the log within your development environment. Examples of both options are described later in this section.

Since logging all host-plugin communication can sometimes generate too much information and cause performance depredation, you can control what is logged.

Left to each function or opcode name there is a light that shows if this function or opcode is logged. You can click the function or opcode name to toggle logging on/off. Note that

that turning off logging for the dispatcher function in the AEffect page will turn off logging for all the opcodes in the Dispatcher OpCodes page, even if the light left to the opcodes is on. In the same way, turning off logging for the audioMaster function will turn off logging for all the opcodes in the AudioMaster OpCodes page.

The Enable All button turns logging on for all functions and opcodes in all pages. The Disable All button turns logging off for all functions and opcodes in all pages. The Inverse All will toggle the logging on/off state for all functions and opcodes in all pages.

Clicking the Time Display button, in addition to reseting the time display, will also send a log line stating that the time was reset: "PluginConsultant timer reset after XXX secs".

The Stop Logging button will stop all log output, even if logging is on for specific functions or opcodes. The button will then turn into a Start Logging button that will enable logging again when clicked.

Log Output Format

Plugin Consultant log will always start with the name of Plugin Consultant DLL and the time when the log was started.

(3324) PluginConsultant.dll:Plugin Consultant - PluginConsultant.dll
(3324) PluginConsultant.dll:22 May 2005 11:47:01 am

A log line will always start with the ID thread of the current thread followed by the name of the logged plugin DLL name, for example:

(2152) V-Station.dll:Loaded V-Station.dll
(3036) V-Station.dll:DLL_THREAD_ATTACH

Note that the log viewer might add its own information before each Plugin Consultant log line, for example: current time, process ID, etc.

Functions and opcodes are logged with their parameters (both input and output) names and value, if they have any.

Example of opcode without parameters or return value:

(2152) V-Station.dll:effOpen

Example of opcode with parameters (this opcodes invokes the resume method in the C++ SDK):

(2152) V-Station.dll:effMainsChanged state=1

Example of opcode with return values:

(2152) V-Station.dll:effGetProductString

(2152) V-Station.dll:effGetProductString returned bool=1 text=V-Station

Examples of opcode with both parameters and return values:

(5184) energyXT.dll:effCanDo text=midiProgramNames

(5184) energyXT.dll:effCanDo returned long=0

Note that char* parameters are displayed as a text string. Structures and buffers pointers are currently displayed as pointers only and without the structure or buffer contents:

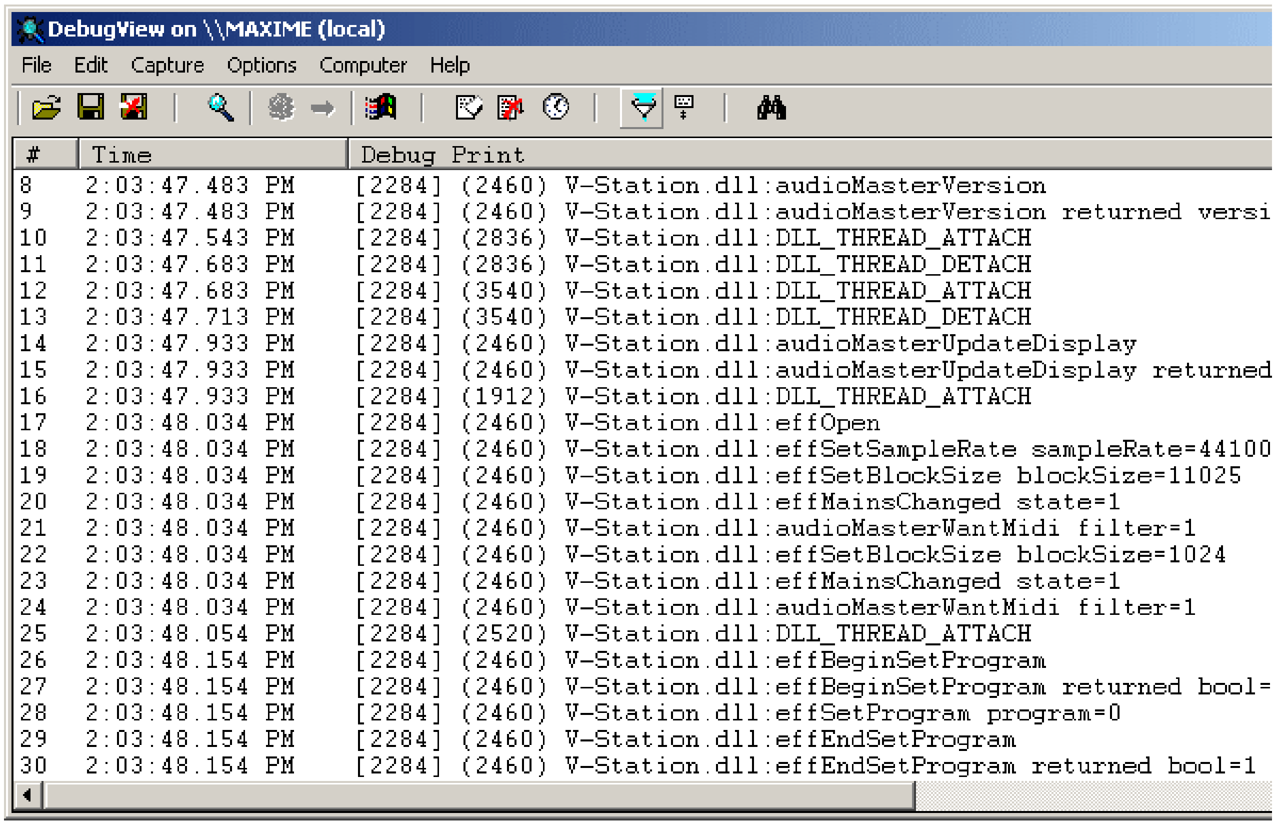
(5184) energyXT.dll:effGetOutputProperties index=0 properties=0012F494

You can turn off logging of input or output parameters using the Log Parameters and Log Return Parameters check boxes.

Logging with DebugView

Note: Although this section describes working with DebugView, other debug output viewers can be used as well and will probably provide similar functionality.

DebugView is a free Windows applications that lets you monitor debug output. You you need such an application when you are not using a debugger but you still want to capture the Plugin Consultant log. Remember to run DebugView before you start the Plugin Consultant session so you will be able to see all logs from the beginning of the sessions. Also make sure the Capture Win32 option is checked.


DebugView on \\MAXIME (local)
File Edit Capture Options Computer Help

```
#   Time              Debug Print
8   2:03:47.483 PM    [2284] (2460) V-Station.dll:audioMasterVersion
9   2:03:47.483 PM    [2284] (2460) V-Station.dll:audioMasterVersion returned versi
10  2:03:47.543 PM    [2284] (2836) V-Station.dll:DLL_THREAD_ATTACH
11  2:03:47.683 PM    [2284] (2836) V-Station.dll:DLL_THREAD_DETACH
12  2:03:47.683 PM    [2284] (3540) V-Station.dll:DLL_THREAD_ATTACH
13  2:03:47.713 PM    [2284] (3540) V-Station.dll:DLL_THREAD_DETACH
14  2:03:47.933 PM    [2284] (2460) V-Station.dll:audioMasterUpdateDisplay
15  2:03:47.933 PM    [2284] (2460) V-Station.dll:audioMasterUpdateDisplay returned
16  2:03:47.933 PM    [2284] (1912) V-Station.dll:DLL_THREAD_ATTACH
17  2:03:48.034 PM    [2284] (2460) V-Station.dll:effOpen
18  2:03:48.034 PM    [2284] (2460) V-Station.dll:effSetSampleRate sampleRate=44100
19  2:03:48.034 PM    [2284] (2460) V-Station.dll:effSetBlockSize blockSize=11025
20  2:03:48.034 PM    [2284] (2460) V-Station.dll:effMainsChanged state=1
21  2:03:48.034 PM    [2284] (2460) V-Station.dll:audioMasterWantMidi filter=1
22  2:03:48.034 PM    [2284] (2460) V-Station.dll:effSetBlockSize blockSize=1024
23  2:03:48.034 PM    [2284] (2460) V-Station.dll:effMainsChanged state=1
24  2:03:48.034 PM    [2284] (2460) V-Station.dll:audioMasterWantMidi filter=1
25  2:03:48.054 PM    [2284] (2520) V-Station.dll:DLL_THREAD_ATTACH
26  2:03:48.154 PM    [2284] (2460) V-Station.dll:effBeginSetProgram
27  2:03:48.154 PM    [2284] (2460) V-Station.dll:effBeginSetProgram returned bool=
28  2:03:48.154 PM    [2284] (2460) V-Station.dll:effSetProgram program=0
29  2:03:48.154 PM    [2284] (2460) V-Station.dll:effEndSetProgram
30  2:03:48.154 PM    [2284] (2460) V-Station.dll:effEndSetProgram returned bool=1
```

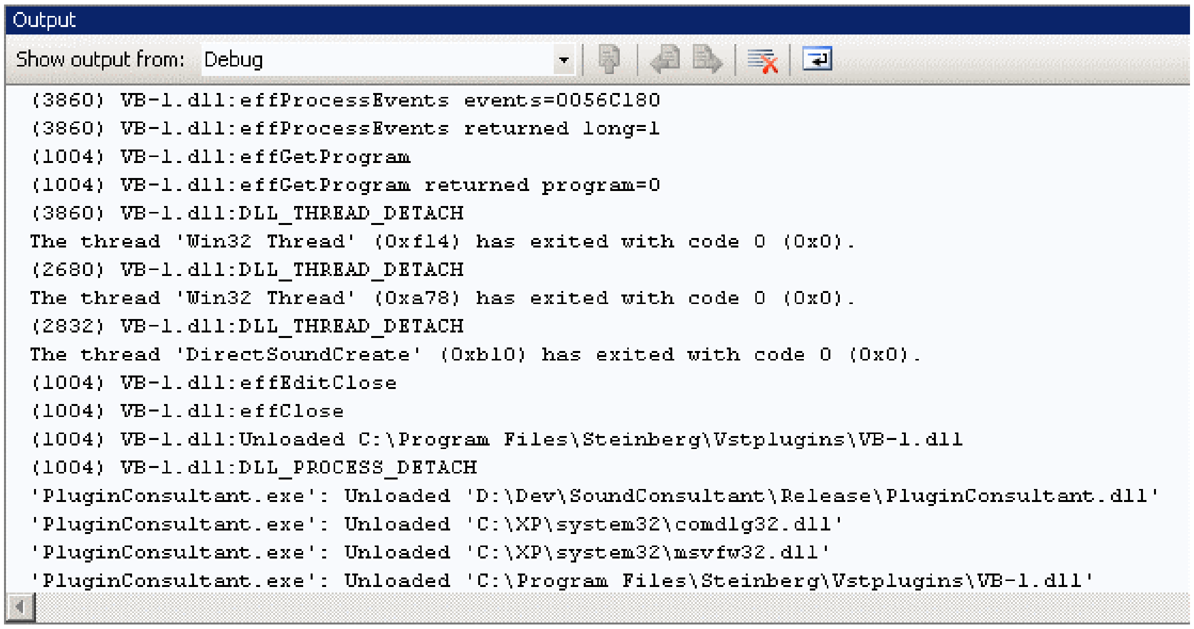
Note that DebugView adds its own fields in the log, before Plugin Consultant fields: log line serial number, log time and process ID.

You can also use DebugView filtering feature to filter the log output or highlight specific log lines. Its recommended not to use DebugView to filter out specific opcodes or functions since doing so using Plugin Consultant control will give much better performance.

Logging with Visual C++ 2005 Express Edition

Note: Although this section describes working with Visual C++ 2005 Express Edition, other development environments can be used as well and will probably provide similar functionality.

When you debug your plugin or host with Visual C++ 2005 Express Edition you will be able to view Plugin Consultant log output in the Visual C++ Output window. Visual C++ might add some additional log output of its own, usually describing thread and DLL events. You can identify Plugin Consultant logs by identifying the VST DLL name being logged. Visual C++ does not add additional information to Plugin Consultant log output.



```
(3860) VB-1.dll:effProcessEvents events=0056C180
(3860) VB-1.dll:effProcessEvents returned long=1
(1004) VB-1.dll:effGetProgram
(1004) VB-1.dll:effGetProgram returned program=0
(3860) VB-1.dll:DLL_THREAD_DETACH
The thread 'Win32 Thread' (0xf14) has exited with code 0 (0x0).
(2680) VB-1.dll:DLL_THREAD_DETACH
The thread 'Win32 Thread' (0xa78) has exited with code 0 (0x0).
(2832) VB-1.dll:DLL_THREAD_DETACH
The thread 'DirectSoundCreate' (0xb10) has exited with code 0 (0x0).
(1004) VB-1.dll:effEditClose
(1004) VB-1.dll:effClose
(1004) VB-1.dll:Unloaded C:\Program Files\Steinberg\Vstplugins\VB-1.dll
(1004) VB-1.dll:DLL_PROCESS_DETACH
'PluginConsultant.exe': Unloaded 'D:\Dev\SoundConsultant\Release\PluginConsultant.dll'
'PluginConsultant.exe': Unloaded 'C:\XP\system32\comdlg32.dll'
'PluginConsultant.exe': Unloaded 'C:\XP\system32\msvfw32.dll'
'PluginConsultant.exe': Unloaded 'C:\Program Files\Steinberg\Vstplugins\VB-1.dll'
```

Using Multiple Instances

Plugin Consultant does not allow itself to be loaded more than one time in the same process. Running Plugin Consultant in multiple processes in the same time can also be a problem since all the preferences are saved in an XML file when Plugin Consultant is closed so you won't be able to keep the preferences for the different logging sessions.

To overcome those issues, you can easily create multiple instances of Plugin Consultant by copying the PluginConsultant.DLL file to new files with different names, for example PluginConsultantMySynth.DLL and PluginConsultantMyEffect.DLL. Since the XML preferences file name is based on the DLL name, each instance will create its own XML file so the preferences will be saved per instance, for example PluginConsultantMySynth.XML and PluginConsultantMyEffect.XML.

There are several benefits for using this method to create multiple instances of Plugin Consultant:

- You can log multiple VST plugins in the same host and in the same time
- You can have kind of 'presets' of different preferences for different type of logging sessions (logging only editor stuff, only audio stuff, etc.)
- You can avoid the need to perform full plugin re-scan in hosts like Cubase each time you want to log another plugin, just prepare multiple instances, one for each plugin, and perform only one plugin scan to set things up

Performance Issues

Plugin Consultant does not take a lot of CPU when monitoring the VST API since it does not perform almost any logic for each call and the GUI is running in another thread and updated using a timer. You should not experience any performance degradation or audio problem when you only monitor the API (i.e. all logs are turned off).

However, logging can take a lot of CPU, from causing some audio problems to a complete freeze of a system. Most of the CPU is taken by the log output viewer and not by Plugin Consultant but even without an active debug output viewer there might be some issues with the audio quality.

In order to reduce the CPU hit you can use Plugin Consultant GUI and turn off logging for frequently called functions and opcodes that you do not need to log. Frequently called functions and opcodes include process, processReplacing, ProcessEvents, EditIdle and others. Since different hosts and plugins can frequently call additional functions or opcodes, you can use Plugin Consultant to understand which are these by seeing the counters for these advance quickly. Note that audioMaster and dispatcher counters might advance quickly but since these are dispatching function you can go to the relevant page and turn logging off for the relevant opcodes and not for all plugin or host opcodes.

If you must log a frequently called function like processReplacing you can try and add to your sound card buffer size since many hosts use this buffer size as the buffer size for the audio and midi calls. For example if the buffer size is 2ms then the processReplacing function will be called 500 times in one second but if you make it 100ms then it will be called only 10 times in one second. You can use larger buffers when debugging and later restore to normal latency when you turn off the logging for the audio and midi functions and opcodes.

In case the host or the system freezes because of too much CPU taken by Plugin Consultant logging and you cannot even access the Plugin Consultant GUI, try to edit the PluginConsultant.XML file with a text editor and turn off the offending function or function log. Save the XML file and the next time you start Plugin Consultant, it will use the changed preferences from the XML file. The XML file format is self explanatory.