VBVMR-API

Voicemeeter Remote API

Documentation

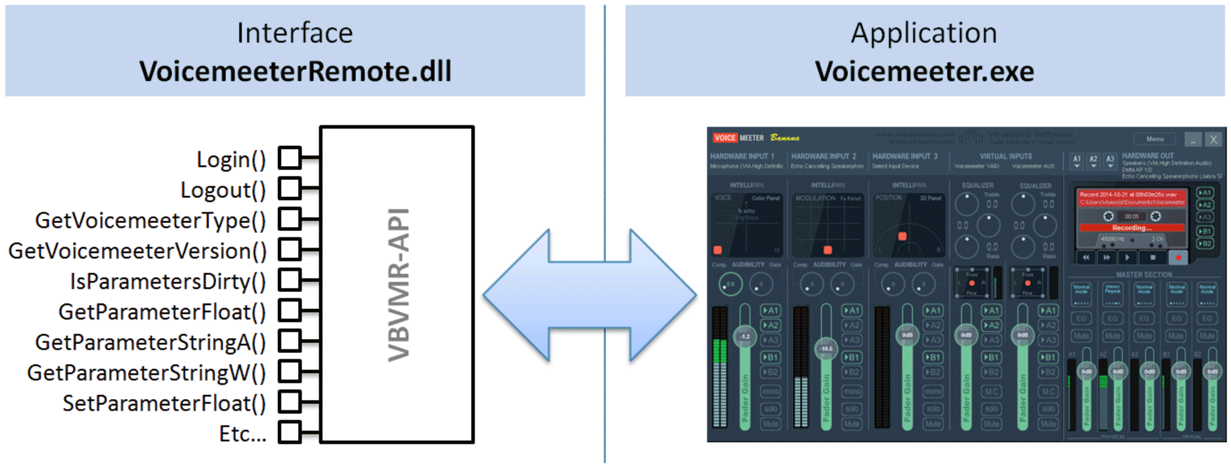
OFFICIAL WEBSITE

www.voicemeeter.com

Table of Content

INTRODUCTION:

VoicemeeterRemote API is based on a standard DLL providing all functions to control Voicemeeter Audio Engine by any user application programmed in any language.



Up to 4 Clients Applications, to remote Voicemeeter or Banana version

Installation:

To make the Voicemeeter Remote API work, Voicemeeter Application must be installed and must run. VoicemeeterRemote.dll must be linked to the user application willing to control Voicemeeter (Download and user manual on www.voicemeeter.com).

Package Content:

Voicemeeter Remote API includes the following files:

- VoicemeeterRemote.h (Reference ‘C’ Header file)
- VoicemeeterRemoteAPI.pdf (this documentation file)
- ReadMe.txt

- Example0: Win32 client Application example (in ‘C’) showing the different communication mechanisms.
- Matrix 8x8 example of Audio Processing application.
- VMR_OSD: example of basic OSD
- Script: different text file changing different group of parameters.

VoicemeeterRemote.dll’s are installed with Voicemeeter (since package 2.0.2.3)

Licensing:

VoicemeeterRemote API is free to use in any kind of project: for testing or educational purpose. for personal use or collaborative work. Open Source project can also use, modify and diffuse any source code provided in Voicemeeter Remote SDK.

Since March 2021 it is also free to use the Voicemeeter Remote API in professional or commercial project. Then any developer or software manufacturer can develop their own application / solution to control and use Voicemeeter Audio Engine.

This license does not include Voicemeeter license. Distributed as donationware for End User, Voicemeeter License must be paid for any professional or commercial use. To Bundle or integrate Voicemeeter in a specific solution, thanks to contact us by e-mail:

https://vb-audio.com/Services/contact.htm

API general policy

All functions are following the __stdcall convention returning 32bit integer as result (if above zero) or error code (if below zero). 0 means OK or simply ZERO.

If not mentioned, all functions are thread safe and non-blocking (or asynchronous) – means they gives the result right after the call.

Function can be used in any language supporting __stdcall function and the following data types:

```
long (32 bits integer).
char *  (pointer on ANSI String).
WCHAR * or unsigned short * (pointer on UNICODE String).
float (32 bit float).
```

Link to Voicemeeter DLL:

VoicemeeterRemote.dll is installed with Voicemeeter package and it is highly recommended to link your application to this DLL to warrant the compatibility in the time. The DLL installed with Voicemeeter is made 100% compatible with the current Voicemeeter Version, while using an old version or different version of the DLL could not work or produce trouble.

Here it is below a part of the source code example, showing how to link to the VoicemeeterRemote.dll installed with the current version of Voicemeeter:

```
//get folder where is installed Voicemeeter
if (RegistryGetVoicemeeterFolder(szDllName) == FALSE)
{
   // voicemeeter not installed
   return -100;
}
//use right DLL according O/S type
if (sizeof(void*) == 8) strcat(szDllName,"\\VoicemeeterRemote64.dll");
else strcat(szDllName,"\\VoicemeeterRemote.dll");

// Load Dll
G_H_Module=LoadLibrary(szDllName);
if (G_H_Module == NULL) return -101;
// Get function pointers
iVMR.VBVMR_Login=(T_VBVMR_Login)GetProcAddress(G_H_Module,"VBVMR_Login");
iVMR.VBVMR_Logout=(T_VBVMR_Logout)GetProcAddress(G_H_Module,"VBVMR_Logout");
```

etc…

See detail of the following function is in the **vmr_client.c** code example.

```
BOOL __cdecl RegistryGetVoicemeeterFolder(char * szDir)
```

Login / Logout:

Communication with Voicemeeter Audio Engine is made through a client/server protocol. The client application must login first (it's a kind of user registration to the voicemeeter remote service) and logout at the end, to release the remote service.

Login / logout must be called once, at the beginning and end of the client application. The login does not depend on Voicemeeter status, if it's running or not, and Voicemeeter application can be shutdown and restarted during the same login. Again Login function must be called once by the user application, for all its duration life.

```
rep=iVMR.VBVMR_Login();
if (rep < 0)
{
	MessageBox(hw,"Failed To Login",”Unexpected Error”,MB_APPLMODAL | MB_OK | MB_ICONERROR);
	return FALSE;
}
```

The Login must work anyway (with Voicemeeter launched or not). Then you can launch a Voicemeeter version just after:

```
if (rep == 1)
{
	fConnected = TRUE;
	iVMR.VBVMR_RunVoicemeeter(2);
	Sleep(1000);
}
```

Call IsParameterDirty just after to initialize/Update parameters

```
ComError = iVMR.VBVMR_IsParametersDirty();

ComError = iVMR.VBVMR_MacroButton_IsDirty();//for MacroButtons communication only.
```

The returned code can be used to detect a reconnection / disconnection

```
if (ComError >= 0
{
	If (fConnected == FALSE)
	{
		fConnected=TRUE; //connection ok
	}
}
else
{
	if (fConnected == TRUE)
	{
		fConnected= FALSE; //disconnection
	}
}
```

IMPORTANT REM: Call `VBVMR_Logout()`once at the end of the program.

Special Commands:

Special Command are not made to change parameter but to make an action. Here some command presented in the menu (write only).

Command Name	Value Range	Remark	Ver.
`Command.Shutdown`	1	Shutdown Voicemeeter	1
`Command.Show`	1	Show Voicemeeter	1
`Command.Restart`	1	Restart Audio Engine	1
`Command.Eject`	1	Eject Cassette	1
`Command.Reset`	1	Reset All configuration	1
`Command.Save`	String	Complete filename (xml)	1
`Command.Load`	String	Complete filename (xml)	1
`Command.Button[i].State`	0 or 1	Change Macro Button State	1
`Command.Button[i].StateOnly`	0 or 1	Change Button State only	1
`Command.Button[i].Trigger`	0 or 1	Change Trigger Enable State	1
`Command.DialogShow.VBANCHAT`	0 or 1	Show VBAN-Chat Dialog	1

i= MacroButton ID (zero based index).

Typical use:.

```
Command.Restart = 1;// request to restart audio engine

Command.Load= “C:\My Documents\VMConfig1.xml”; // load Config File
```

NOTE: command request are proprietary to other requests. It means other type of request could not be processed if in the same request packet than a command request. For example Shutdown Request, simply close the program without processing next request. LOAD request reset all possible previous or next request present in the same packet.

MacroButtons function:

Coming with version 3.0.1.4 / 2.0.5.4 / 1.0.7.4 it’s now possible to remote some basic MacroButtons functions to PUSH or RELEASE a given button, change the displayed state or the trigger enable status. This is given by basically 2x functions:

```
long __stdcall VBVMR_MacroButton_GetStatus(long nuLogicalButton, float * pValue, long bitmode);

long __stdcall VBVMR_MacroButton_SetStatus(long nuLogicalButton, float fValue, long bitmode);
```

And the synchronization function to know if there has been a change in MacroButtons. See next page to get more information.

```
long __stdcall VBVMR_MacroButton_IsDirty(void);
```

Get/Set parameters:

Parameters management can be done easily by few functions, like Get and Set parameters:

```
long __stdcall VBVMR_GetParameterFloat(char * szParamName, float * pValue);
long __stdcall VBVMR_SetParameterFloat(char * szParamName, float Value);
```

where the szParamName is a simple string naming the data (see parameters list below).

Synchronization:

To know when parameters have been changed on Voicemeeter user interface or by another client application connected to Voicemeeter, you have to periodically call the following function:

```
long __stdcall VBVMR_IsParametersDirty(void);
```

This is returning 1 if one or more parameters have been changed.

Typical usage is to call this function in a timer event or in a polling thread to know when to update parameters display.

```
fDisplayParam=iVMR.VBVMR_IsParametersDirty();
if (fDisplayParam!= 0) UpdateVisibleParameter(…);
```

EXAMPLE OF SYNCHRONIZATION METHOD:

INITIALIZATION

```
	//get DLL interface
	rep=InitializeDLLInterfaces();
	if (rep < 0)
	{
		if (rep == -100) MessageBox(hw,"Voicemeeter is not 
installed",szTitle,MB_APPLMODAL | MB_OK | MB_ICONERROR);
		else MessageBox(hw,"Failed To Link To 
VoicemeeterRemote.dll",szTitle,MB_APPLMODAL | MB_OK | MB_ICONERROR);
		return FALSE;
	}
	//Log in
	rep=iVMR.VBVMR_Login();
	if (rep < 0)
	{
		MessageBox(hw,"Failed To Login",szTitle,MB_APPLMODAL | MB_OK | 
MB_ICONERROR);
		return FALSE;
	}
	//call this to get first parameters state (if server already launched)
	lpapp->vbvmr_error = iVMR.VBVMR_IsParametersDirty();
	if (lpapp->vbvmr_error == 0)
	{
		rep=iVMR.VBVMR_GetVoicemeeterType(&vmType);
		if (rep == 0) lpapp->vbvmr_connect =vmType;
	}
	else lpapp->vbvmr_connect=0;
```

WM_TIMER PROCESS:

```
	lpapp->vbvmr_error=iVMR.VBVMR_IsParametersDirty();
	if (lpapp->vbvmr_error >= 0)
	{
		if (lpapp->vbvmr_connect == 0)
		{
			rep=iVMR.VBVMR_GetVoicemeeterType(&vmType);
			if (rep == 0) lpapp->vbvmr_connect =vmType;
			InvalidateRect(hw,NULL,TRUE);
		}
	}
	else
	{
		if (lpapp->vbvmr_connect != 0)
		{
			//Voicemeeter has been shut down
			lpapp->vbvmr_connect = 0;
			InvalidateRect(hw,NULL,TRUE);
		}
	}
```

Synchronization for MacroButtons Application:

To know when a MacroButtons state have been changed you have to periodically call the following function:

```
long __stdcall VBVMR_MacroButton_IsDirty(void);
```

This is returning 1 if one or more Button states have been changed.

Typical usage is to call this function in a timer event or in a polling thread to know when to update parameters display.

```
fDisplayButtonState =iVMR.VBVMR_MacroButton_IsDirty ();
if (fDisplayButtonState!= 0) UpdateVisibleStuff(…);
```

Real Time Level Meter:

Like for the synchronization, the **GetLevel** function has to be called periodically to get the current level in Voicemeeter.

Example to know the 22 input levels of Voicemeeter Banana Version:

```
	for (vi=0;vi<22;vi++)
	{
		NormalLevel=0.0f;
		iVMR.VBVMR_GetLevel(0, vi, &NormalLevel);
		DisplayLevel(dc, NormalLevel);
	}
```

See VoicemeeterRemote.h header file to know more about GetLevel function

Real Timer M.I.D.I. Messages:

In the same synchronization loop, it is also possible to call the **VBVMR_GetMidiMessage** function to get M.I.D.I. message incoming in the M.I.D.I. Input device selected in the Voicemeeter M.I.D.I. Mapping Dialog Box.

Parameters List

Parameters are given by a structured name, given in a simple ANSI string. VoicemeeterRemote.dll provides function to get or set a single parameters, and one function to set several parameters by a simple text script.

Input Strip Parameters:

Strip index is a zero based index related to Voicemeeter version (3 strips on Voicemeeter, 5 on Voicemeeter Banana, 8 on Voicemeeter Potato)

Strip functions/parameters

Parameter Name	Value Range	Remark	Ver.
`Strip[i].Mono`	0 (off) or 1 (on)	Mono Button	1
`Strip[i].Mute`	0 (off) or 1 (on)	Mute Button	1
`Strip[i].Solo`	0 (off) or 1 (on)	Solo Button	1
`Strip[i].MC`	0 (off) or 1 (on)	Mute Center Button	1
`Strip[i].Gain`	-60 to +12 dB	Gain slider	1
`Strip[i].GainLayer[j]`	-60 to +12 dB	Gain slider for a bus	3
`Strip[i].Pan_x`	-0.5 to +0.5		1
`Strip[i].Pan_y`	0 to 1.0	-0.5 to 0.5 for 5.1 pan pot	1
`Strip[i].Color_x`	-0.5 to +0.5	Physical Strip Only	1
`Strip[i].Color_y`	0 to 1.0	Physical Strip Only	1
`Strip[i].fx_x`	-0.5 to +0.5	Physical Strip Only	2
`Strip[i].fx_y`	0 to 1.0	Physical Strip Only	2
`Strip[i].Audibility`	0 to 10	Voicemeeter 1 only	1
`Strip[i].Comp`	0 to 10		2
`Strip[i].Gate`	0 to 10		2
`Strip[i].Karaoke`	0,1, 2, 3,4	OFF or one of 4 karaoke algorithm	2
`Strip[i].Limit`	-40 to +12 dB		2
`Strip[i].EQGain1`	-12 to +12 db	Virtual Strip Only	1
`Strip[i].EQGain2`	-12 to +12 db	Virtual Strip Only	1
`Strip[i].EQGain3`	-12 to +12 db	Virtual Strip Only	1
`Strip[i].Label`	String	Strip Label	1
`Strip[i].A1`	0 (off) or 1 (on)	Out BUS Assignation	1
`Strip[i].A2`	0 (off) or 1 (on)	Out BUS Assignation	2
`Strip[i].A3`	0 (off) or 1 (on)	Out BUS Assignation	2
`Strip[i].A4`	0 (off) or 1 (on)	Out BUS Assignation	3
`Strip[i].A5`	0 (off) or 1 (on)	Out BUS Assignation	3
`Strip[i].B1`	0 (off) or 1 (on)	Out BUS Assignation	1
`Strip[i].B2`	0 (off) or 1 (on)	Out BUS Assignation	2
`Strip[i].B3`	0 (off) or 1 (on)	Out BUS Assignation	3
`Strip[i].FadeTo`	String	(dBTarget, msTime);	1
`Strip[i].FadeBy`	String	(dB relativechange, msTime);	1
`Strip[i].Reverb`	0 to 10	Send Level To Reverb	3
`Strip[i].Delay`	0 to 10	Send Level To Delay	3
`Strip[i].Fx1`	0 to 10	Send Level To External Fx1	3
`Strip[i].Fx2`	0 to 10	Send Level To External Fx2	3
`Strip[i].PostReverb`	0 (off) or 1 (on)	Post Reverb button	3
`Strip[i].PostDelay`	0 (off) or 1 (on)	Post Delay button	3
`Strip[i].PostFx1`	0 (off) or 1 (on)	Post Fx1 button	3
`Strip[i].PostFx2`	0 (off) or 1 (on)	Post Fx2 button	3

i= strip zero based index. J= Bus zero based index.

Additionally, it is possible to control Applications Gain / Mute connected to virtual input strips (Potato version only – Write Only): If using AppName (Application Name) all applications beginnings by the given string will be changed.

`Strip[i].App[k].Gain`	0.0 to 1.0	Application gain	3
`Strip[i].App[k].Mute`	0 (off) or 1 (on)	Application Mute	3
`Strip[i].AppGain`	String	(AppName, Gain)	3
`Strip[i].AppMute`	String	(AppName, MuteState)	3

i= strip zero based index. k= app zero based index.

Example of AppGain & AppMute:

```
Strip(5).AppGain= ("Skype", 0.5); //gain is in the range [0.0 to 1.0]
Strip(5).AppMute= ("Spotify", 1); //Mute Spotify if exist
```

Strip Audio Devices (physical strip only)

Parameter Name	Value Range	Remark	Ver.
`Strip[i].device.name`	String	Read only	1
`Strip[i].device.sr`	samplerate	Read Only	1
`Strip[i].device.wdm`	Device Name	Write only	1
`Strip[i].device.ks`	Device Name	Write only	1
`Strip[i].device.mme`	Device Name	Write only	1
`Strip[i].device.asio`	Device Name	Write only	1

i= strip zero based index.

BUS Parameters:

Bus index is a zero based index related to Voicemeeter version (2 busses on Voicemeeter, 5 on Voicemeeter Banana)

Bus functions/parameters

Parameter Name	Value Range	Remark	Ver.
`Bus[i].Mono`	0 (off), 1 (mono) 2 (stereo reverse)	Mono Button	1
`Bus[i].Mute`	0 (off) or 1 (on)	Mute Button	1
`Bus[i].EQ.on`	0 (off) or 1 (on)	EQ Button	2
`Bus[i].EQ.AB`	0 (A) or 1 (B)	EQ Memory Slot	2
`Bus[i].Gain`	-60 to +12 db	Gain slider	1
`Bus[i].mode.normal`	0 (off) or 1 (on)	BUS Mode	1
`Bus[i].mode.Amix`	0 (off) or 1 (on)	BUS Mode	1
`Bus[i].mode.Bmix`	0 (off) or 1 (on)	BUS Mode	2
`Bus[i].mode.Repeat`	0 (off) or 1 (on)	BUS Mode	1
`Bus[i].mode.Composite`	0 (off) or 1 (on)	BUS Mode	1
`Bus[i].mode.TVMix`	0 (off) or 1 (on)	BUS Mode	2
`Bus[i].mode.UpMix21`	0 (off) or 1 (on)	BUS Mode	2
`Bus[i].mode.UpMix41`	0 (off) or 1 (on)	BUS Mode	2
`Bus[i].mode.UpMix61`	0 (off) or 1 (on)	BUS Mode	2
`Bus[i].mode.CenterOnly`	0 (off) or 1 (on)	BUS Mode	2
`Bus[i].mode.LFEOnly`	0 (off) or 1 (on)	BUS Mode	2
`Bus[i].mode.RearOnly`	0 (off) or 1 (on)	BUS Mode	2
`Bus[i].EQ.channel[j].cell[k].on`	0 (off) or 1 (on)	EQ cell On/Off	2

Bus[i].EQ.channel[j].cell[k].type	0 to 6	EQ type of the Cell	2
Bus[i].EQ.channel[j].cell[k].f	20 to 20.000 Hz	Cell Frequency	2
Bus[i].EQ.channel[j].cell[k].gain	-12 to +12 db	Cell Gain	2
Bus[i].EQ.channel[j].cell[k].q	1 to 100	Cell Quality	2
Bus[i].FadeTo	String	(dBTarget, msTime);	1
Bus[i].FadeBy	String	(dB change,msTime);	1
Bus[i].Sel	0 (off) or 1 (on)	BUS SEL Button	3
Bus[i].ReturnReverb	0 to 10	Reverb return	3
Bus[i].ReturnDelay	0 to 10	Delay return	3
Bus[i].ReturnFx1	0 to 10	FX1 return	3
Bus[i].ReturnFx2	0 to 10	FX2 return	3

i= bus zero based index, j=channel zero based index (0 to 7), k=cell zero based index (0 to 5).

REM: Strip().FadeTo or Bus().FadeTo function allow to set the gain slider with a progressive fade by settings a dB value and a time to reach it (time in ms between 0 and 120.000). The parameter is a string since it needs 2 parameters, example:

```
Strip(0).FateTo= (-10.0, 500); //will set the slider to -10 dB in 500ms
Strip(0).FateTo= (-20.0, 2000); //will set the slider to -10 dB in 2 seconds
Bus(0).FateTo= (0.0, 1500); //will set the bus slider to 0 dB in 1,5 seconds
```

Same remark is valid for the instruction FadeBy

BUS Audio Devices (physical bus only)

Parameter Name	Value Range	Remark	Ver.
Bus[i].device.name	String	Read only	1
Bus[i].device.sr	samplerate	Read Only	1
Bus[i].device.wdm	Device Name	Write only	1
Bus[i].device.ks	Device Name	Write only	1
Bus[i].device.mme	Device Name	Write only	1
Bus[i].device.asio	Device Name	Write only	1

i= bus zero based index.

FX Settings:

Voicemeeter Potato Internal FX button can also be controlled by the following instructions.

Internal FX

Parameter Name	Value Range	Remark	Ver.
Fx.Reverb.On	0 (off) or 1 (on)	Switch On/Off	3
Fx.Delay.On	0 (off) or 1 (on)	Switch On/Off	3

System Settings Option:

Voicemeeter remote API also allows changing different configuration parameters: Patch and System Settings.

Patch Options

Parameter Name	Value Range	Remark	Ver.
patch.asio[i]	0 to ASIO input	ASIO Patch	1

`Patch.composite[j]`	0 to 22 (1 = first channel)	0 = default BUS	2
`Patch.insert[k]`	0 (off) or 1 (on)	Virtual ASIO insert	2
`Patch.PostFaderComposite`	0 (PRE) or 1 (POST)	COMPOSITE Mode	2
`Patch.PostFxInsert`	0 (PRE) or 1 (POST)	Virtual INSERT Point	2

i= input channel zero based index (for physical strips only – 2 channels per strip).
j= composite channel zero based index (0 to 7) COMPOSITE mode is maed of 8 channels.
k= input channel zero based index (0 to 21).

System Settings

Parameter Name	**Value Range**	**Remark**	**Ver.**
`Option.sr`	44.1, 48, 88.2, 96, 176.4 or 192 kHz	Preferred samplerate	1
`Option.ASIOsr`	0: default ASIO Samplerate 1: preferred samplerate.	For ASIO driver connected on output A1	1
`Option.delay[i]`	0 to 500ms max	BUS output delay	1
`Option.buffer.mme`	128 to 2048	MME buffer size	1
`Option.buffer.wdm`	128 to 2048	WDM buffer size	1
`Option.buffer.ks`	128 to 2048	KS buffer size	1
`Option.buffer.asio`	128 to 2048 0: default ASIO buffer size	ASIO Buffer Size	1
`Option.mode.exclusif`	0 (off) or 1 (on)	WDM input exclusive	1
`Option.mode.swift`	0 (off) or 1 (on)	WDM swift mode	1

i= output zero based index (for physical bus only)

Tape Recorder Options:

Voicemeeter remote API allows controlling the integrated recorder in Voicemeeter Banana.

Recorder Options

Parameter Name	Value Range	Remark	Ver.
`recorder.stop`	0 (off) or 1 (on)	STOP	2
`recorder.play`	0 (off) or 1 (on)	PLAY	2
`recorder.replay`	1	PLAY FROM 0	2
`recorder.ff`	0 (off) or 1 (on)	Fast Forward button	2
`recorder.rew`	0 (off) or 1 (on)	Reward	2
`Recorder.goto`	00:00:00 (hh:mm:ss)	position in seconds	2
`recorder.A1`	0 (off) or 1 (on)	Out BUS Assignation	2
`recorder.A2`	0 (off) or 1 (on)	Out BUS Assignation	2
`recorder.A3`	0 (off) or 1 (on)	Out BUS Assignation	2
`recorder.A4`	0 (off) or 1 (on)	Out BUS Assignation	3
`recorder.A5`	0 (off) or 1 (on)	Out BUS Assignation	3
`recorder.B1`	0 (off) or 1 (on)	Out BUS Assignation	2
`recorder.B2`	0 (off) or 1 (on)	Out BUS Assignation	2
`recorder.B3`	0 (off) or 1 (on)	Out BUS Assignation	3
`recorder.record`	0 (off) or 1 (on)	REC	2
`recorder.pause`	0 (off) or 1 (on)	PAUSE	2
`Recorder.load`	File name to playback	Write only	2
`Recorder.samplerate`			2
`Recorder.ArmStrip(i)`	Arming Input Status		2
`Recorder.ArmBus(i)`	Arming BUS Status		2
`Recorder.mode.recbus`	0 (off) or 1 (on)	0 to record inputs	2
`Recorder.mode.PlayOnLoad`	0 (off) or 1 (on)		2
`Recorder.mode.Loop`	0 (off) or 1 (on)	Loop Mode	2
`Recorder.mode.MultiTrack`	0 (off) or 1 (on)	MultiTrack mode	2
`Recorder.bitResolution`	8, 16, 24, 32	32 is float type	2
`Recorder.Channel`	1 - 8		2
`Recorder.kbps`	For mp3 format		2
`Recorder.FileType`	1 = WAV, 2 = AIFF, 3=BWF 100 = MP3		2
`Recorder.gain`	-60 to +12 db	Playback Gain	2

i= zero based index (0 to 8).

VBAN Options:

Voicemeeter remote API allows controlling VBAN features and all parameters presented on the VBAN dialog Box. Then it is possible to remote VBAN functions to route/send/receive audio to/from different computers.

Recorder Options

Parameter Name	Value Range	Remark	Ver.
`vban.Enable`	0 (off) or 1 (on)	VBAN functions	1
`vban.instream[i].on`	0 (off) or 1 (on)	Stream On/Off	1
`vban.instream[i].name`	String	Stream Name	1
`vban.instream[i].ip`	String	IP Address from	1
`vban.instream[i].port`	16 bit range	PORT (Ethernet)	1
`vban.instream[i].sr`	11025 to 96 kHz	Read only	1
`vban.instream[i].channel`	1 to 8	Read only	1
`vban.instream[i].bit`	VBAN data type	Read only	1
`vban.instream[i].quality`	0 to 4	0 = Optimal	1
`vban.instream[i].route`	0 to 8	Strip Selector	1
`vban.outstream[i].on`	0 (off) or 1 (on)	Stream On/Off	1
`vban.outstream[i].name`	String	Stream Name	1
`vban.outstream[i].ip`	String	IP Address To	1
`vban.outstream[i].port`	16 bit range	PORT (Ethernet)	1
`vban.outstream[i].sr`	11025 to 96 kHz		1
`vban.outstream[i].channel`	1 to 8		1
`vban.outstream[i].bit`	VBAN data type	1 = 16 bits PCM	1
`vban.outstream[i].quality`	0 to 4	0 = Optimal	1
`vban.outstream[i].route`	0 to 8	BUS selector	1

i= zero based index (0 to 7).

Like in the VBAN Dialog Box the following parameters change are producing an Audio Engine Restart:

- `vban.Enable`
- `vban.instream[i].port`
- `vban.instream[i].quality`
- `vban.outstream[i].quality`

VBAN SampleRate:
11025, 16000, 22050, 24000, 32000, 44100, 48000, 64000, 88200, 96000 Hz

VBAN Quality:
0 (Optimal), 1 (Fast), 2 (Medium), 3 (Slow), 4 (very slow). Quality parameter is conditioning the size of internal stack (also the latency) to cope to possible network instability and then increase the stream stability if required. **Optimal** quality considers the network is able to transmit packet in real time (with a good regularity), while **very slow** quality considers the network can have timing problem and unexpected waiting cycles.

VBAN Bit Resolution / data format:
Allowed Format are 1 (16 bit PCM) or 2 (24 bits PCM).

Relative parameters:

Some parameters can be change by relative move with the syntax += or -= to add or subtract a value to the current parameter

Typically for gain (VOL+ and VOL- command)

```
Strip[i].Gain +=3 ; //add 3 dB to current gain
Strip[i].Gain -=3 ; //remove 3 dB to current gain
```

Typically for switch

```
Strip[i].Mono +=1 ; //switch the current mono state.
Strip[i].A1 +=1 //switch the A1 current assignation state.
```

Audio Device Enumeration

Voicemeeter Remote Dll provides function to enumerate audio device in order to select it on Voicemeeter physical input strip and physical output busses.

Audio device inputs:

To enumerate audio device, we call first the _GetDeviceNumber() function to know the number of audio device detected on the system. Then we call for all indexes the function _GetDeviceDesc() that returns the type (MME, WDM, KS), the public device name and driver identifier (szHardwareId).

Typical example:

```
//Build Input Device Menu
nb=    iVMR.VBVMR_Input_GetDeviceNumber();
hSubMenu=CreatePopupMenu();
for (vi=0;vi<nb;vi++)
{
       rep = iVMR.VBVMR_Input_GetDeviceDescA(vi,&nType, szName, szHardwareId);
       if (rep == 0)
       {
              switch(nType)
              {
              case VBVMR_DEVTYPE_MME:
                     sprintf(sss,"MME: %s",szName);
                     break;
              case VBVMR_DEVTYPE_WDM:
                     sprintf(sss,"WDM: %s",szName);
                     break;
              case VBVMR_DEVTYPE_KS:
                     sprintf(sss,"KS: %s",szName);
                     break;
              }
              AppendMenu(hSubMenu,MF_STRING,IDM_DEVICE_IN1+vi+1,sss);
       }
}
```

NOTE: szHardwareId can be used to identify precisely a driver. But to select a device in Voicemeeter, it need the public name and the type

Selecting Audio device in Voicemeeter:

Setting Audio Device

Parameter Name	Value Range	Remark	Ver.
`Strip[i].Device.mme`	Device Name as String	Write only	1
`Strip[i].Device.wdm`	Device Name as String	Write only	1
`Strip[i].Device.ks`	Device Name as String	Write only	1
`Bus[j].Device.asio`	Device Name as String	Write only	1
`Bus[j].Device.mme`	Device Name as String	Write only	1
`Bus[j].Device.wdm`	Device Name as String	Write only	1
`Bus[j].Device.ks`	Device Name as String	Write only	1

i= Input Strip zero based index (physical strip only).
j= Output Bus zero based index (physical bus only).

Audio device Outputs:

For output audio devices, the sequence is the same expect that _GetDeviceDesc() returns aalso ASIO device type that can be used on output A1 only..

Typical example:

```
nb=    iVMR.VBVMR_Output_GetDeviceNumber();
hSubMenu=CreatePopupMenu();
for (vi=0;vi<nb;vi++)
{
       rep = iVMR.VBVMR_Output_GetDeviceDescA(vi,&nType, szName, szHardwareId);
       if (rep == 0)
       {
              switch(nType)
              {
              case VBVMR_DEVTYPE_MME:
                     sprintf(sss,"MME: %s",szName);
                     break;
              case VBVMR_DEVTYPE_WDM:
                     sprintf(sss,"WDM: %s",szName);
                     break;
              case VBVMR_DEVTYPE_KS:
                     sprintf(sss,"KS: %s",szName);
                     break;
              case VBVMR_DEVTYPE_ASIO:
                     sprintf(sss,"ASIO: %s",szName);
                     break;
              }
              AppendMenu(hSubMenu,MF_STRING,IDM_DEVICE_OUT1+vi+1,sss);
       }
}
```

VB-Audio Callback

Voicemeeter Remote DLL provides functions to directly process audio signal inside Voicemeeter Virtual Audio Device Mixer. This is made possible by a set of 4 functions allowing you to register your callback function and start/stop the audio stream to your callback.

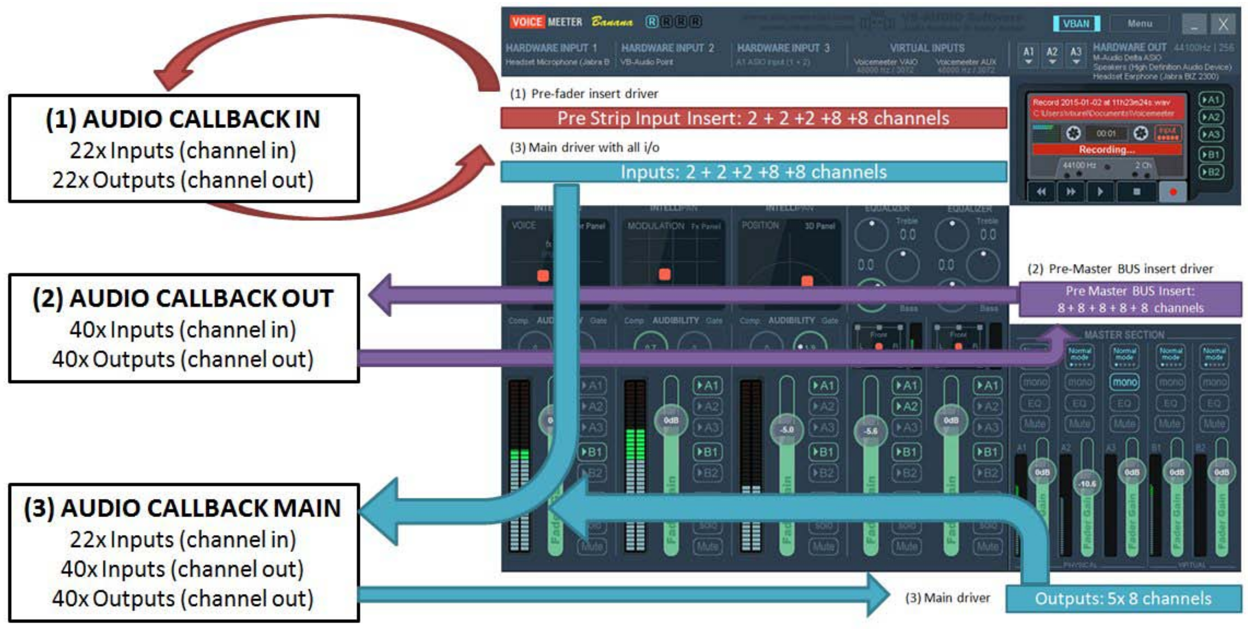
```
VBVMR_AudioCallbackRegister();
VBVMR_AudioCallbackStart();
VBVMR_AudioCallbackStop();
VBVMR_AudioCallbackUnregister();
```

You will find the detail of the implementation in **VoicemeeterRemote.h** (header) and example of code in **vmr_client.c**.

It is possible to register up to 3 callbacks (possibly from 3 different applications) to process audio signal in 3 different points of Voicemeeter:

1- **Voicemeeter Input Insert** (Pre-Strip insert). This is made to be able to process audio at the early input stage where all inputs are just synchronized and ready to be processed in Voicemeeter.
2- **Voicemeeter Output Insert** (Pre Master BUS insert). This is made to be able to process audio outputs before master section.
3- **Voicemeeter Main** (all input and all output). This made to receive all I/O to record or process it and replace or mix to BUS outputs.

The 3 possible callbacks or streams will be called in this order to let you possibly make DSP processing on 3 stages if needed.


(1) AUDIO CALLBACK IN
22x Inputs (channel in)
22x Outputs (channel out)
(2) AUDIO CALLBACK OUT
40x Inputs (channel in)
40x Outputs (channel out)
(3) AUDIO CALLBACK MAIN
22x Inputs (channel in)
40x Inputs (channel out)
40x Outputs (channel out)
(1) Pre-fader insert driver
Pre Strip Input Insert: 2 + 2 +2 +8 +8 channels
(3) Main driver with all i/o
Inputs: 2 + 2 +2 +8 +8 channels
(2) Pre-Master BUS insert driver
Pre Master BUS Insert: 8 + 8 + 8 + 8 + 8 channels
(3) Main driver
Outputs: 5x 8 channels

VB-Audio Callback Buffers:

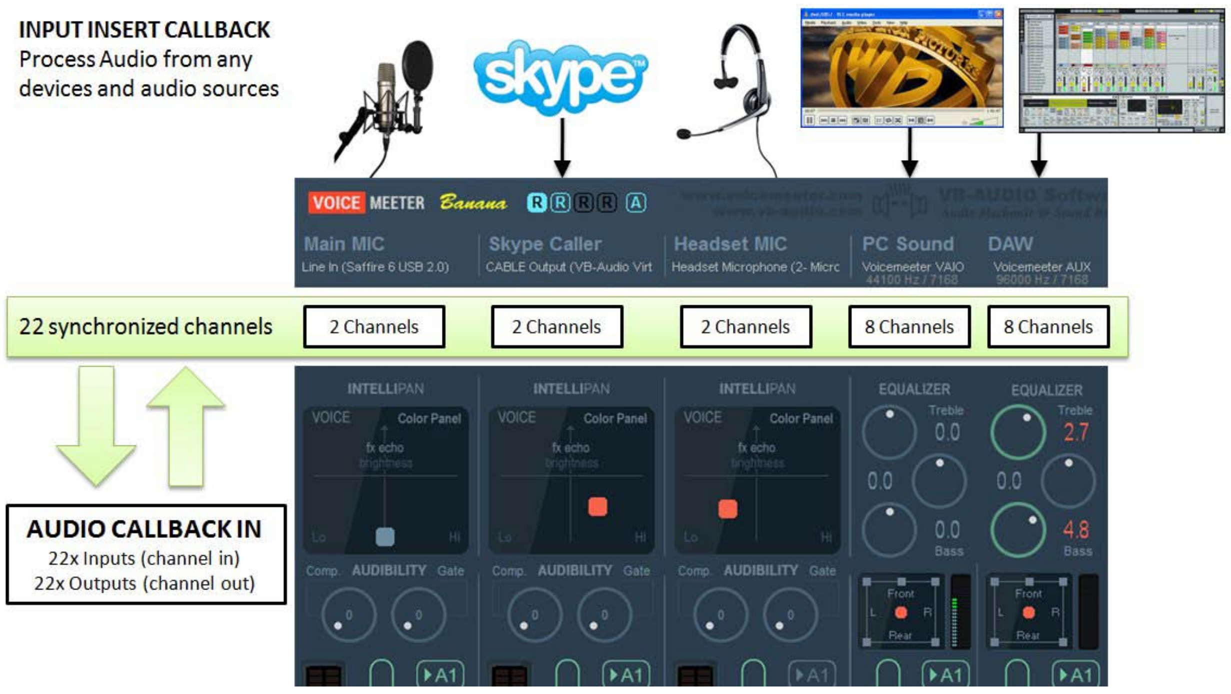
The callback will receive a pointer on the structure below, containing a list of buffer (in 32bits float) related to the number of input channels (read buffer) and output channels (write buffer).

```
typedef struct tagVBVMR_AUDIOBUFFER
{
	long audiobuffer_sr;  //Sampling Rate
	long audiobuffer_nbs; //number of sample per frame
	long audiobuffer_nbi; //number of inputs
	long audiobuffer_nbo; //number of outputs
	float * audiobuffer_r[128];//nbi input pointers containing frame of nbs sample
	float * audiobuffer_w[128];//nbo output pointers containing frame of nbs sample
} VBVMR_T_AUDIOBUFFER, *VBVMR_PT_AUDIOBUFFER, *VBVMR_LPT_AUDIOBUFFER;
```

The maximum number of used channels is fixed and dimensioned to Voicemeeter 8 (version expected for end of year 2016), providing 5x physical strips and 3x virtual strips (it makes 5x 2 + 3x 8 channels) and 5x Physical Busses and 3x Virtual Busses (it makes 5x 8 + 3x 8 channels). Consequently Voicemeeter Input Insert Callback will provide 34 buffers for input and 34 buffers for output, but Banana will use only the 22 firsts, and Voicemeeter Standard version will use only the 12 firsts (see channel organization on next pages or in VoicemeeterRemote.h).

Callback on Voicemeeter inputs:

This callback stream is made to make a PRE-STRIP INSERT processing, typically to change, modify input signal of any Voicemeeter inputs or to record them before any processing (Voicemeeter is then used as monitoring mixing console).



Voicemeeter Input Channel Organization:

Description:	Strip #1	Strip #2	Virtual
Index:	0	2	4
Channels:	2 (Stereo)	2 (Stereo)	8 (7.1)

Voicemeeter Banana Input Channel Organization:

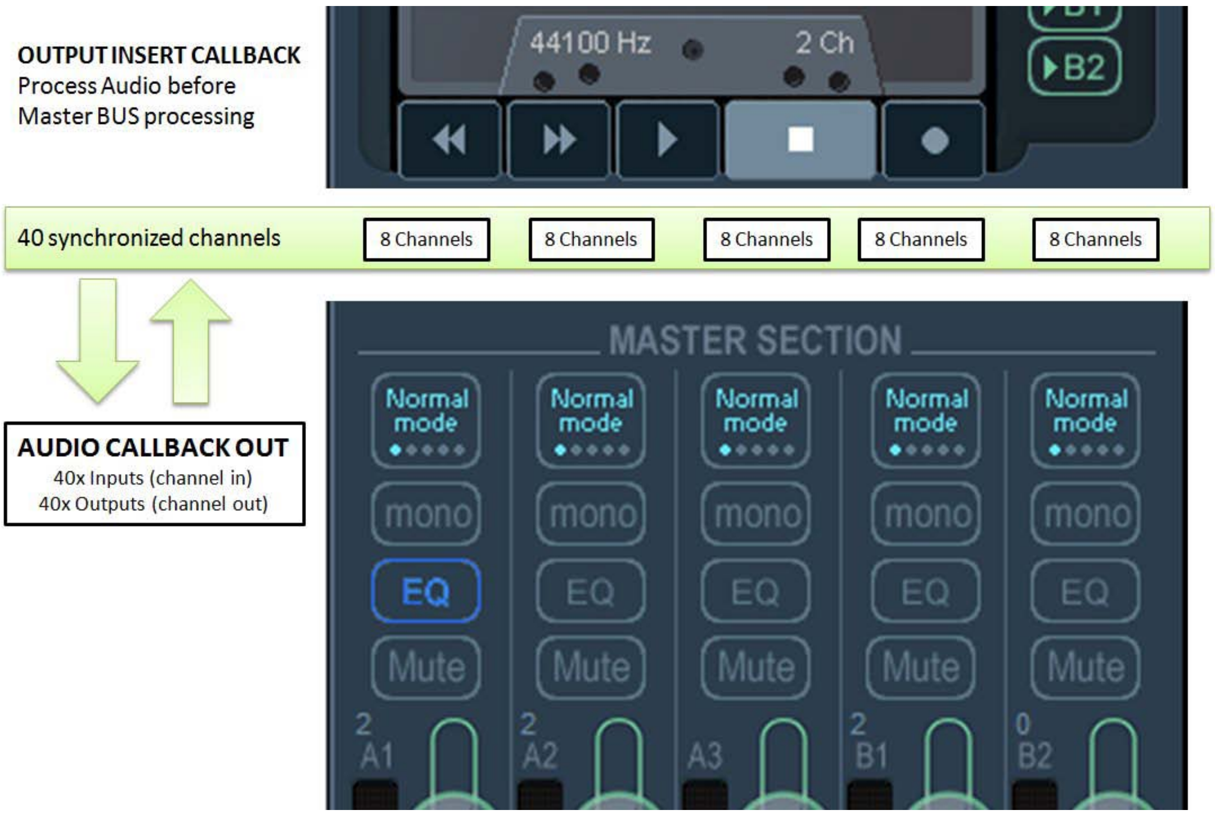
Description:	Strip #1	Strip #2	Strip #3	Virtual	Virtual AUX
Index:	0	2	4	6	14
Channels:	2 (Stereo)	2 (Stereo)	2 (Stereo)	8 (7.1)	8 (7.1)

Voicemeeter Potato Input Channel Organization:

Description:	Strip #1	Strip #2	Strip #3	Strip #4	Strip #5	Virtual	Virtual AUX	VAIO 3
Index:	0	2	4	6	8	10	18	26
Channels:	2	2	2	2	2	8 (7.1)	8 (7.1)	8 (7.1)

Callback on Voicemeeter Outputs:

This Output Insert callback stream is made to make a PRE-MASTER INSERT processing, typically to process output BUS before master section. On Voicemeeter every BUS is made of 8 channels (to manage 5.1 or 7.1 signal).



Voicemeeter Output Channel Organization:

Description:	BUS A	BUS B
Index:	0	8
Channels:	8 (7.1)	8 (7.1)

Voicemeeter Banana Output Channel Organization:

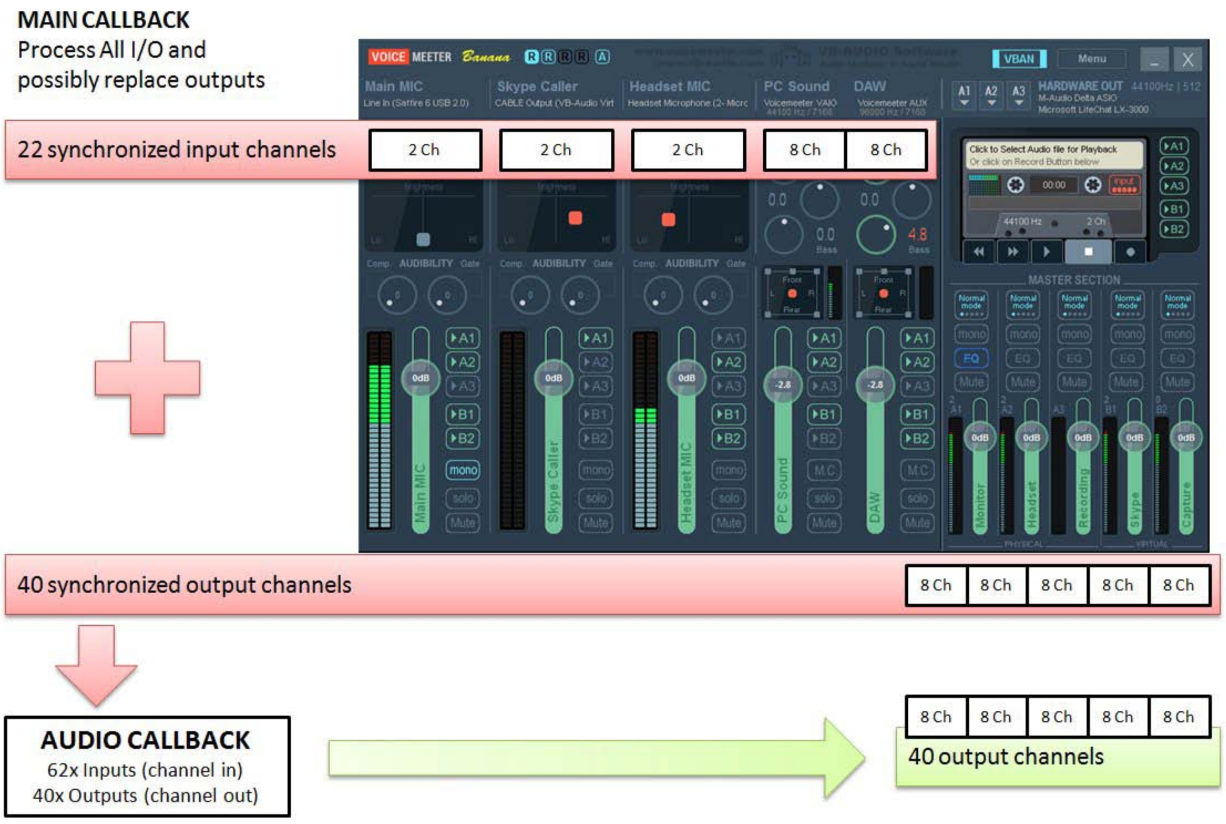
Description:	BUS A1	BUS A2	BUS A3	BUS B1	BUS B2
Index:	0	8	16	24	32
Channels:	8 (7.1)	8 (7.1)	8 (7.1)	8 (7.1)	8 (7.1)

Voicemeeter Potato Output Channel Organization:

Description:	BUS A1	BUS A2	BUS A3	BUS A4	BUS A5	BUS B1	BUS B2	BUS B3
Index:	0	8	16	24	32	40	48	56
Channels:	8 (7.1)	8 (7.1)	8 (7.1)	8 (7.1)	8 (7.1)	8 (7.1)	8 (7.1)	8 (7.1)

Callback on All Voicemeeter I/O's:

The main callback stream receives all inputs (after input INSERT) and all outputs (after master gain, so after output INSERT). It is used to be able to process/record input or bus outputs and possibly change / replace output signal.



Voicemeeter Input Channel Organization:

Description:	Strip #1	Strip #2	Virtual	BUS A	BUS B
Index:	0	2	4	6	14
Channels:	2 (Stereo)	2 (Stereo)	8 (7.1)	8 (7.1)	8 (7.1)

Voicemeeter Banana Input Channel Organization:

Description	In #1	In #2	In #3	V1	V2	A1	A2	A3	B1	B2
Index	0	2	4	6	14	22	30	38	46	54
Channels:	2	2	2	8	8	8	8	8	8	8

Voicemeeter Potato Input Channel Organization:

Description	In #1	In #2	In #3	In #4	In #5	V1	V2	V3	A1	A2	A3	A4	A5	B1	B2	B3
Index	0	2	4	6	8	10	18	26	34	42	50	58	66	74	82	90
Channels:	2	2	2	2	2	8	8	8	8	8	8	8	8	8	8	8

Output channels organization is the same than for Output Callback.

Callback Sample Code Example:

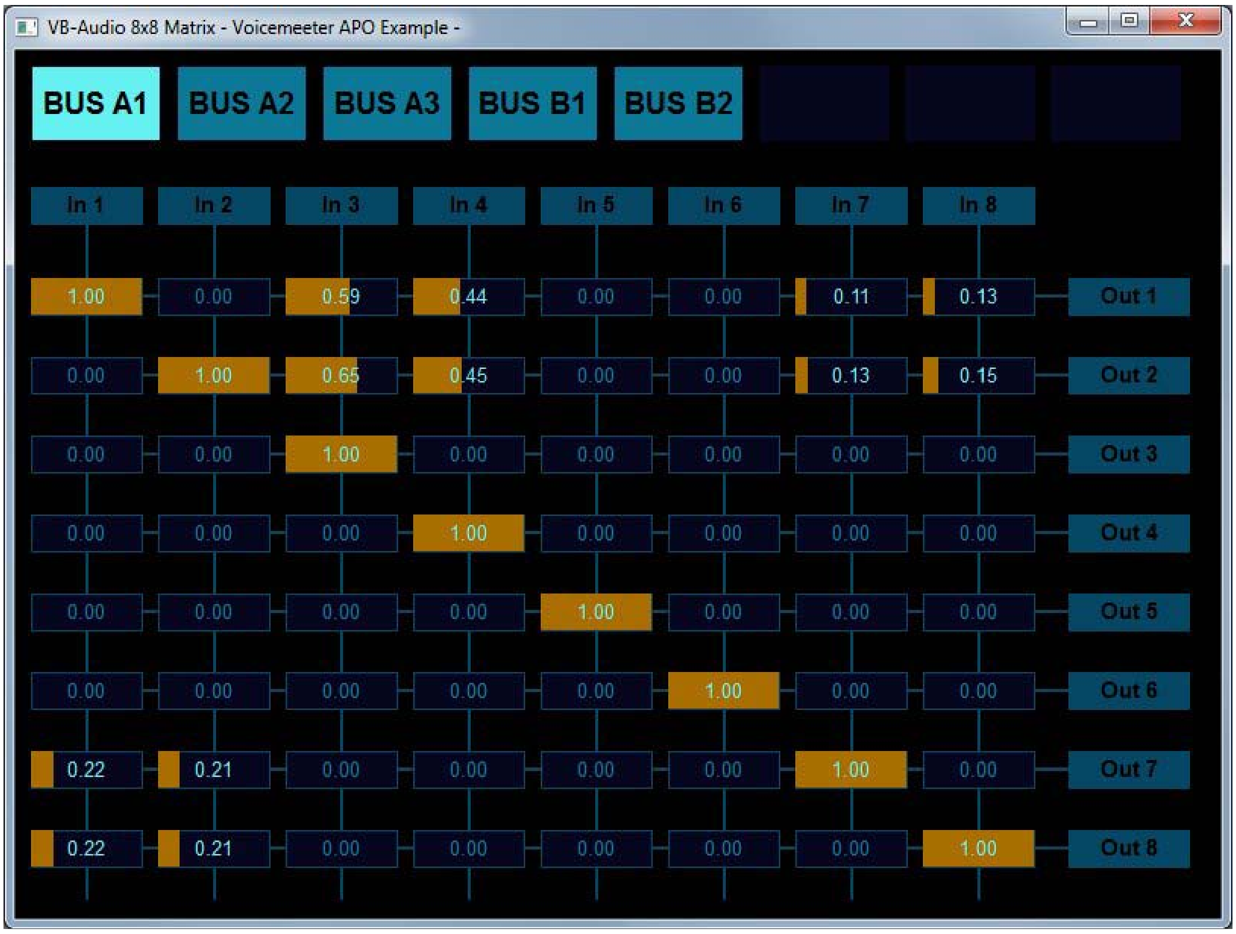
The Audio Callback is called to initialize/release your possible DSP object (according samplerate and buffer size) and to process audio buffers. This callback can manage 3 type of audio stream (possibly in the same callback) to process the INPUT INSERT Stream, the OUTPUT INSERT STREAM and/or the MAIN STREAM. (see vmr_client source code example).

```
long __stdcall PROCESSING_MyCallback(void * lpUser, long nCommand, void * lpData, long nnn)
{
	float * lpBufferIn;
	float * lpBufferOut;
	LPT_AUDIODSDPCTX lpctx;
	VBVMR_LPT_AUDIOINFO pinfo;
	VBVMR_LPT_AUDIOBUFFER lpa;
	// we can get back our context / object there
	lpctx = (LPT_AUDIODSDPCTX)lpUser;
	switch(nCommand)
	{

	//-------------------------------------
	// Init/End your object and allocate your memory
	//-------------------------------------
	case VBVMR_CBCOMMAND_STARTING:
		pinfo = (VBVMR_LPT_AUDIOINFO)lpData;
		// this is the first call of your Callback, made to let you initialize your
		// possible different DSP processing objects, allocate memory, precompute date...
		break;
	case VBVMR_CBCOMMAND_ENDING:
		pinfo = (VBVMR_LPT_AUDIOINFO)lpData;
		// this is the last call of your Callback, to release all your structure
		// previously allocated in the first call.
		break;
	case VBVMR_CBCOMMAND_CHANGE:
		pinfo = (VBVMR_LPT_AUDIOINFO)lpData;
		// this command is called if the samplerate or buffer size have changed
		break;
	//-------------------------------------
	// process buffer for Input INSERT
	//-------------------------------------
	case VBVMR_CBCOMMAND_BUFFER_IN:
		lpa =(VBVMR_LPT_AUDIOBUFFER)lpData;
		//with INSERT Callback, we have equal number of inputs and outputs.
		break;
	//-------------------------------------
	// process buffer for Output INSERT
	//-------------------------------------
	case VBVMR_CBCOMMAND_BUFFER_OUT:
		lpa =(VBVMR_LPT_AUDIOBUFFER)lpData;
		//with INSERT Callback, we have equal number of inputs and outputs.
		break;
	//-------------------------------------
	// process buffer for All I/O
	//-------------------------------------
	case VBVMR_CBCOMMAND_BUFFER_MAIN:
		lpa =(VBVMR_LPT_AUDIOBUFFER)lpData;
		//opposite to INSERT Callback we got all I/O as input
		//for Voicemeeter Banana 22 inputs + 40 outputs
		//for Voicemeeter Potato 34 inputs + 64 outputs
		//and only Voicemeeter outputs as output buffer (2x 5x or 8x BUS of 8 channels).
		break;
	}
	return 0;
}
```

Voicemeeter APO approach (Audio Processing Object):

With Audio Callback Insert Point, it is possible to build Audio Processing Object as Application easily. The SDK provides an example of 8x8 gain matrix to process a selected BUS (see vmr_matrix directory).



In 1	In 2	In 3	In 4	In 5	In 6	In 7	In 8	
1.00	0.00	0.59	0.44	0.00	0.00	0.11	0.13	Out 1
0.00	1.00	0.65	0.45	0.00	0.00	0.13	0.15	Out 2
0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	Out 3
0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	Out 4
0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	Out 5
0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	Out 6
0.22	0.21	0.00	0.00	0.00	0.00	1.00	0.00	Out 7
0.22	0.21	0.00	0.00	0.00	0.00	0.00	1.00	Out 8

The source code example of this application can be used as a starting point to build more complex or commercial applications needing to process BUSSES (as pre Master Section Insert).