

VSG60 API

Generated by Doxygen 1.9.4

1 VSG60 API Programming Manual	1
1.1 Examples	1
1.2 Signal Generation	1
1.3 Build/Version Notes	1
1.4 Development Requirements	2
1.5 I/Q Data and Output Power	2
1.6 Thread Safety	2
1.7 Multiple Devices and Multiple Processes	3
1.8 Status Codes and Error Handling	3
1.9 Functions	3
1.10 Linux Notes	3
1.10.1 Throughput	3
1.10.2 Multiple Devices	4
1.11 Other Programming Languages	4
1.12 Power Saving CPU Mode	4
1.13 Contact Information	4
2 Theory of Operation	5
2.1 Opening a Device	5
2.2 Basic Signal Generation	5
2.2.1 Example	5
2.2.2 Usage	6
2.3 Complex Signal Generation (Streaming)	6
2.3.1 Example	7
2.3.2 Usage	7
2.4 Closing the Device	8
2.5 Recalibration	8
3 File Index	9
3.1 File List	9
4 File Documentation	11
4.1 vsg_api.h File Reference	11
4.1.1 Detailed Description	12
4.1.2 Macro Definition Documentation	13
4.1.2.1 VSG_MAX_DEVICES	13
4.1.2.2 VSG60_MIN_FREQ	13
4.1.2.3 VSG200_MIN_FREQ	13
4.1.2.4 VSG60_MAX_FREQ	13
4.1.2.5 VSG200_MAX_FREQ	13
4.1.2.6 VSG_MIN_SAMPLE_RATE	13
4.1.2.7 VSG_MAX_SAMPLE_RATE	13
4.1.2.8 VSG_MIN_LEVEL	14

4.1.2.9 VSG_MAX_LEVEL	14
4.1.2.10 VSG_MIN_IQ_OFFSET	14
4.1.2.11 VSG_MAX_IQ_OFFSET	14
4.1.2.12 VSG_MIN_TRIGGER_LENGTH	14
4.1.2.13 VSG_MAX_TRIGGER_LENGTH	14
4.1.3 Enumeration Type Documentation	14
4.1.3.1 VsgDeviceType	14
4.1.3.2 VsgTimebaseState	15
4.1.3.3 VsgBool	15
4.1.3.4 VsgStatus	15
4.1.4 Function Documentation	16
4.1.4.1 vsgGetAPIVersion()	16
4.1.4.2 vsgGetDeviceList()	16
4.1.4.3 vsgOpenDevice()	17
4.1.4.4 vsgOpenDeviceBySerial()	17
4.1.4.5 vsgCloseDevice()	17
4.1.4.6 vsgPreset()	18
4.1.4.7 vsgRecal()	18
4.1.4.8 vsgAbort()	19
4.1.4.9 vsgGetSerialNumber()	19
4.1.4.10 vsgGetDeviceType()	19
4.1.4.11 vsgGetFirmwareVersion()	20
4.1.4.12 vsgGetCalDate()	20
4.1.4.13 vsgReadTemperature()	20
4.1.4.14 vsgSetRFOutputState()	22
4.1.4.15 vsgGetRFOutputState()	22
4.1.4.16 vsgSetTimebase()	23
4.1.4.17 vsgGetTimebase()	23
4.1.4.18 vsgSetTimebaseOffset()	23
4.1.4.19 vsgGetTimebaseOffset()	24
4.1.4.20 vsgSetFrequency()	24
4.1.4.21 vsgGetFrequency()	25
4.1.4.22 vsgSetSampleRate()	25
4.1.4.23 vsgGetSampleRate()	26
4.1.4.24 vsgSetLevel()	26
4.1.4.25 vsgGetLevel()	26
4.1.4.26 vsgSetAtten()	27
4.1.4.27 vsgGetIQScale()	27
4.1.4.28 vsgSetIQOffset()	28
4.1.4.29 vsgGetIQOffset()	28
4.1.4.30 vsgSetDigitalTuning()	29
4.1.4.31 vsgGetDigitalTuning()	29

4.1.4.32 vsgSetTriggerLength()	29
4.1.4.33 vsgGetTriggerLength()	30
4.1.4.34 vsgSubmitIQ()	30
4.1.4.35 vsgSubmitTrigger()	31
4.1.4.36 vsgFlush()	31
4.1.4.37 vsgFlushAndWait()	31
4.1.4.38 vsgOutputWaveform()	32
4.1.4.39 vsgRepeatWaveform()	32
4.1.4.40 vsgOutputCW()	33
4.1.4.41 vsgIsWaveformActive()	33
4.1.4.42 vsgGetUSBStatus()	33
4.1.4.43 vsgEnablePowerSavingCpuMode()	34
4.1.4.44 vsgGetErrorString()	34
4.2 vsg_api.h	34
Index	37

Chapter 1

VSG60 API Programming Manual

This document is a reference for the VSG60 application programming interface (API). The API provides a set of functions for controlling the **VSG60** signal generator.

1.1 Examples

All code examples are located in the *examples/* folder in the SDK.

1.2 Signal Generation

There are two main approaches to generating signals with the VSG60A:

- [Basic Signal Generation](#)
- [Complex Signal Generation \(Streaming\)](#)

1.3 Build/Version Notes

Versions are of the form **major.minor.revision**.

A **major** change signifies a significant change in functionality relating to one or more measurements, or the addition of significant functionality. Function prototypes have likely changed.

A **minor** change signifies additions that may improve existing functionality or fix major bugs but makes no changes that might affect existing user's measurements. Function prototypes can change but do not change existing parameters meanings.

A **revision** change signifies minor changes or bug fixes. Function prototypes will not change. Users should be able to update by simply replacing DLL.

Version 1.0.0 – Official release

1.4 Development Requirements

- Windows development
 - Windows 11/10
 - Windows C/C++ development tools and environment.
 - * API was compiled using VS2019 and VS2012.
 - VS2019/VS2012 C++ redistributables are required.
 - Library files [vsg_api.h](#), [vsg_api.lib](#), and [vsg_api.dll](#).
- Linux development
 - Ubuntu 22.04/20.04/18.04
 - Compiled with system GCC compiler on Ubuntu 18.04.
 - Library files [vsg_api.h](#), [vsg_api.so](#)
- VSG60 device
- USB 3.0 connectivity provided through 4th generation or later Intel CPUs.
- Dual core Intel i5/i7 processor minimum.

1.5 I/Q Data and Output Power

Waveforms are provided to the API as I/Q samples. I/Q samples should be provided as contiguous interleaved real and imaginary pairs.

Example: $[re_1, im_1, re_2, im_2, \dots, re_N, im_N]$ would be an array of N I/Q samples, which would equal $2 * N$ contiguous floating-point values.

Each {re, im} pair represents a single sample. Each real and imaginary value should be a 32-bit floating point value. I/Q samples are provided in full scale. An I/Q magnitude equal to 1.0 will transmit at the output level set by the user.

Magnitude of I/Q sample is calculated as $\sqrt{I^2 + Q^2}$

To measure the output power of a sample in dBm, use the formula:

Power of I/Q sample = Output power set in [vsgSetLevel](#) + $20 * \log_{10}(\text{magnitude of I/Q sample})$

Internally I/Q samples are scaled to achieve the user-selected output power. The default scaling is 0.5 and increases/decreases around this value to digitally scale where the internal amplifier and attenuator cannot. The internal scale can be queried through the API. Because we use a base scale of 0.5, this means magnitudes greater than 1.0 can be tolerated. Clipping occurs when either I or Q value exceeds 1.0 post scaling. The scaling is performed as such.

Scaled I/Q = { real * scaleFactor, imag * scaleFactor }

If you have a waveform with amplitudes much greater than 1.0, it is recommended to query the scale to verify clipping won't occur or scale the entire waveform and adjust the output power to compensate.

1.6 Thread Safety

The VSG60 API is not thread safe. A multi-threaded application is free to call the API from any number of threads if the function calls are synchronized (i.e. using a mutex). Not synchronizing your function calls will lead to undefined behavior.

1.7 Multiple Devices and Multiple Processes

The API can manage multiple devices within one process. In each process the API manages a list of open devices to prevent a process from opening a device more than once. You may open multiple devices by specifying the serial number of the device directly or allowing the API to discover them automatically.

If you wish to use the API in multiple processes, it is the user's responsibility to manage a list of devices to prevent the possibility of opening a device twice from two different processes. Two processes communicating to the same device will result in undefined behavior. One possible way to manage inter-process information is to use a named mutex on a Windows system.

If you wish to interface multiple devices on Linux, see [Multiple Devices](#).

1.8 Status Codes and Error Handling

All functions return a [VsgStatus](#) error code. [VsgStatus](#) is an enumerated type representing the success of a given function call. The integer values associated with each status provides information about whether a function call succeeded or failed.

An integer value of zero indicates no error or warnings. Negative integer status values indicate errors and positive values represent warnings.

A descriptive string of each status type can be retrieved using the [vsgGetErrorString](#) function.

1.9 Functions

All functions other than initialization functions take a device handle as the first parameters. This integer is obtained after opening the device through either the [vsgOpenDevice](#) or [vsgOpenDeviceBySerial](#) function. This handle uniquely identifies the receiver for the duration of the application execution, or until [vsgCloseDevice](#) is called.

Each function returns an error code which can provide warnings or errors related to the execution of the function. There are many cases where you will need to monitor these codes to determine the success or failure of an operation. See a list of common error codes and their descriptions in the Appendix.

1.10 Linux Notes

1.10.1 Throughput

By default, Linux applications cannot increase the priority of individual threads unless ran with elevated privilege (root). On Windows this issue does not exist, and the API will elevate the USB data acquisition threads to a higher priority to ensure USB data loss does not occur. On Linux, the user will need to run their application as root to ensure USB data acquisition is performed at a higher priority.

If this is not done, there is a higher risk of USB data loss.

In our testing, if little additional processing is occurring outside the API, 1 or 2 devices typically will not experience data loss due to this issue. Once the user application increases the processing load or starts performing I/O such as storing data to disk, the occurrence of USB data loss increases and the need to run the application as root increases.

1.10.2 Multiple Devices

There are limitations that apply when attempting to use multiple devices on Linux. The maximum amount of memory that can be allocated for USB transfers on Linux is 16MB. A single VSG60A can stay within this limitation, but two devices will exceed this limitation and can cause the API to crash when you do. The USB allocation limit can be changed by writing to the file

```
/sys/module/usbcore/parameters/usbfs_memory_mb
```

A good value would be $N * 16$ where N is the number of devices you plan on interfacing.

One way to write to this file is with the command

```
sudo sh -c 'echo 32 > /sys/module/usbcore/parameters/usbfs_memory_mb'
```

where 32 can be replaced with any value you wish.

1.11 Other Programming Languages

The VSG60 interface is C compatible which ensures it is possible to interface the API in most languages that can call C functions. These languages include C++, C#, Python, MATLAB, LabVIEW, Java, etc. Some examples of calling the VSG60 API in these other languages are included in the code examples folder.

The VSG60 API consists of several enumerated(enum) types, which are often used as parameters. These values can be treated as 32-bit integers when calling the API functions from other programming languages. You will need to match the enumerated values defined in the API header file.

1.12 Power Saving CPU Mode

Newer CPU models implement efficient power saving techniques that can interfere with and reduce USB bandwidth. If you are using one of these CPU models, you can experience issues with the VSG60 which might appear as data loss when inspecting the output of the VSG60.

We offer 2 potential solutions to this problem:

- 1) Enable the power saving CPU mode through the API. This has the effect of adding an artificial load to the API to keep the CPU from entering any low power CPU states that might affect the USB throughput. You will see an increase in CPU usage through this method.
- 2) Disable "C-States" in the BIOS of the PC. This prevents the OS from being able to put the CPU in these low power states which affect USB performance. This will increase power consumption of the PC which will affect battery life but will see lower CPU usage (since power saving CPU mode can be disabled).

The default state of this mode is disabled in the API.

PCs most affected are laptops and ultraportable devices running Windows.

1.13 Contact Information

For technical questions, email aj@signalhound.com.

For sales questions, email sales@signalhound.com.

Chapter 2

Theory of Operation

There are two primary ways to generate waveforms with the VSG60 API:

1. **Basic** Provide a complete waveform which the API will output once or repeat until stopped. This is the simplest method for generation. These methods are ideal for fixed frequency output waveforms up to many seconds in length.
2. **Complex** Use the streaming functions which allow for long waveforms or complex sequences which might involve frequency hopping and level changes.

2.1 Opening a Device

Before any generation can occur, the device must be opened and initialized. Opening and initializing a device through the API is performed through the [vsgOpenDevice](#) or [vsgOpenDeviceBySerial](#) functions. These functions will perform the full initialization of the device and if successful, will return an integer handle which can be used to reference the device for the remainder of your program. See the list of all VSG60 devices connected to the PC via the [vsgGetDeviceList](#) function.

2.2 Basic Signal Generation

To output a static waveform, either once, or repeatedly, use basic signal generation.

2.2.1 Example

For a list of all examples, please see the *examples/* folder in the SDK.

```
/*
 * This example illustrates generating a pulse signal at a specific freq and level.
 * This example is fully standalone.
 */
#include "vsg_api.h"
#include <stdio>
#include <vector>
#include <Windows.h>
struct Cplx32f {
    float re, im;
};
void vsg_example_basic_generation_2()
{
    // Open device, get handle, check open result
```

```

int handle;
VsgStatus status = vsgOpenDevice(&handle);
if(status < vsgNoError) {
    printf("Error: %s\n", vsgGetErrorString(status));
    return;
}
// Configure generator
const double freq = 1.0e9; // Hz
const double sampleRate = 50.0e6; // samples per second
const double level = -10.0; // dBm
vsgSetFrequency(handle, freq);
vsgSetLevel(handle, level);
vsgSetSampleRate(handle, sampleRate);
// Create pulse with specific width and period, then continually loop the pulsed signal
const Cplx32f cplxOne = {1.0, 0.0}, cplxZero = {0.0, 0.0};
std::vector<Cplx32f> iq; // The I/Q waveform
const double pulseWidth = 1.0e-6; // lus
const double pulsePeriod = 10.0e-6; // 10us
const int pulseOnSamples = pulseWidth * sampleRate;
const int pulseOffSamples = (pulsePeriod - pulseWidth) * sampleRate;
for(int i = 0; i < pulseOnSamples; i++) {
    iq.push_back(cplxOne);
}
for(int i = 0; i < pulseOffSamples; i++) {
    iq.push_back(cplxZero);
}
vsgRepeatWaveform(handle, (float*)&iq[0], iq.size());
// Will transmit until you close the device or abort
Sleep(5000);
// Stop waveform
vsgAbort(handle);
// Done with device
vsgCloseDevice(handle);
}

```

2.2.2 Usage

Basic signal generation involves configuring the generator and then using one of the following functions:

- [vsgOutputWaveform](#)
- [vsgRepeatWaveform](#)
- [vsgOutputCW](#) (convenience function)

Waveforms are provided as interleaved I/Q complex pairs. The API can output the waveform once using the [vsgOutputWaveform](#) function, or continually generate the waveform using the [vsgRepeatWaveform](#) function.

The [vsgOutputWaveform](#) is a blocking function which returns once fully output. When a waveform is on repeat, any changes in configuration will cause the waveform to be paused, and then restarted once reconfiguration has completed.

Submitting a trigger or I/Q data through the [vsgSubmitIQ](#) function will stop any waveforms on repeat.

Calling [vsgAbort](#) will end any active waveforms on repeat.

2.3 Complex Signal Generation (Streaming)

For long waveform generation or for transmitting a complex sequence of events such as frequency hopping or triggers, a collection of functions are available in the API which allow a user to buffer a sequence of configuration and transmit events.

2.3.1 Example

For a list of all examples, please see the *examples/* folder in the SDK.

```
/*
 * This example illustrates how to use the streaming function of the API
 * to generate a complex frequency hopping CW signal.
 * This example is fully standalone.
 */
#include "vsg_api.h"
#include <cstdio>
#include <vector>
#include <Windows.h>
struct Cplx32f {
    float re, im;
};
static double RandBetween(double f1, double f2)
{
    double r = (double)rand() / (double)RAND_MAX;
    return f1 + r * (f2 - f1);
}
void vsg_example_complex_freq_hopping()
{
    // Open device, get handle, check open result
    int handle;
    VsgStatus status = vsgOpenDevice(&handle);
    if(status < vsgNoError) {
        printf("Error: %s\n", vsgGetErrorString(status));
        return;
    }
    // Configure generator
    const double sampleRate = 50.0e6; // samples per second
    const double level = -10.0; // dBm
    vsgSetLevel(handle, level);
    vsgSetSampleRate(handle, sampleRate);
    // Each frequency switch takes 200us
    // We will output an 800us CW at each frequency for a hop period of 1ms
    std::vector<Cplx32f> iq(800.0e-6 * sampleRate);
    for(int i = 0; i < iq.size(); i++) {
        iq[i].re = 1.0;
        iq[i].im = 0.0;
    }
    // Number of 1ms hops to perform
    int hops = 1000;
    while(hops-- > 0) {
        vsgSetFrequency(handle, RandBetween(990.0e6, 1010.0e6));
        vsgSubmitIQ(handle, (float*)&iq[0], iq.size());
    }
    vsgFlushAndWait(handle);
    // Done with device
    vsgCloseDevice(handle);
}
```

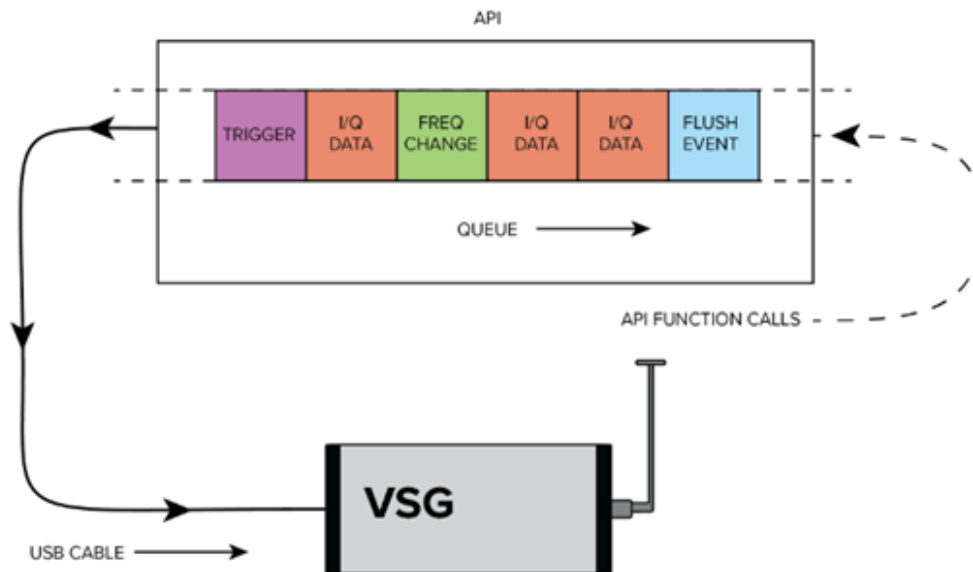
2.3.2 Usage

The following functions can be buffered/queued:

- [vsgSetFrequency](#)
- [vsgSetLevel](#)
- [vsgSubmitIQ](#)
- [vsgSubmitTrigger](#)
- [vsgFlushAndWait](#)

Roughly 1/5th of a second of I/Q data can be buffered. If the buffer is full and [vsgSubmitIQ](#) data is called, the function blocks until space is available in the buffer.

Calling [vsgAbort](#) will cause all I/Q data in the buffer to be dumped and any pending frequency/level changes to be completed in the order received.



2.4 Closing the Device

When finished, you can close the device and free all resources related to the device with the [vsgCloseDevice](#) function. Once closed, the device will appear in the open device list again. It is possible to open and close a device multiple times during the execution of a program.

2.5 Recalibration

Recalibration is performed by calling the [vsgRecal](#) function which retrieves the current device temperature and recalibrates the device for the current device settings. This will interrupt any signal generation currently in progress.

Large temperature changes affect signal generation in the form of reduce amplitude accuracy and reduced spurious performance, and it is recommended to reconfigure the device after large environmental changes and during device warmup.

Recalibration can occur automatically during periods of activity that include frequency changes, but when generating the same signal for long periods of time, or after a long period of inactivity, a recalibration is recommended.

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

vsg_api.h	API functions for the VSG60/VSG200 vector signal generators	11
---------------------------	---	----

Chapter 4

File Documentation

4.1 vsg_api.h File Reference

API functions for the VSG60/VSG200 vector signal generators.

Macros

- `#define VSG_MAX_DEVICES (8)`
- `#define VSG60_MIN_FREQ (30.0e6)`
- `#define VSG200_MIN_FREQ (100.0e3)`
- `#define VSG60_MAX_FREQ (6.0e9)`
- `#define VSG200_MAX_FREQ (20.0e9)`
- `#define VSG_MIN_SAMPLE_RATE (12.5e3)`
- `#define VSG_MAX_SAMPLE_RATE (54.0e6)`
- `#define VSG_MIN_LEVEL (-120.0)`
- `#define VSG_MAX_LEVEL (10.0)`
- `#define VSG_MIN_IQ_OFFSET (-1024)`
- `#define VSG_MAX_IQ_OFFSET (1024)`
- `#define VSG_MIN_TRIGGER_LENGTH (0.1e-6)`
- `#define VSG_MAX_TRIGGER_LENGTH (0.1)`

Enumerations

- `enum VsgDeviceType { VsgDeviceType60 = 0 , VsgDeviceType200 = 1 }`
- `enum VsgTimebaseState { vsgTimebaseStateInternal = 0 , vsgTimebaseStateExternal = 1 }`
- `enum VsgBool { vsgFalse = 0 , vsgTrue = 1 }`
- `enum VsgStatus { }`

Functions

- VSG_API const char * [vsgGetAPIVersion](#) ()
- VSG_API [VsgStatus](#) [vsgGetDeviceList](#) (int *serials, int *count)
- VSG_API [VsgStatus](#) [vsgOpenDevice](#) (int *handle)
- VSG_API [VsgStatus](#) [vsgOpenDeviceBySerial](#) (int *handle, int serialNumber)
- VSG_API [VsgStatus](#) [vsgCloseDevice](#) (int handle)
- VSG_API [VsgStatus](#) [vsgPreset](#) (int handle)
- VSG_API [VsgStatus](#) [vsgRecal](#) (int handle)
- VSG_API [VsgStatus](#) [vsgAbort](#) (int handle)
- VSG_API [VsgStatus](#) [vsgGetSerialNumber](#) (int handle, int *serial)
- VSG_API [VsgStatus](#) [vsgGetDeviceType](#) (int handle, [VsgDeviceType](#) *deviceType)
- VSG_API [VsgStatus](#) [vsgGetFirmwareVersion](#) (int handle, int *version)
- VSG_API [VsgStatus](#) [vsgGetCalDate](#) (int handle, uint32_t *lastCalDate)
- VSG_API [VsgStatus](#) [vsgReadTemperature](#) (int handle, float *temp)
- VSG_API [VsgStatus](#) [vsgSetRFOutputState](#) (int handle, [VsgBool](#) enabled)
- VSG_API [VsgStatus](#) [vsgGetRFOutputState](#) (int handle, [VsgBool](#) *enabled)
- VSG_API [VsgStatus](#) [vsgSetTimebase](#) (int handle, [VsgTimebaseState](#) state)
- VSG_API [VsgStatus](#) [vsgGetTimebase](#) (int handle, [VsgTimebaseState](#) *state)
- VSG_API [VsgStatus](#) [vsgSetTimebaseOffset](#) (int handle, double ppm)
- VSG_API [VsgStatus](#) [vsgGetTimebaseOffset](#) (int handle, double *ppm)
- VSG_API [VsgStatus](#) [vsgSetFrequency](#) (int handle, double frequency)
- VSG_API [VsgStatus](#) [vsgGetFrequency](#) (int handle, double *frequency)
- VSG_API [VsgStatus](#) [vsgSetSampleRate](#) (int handle, double sampleRate)
- VSG_API [VsgStatus](#) [vsgGetSampleRate](#) (int handle, double *sampleRate)
- VSG_API [VsgStatus](#) [vsgSetLevel](#) (int handle, double level)
- VSG_API [VsgStatus](#) [vsgGetLevel](#) (int handle, double *level)
- VSG_API [VsgStatus](#) [vsgSetAtten](#) (int handle, int atten)
- VSG_API [VsgStatus](#) [vsgGetIQScale](#) (int handle, double *iqScale)
- VSG_API [VsgStatus](#) [vsgSetIQOffset](#) (int handle, int16_t iOffset, int16_t qOffset)
- VSG_API [VsgStatus](#) [vsgGetIQOffset](#) (int handle, int16_t *iOffset, int16_t *qOffset)
- VSG_API [VsgStatus](#) [vsgSetDigitalTuning](#) (int handle, [VsgBool](#) enabled)
- VSG_API [VsgStatus](#) [vsgGetDigitalTuning](#) (int handle, [VsgBool](#) *enabled)
- VSG_API [VsgStatus](#) [vsgSetTriggerLength](#) (int handle, double seconds)
- VSG_API [VsgStatus](#) [vsgGetTriggerLength](#) (int handle, double *seconds)
- VSG_API [VsgStatus](#) [vsgSubmitIQ](#) (int handle, float *iq, int len)
- VSG_API [VsgStatus](#) [vsgSubmitTrigger](#) (int handle)
- VSG_API [VsgStatus](#) [vsgFlush](#) (int handle)
- VSG_API [VsgStatus](#) [vsgFlushAndWait](#) (int handle)
- VSG_API [VsgStatus](#) [vsgOutputWaveform](#) (int handle, float *iq, int len)
- VSG_API [VsgStatus](#) [vsgRepeatWaveform](#) (int handle, float *iq, int len)
- VSG_API [VsgStatus](#) [vsgOutputCW](#) (int handle)
- VSG_API [VsgStatus](#) [vsgIsWaveformActive](#) (int handle, [VsgBool](#) *active)
- VSG_API [VsgStatus](#) [vsgGetUSBStatus](#) (int handle)
- VSG_API void [vsgEnablePowerSavingCpuMode](#) ([VsgBool](#) enabled)
- VSG_API const char * [vsgGetErrorString](#) ([VsgStatus](#) status)

4.1.1 Detailed Description

API functions for the VSG60/VSG200 vector signal generators.

This is the main file for user accessible functions for controlling the VSG60 and VSG200 vector signal generator.

4.1.2 Macro Definition Documentation

4.1.2.1 VSG_MAX_DEVICES

```
#define VSG_MAX_DEVICES (8)
```

Maximum number of devices that can be managed by the API.

4.1.2.2 VSG60_MIN_FREQ

```
#define VSG60_MIN_FREQ (30.0e6)
```

Minimum configurable center frequency for VSG60 in Hz.

4.1.2.3 VSG200_MIN_FREQ

```
#define VSG200_MIN_FREQ (100.0e3)
```

Minimum configurable center frequency for VSG200 in Hz.

4.1.2.4 VSG60_MAX_FREQ

```
#define VSG60_MAX_FREQ (6.0e9)
```

Maximum configurable center frequency for VSG60 in Hz.

4.1.2.5 VSG200_MAX_FREQ

```
#define VSG200_MAX_FREQ (20.0e9)
```

Maximum configurable center frequency for VSG200 in Hz.

4.1.2.6 VSG_MIN_SAMPLE_RATE

```
#define VSG_MIN_SAMPLE_RATE (12.5e3)
```

Minimum configurable sample (DAC) rate in Hz.

4.1.2.7 VSG_MAX_SAMPLE_RATE

```
#define VSG_MAX_SAMPLE_RATE (54.0e6)
```

Maximum configurable sample (DAC) rate in Hz.

4.1.2.8 VSG_MIN_LEVEL

```
#define VSG_MIN_LEVEL (-120.0)
```

Minimum configurable output level in dBm.

4.1.2.9 VSG_MAX_LEVEL

```
#define VSG_MAX_LEVEL (10.0)
```

Maximum configurable output level in dBm.

4.1.2.10 VSG_MIN_IQ_OFFSET

```
#define VSG_MIN_IQ_OFFSET (-1024)
```

Minimum configurable I/Q offset

4.1.2.11 VSG_MAX_IQ_OFFSET

```
#define VSG_MAX_IQ_OFFSET (1024)
```

Maximum configurable I/Q offset

4.1.2.12 VSG_MIN_TRIGGER_LENGTH

```
#define VSG_MIN_TRIGGER_LENGTH (0.1e-6)
```

Minimum configurable trigger length in seconds.

4.1.2.13 VSG_MAX_TRIGGER_LENGTH

```
#define VSG_MAX_TRIGGER_LENGTH (0.1)
```

Maximum configurable trigger length in seconds.

4.1.3 Enumeration Type Documentation

4.1.3.1 VsgDeviceType

```
enum VsgDeviceType
```

Device type.

Enumerator

VsgDeviceType60	VSG60
VsgDeviceType200	VSG200

4.1.3.2 VsgTimebaseState

```
enum VsgTimebaseState
```

Used to indicate the source of the timebase reference for the device.

Enumerator

vsgTimebaseStateInternal	Use the internal 10MHz timebase.
vsgTimebaseStateExternal	Use an external 10MHz timebase on the 10 MHz In port.

4.1.3.3 VsgBool

```
enum VsgBool
```

Boolean type. Used in public facing functions instead of `bool` to improve API use from different programming languages.

Enumerator

vsgFalse	False
vsgTrue	True

4.1.3.4 VsgStatus

```
enum VsgStatus
```

Status code returned from all API functions.

Enumerator

vsgWaveformAlreadyActiveErr	The auto pattern output is already active.
vsgWaveformNotActiveErr	The waveform is already inactive.
vsgUsbXferErr	There was a problem during USB data transfer.
vsgInvalidParameterErr	Required parameter found to have invalid value.
vsgNullPtrErr	One or more required pointer parameters were null.

Enumerator

<code>vsgInvalidDeviceErr</code>	User specified invalid device index.
<code>vsgDeviceNotFoundErr</code>	Unable to open device.
<code>vsgNoError</code>	Function returned successfully. No error.
<code>vsgAlreadyFlushed</code>	There are no pending operations.
<code>vsgSettingClamped</code>	One or more of the provided settings were adjusted.

4.1.4 Function Documentation

4.1.4.1 `vsgGetAPIVersion()`

```
VSG_API const char * vsgGetAPIVersion ( )
```

Returns a string indicating the version of the API. The returned string is of the form major.minor.revision. Ascii periods (‘.’) separate positive integers. Major/minor/revision are not guaranteed to be a single decimal digit. The string is null terminated. The string should not be modified or freed by the user. An example string is below. .. [‘3’ | ‘.’ | ‘0’ | ‘.’ | ‘1’ | ‘1’ | ‘\0’] = “3.0.11”

Returns

4.1.4.2 `vsgGetDeviceList()`

```
VSG_API VsgStatus vsgGetDeviceList (
    int * serials,
    int * count )
```

This function is used to retrieve the serial number of all unopened VSG60 devices connected to the PC. If any device is open in a different process, it will be returned by this function. The serial numbers returned can then be used to open specific devices with the [vsgOpenDeviceBySerial](#) function.

Parameters

<code>out</code>	<i>serials</i>	Pointer to array.
<code>in, out</code>	<i>count</i>	Pointer to integer that should equal the size of the serials array. If the function returns successfully, this value will be set to the number devices returned in the serials array. It will not exceed the size of the serials array.

Returns

4.1.4.3 vsgOpenDevice()

```
VSG_API VsgStatus vsgOpenDevice (
    int * handle )
```

Claim the first unopened VSG60 detected on the system. If the device is opened successfully, a handle to the device will be returned. This handle can then be used to interface this device for all future API calls. This function has the same effect as calling [vsgGetDeviceList](#) and using the first device found to call [vsgOpenDeviceBySerial](#).

Parameters

out	<i>handle</i>	Returns device handle.
-----	---------------	------------------------

Returns

4.1.4.4 vsgOpenDeviceBySerial()

```
VSG_API VsgStatus vsgOpenDeviceBySerial (
    int * handle,
    int serialNumber )
```

This function operates similarly to [vsgOpenDevice](#) except it allows you to specify the device you wish to open. This function is often used in conjunction with [vsgGetDeviceList](#) when managing several VSG60 devices on one PC.

Parameters

out	<i>handle</i>	If this function returns successfully, handle will point to an integer that can be used to access this device through the API.
in	<i>serialNumber</i>	The serial number of the device you wish to open.

Returns

4.1.4.5 vsgCloseDevice()

```
VSG_API VsgStatus vsgCloseDevice (
    int handle )
```

This function should be called when you are finished with the VSG60. It will release all resources for the device and the device will become available again for use in the current process. The device handle specified will no longer point to a valid device and the device must be re-opened again to be used. This function should be called before the process exits.

Parameters

in	<i>handle</i>	Device handle.
----	---------------	----------------

Returns**4.1.4.6 vsgPreset()**

```
VSG_API VsgStatus vsgPreset (  
    int handle )
```

Performs a full device preset. When this function returns, the hardware will have performed a full reset, the device handle will no longer be valid, [vsgCloseDevice](#) will have been called for the device handle, and the device will need to be re-opened. This function can be used to recover from an undesirable device state. This function takes roughly 3 seconds to complete.

Parameters

in	<i>handle</i>	Device handle.
----	---------------	----------------

Returns**4.1.4.7 vsgRecal()**

```
VSG_API VsgStatus vsgRecal (  
    int handle )
```

When operating the VSG60 for long periods of time with a fixed configuration, environmental changes leading to changes in the internal operating temperature of the VSG can cause signal drift leading to loss of amplitude accuracy and image rejection performance. This function aborts any current operation, and updates internal temperature corrections for the current configuration.

Parameters

in	<i>handle</i>	Device handle.
----	---------------	----------------

Returns

4.1.4.8 vsgAbort()

```
VSG_API VsgStatus vsgAbort (
    int handle )
```

This function returns the device to an idle state. If any waveform is being generated, it will be stopped. If the device is streaming in the complex generation mode, all I/Q data pending is discarded and all frequency/level changes are finished before returning. When this function returns, the device will be in an idle state.

Parameters

in	<i>handle</i>	Device handle.
----	---------------	----------------

Returns

4.1.4.9 vsgGetSerialNumber()

```
VSG_API VsgStatus vsgGetSerialNumber (
    int handle,
    int * serial )
```

Retreive the device serial number.

Parameters

in	<i>handle</i>	Device handle.
out	<i>serial</i>	Serial number.

Returns

4.1.4.10 vsgGetDeviceType()

```
VSG_API VsgStatus vsgGetDeviceType (
    int handle,
    VsgDeviceType * deviceType )
```

Retreive the device type.

Parameters

in	<i>handle</i>	Device handle.
out	<i>deviceType</i>	Pointer to device type variable.

Returns

4.1.4.11 vsgGetFirmwareVersion()

```
VSG_API VsgStatus vsgGetFirmwareVersion (
    int handle,
    int * version )
```

Retrieve the firmware version number. The version is a single integer value.

Parameters

in	<i>handle</i>	Device handle.
out	<i>version</i>	Firmware version.

Returns

4.1.4.12 vsgGetCalDate()

```
VSG_API VsgStatus vsgGetCalDate (
    int handle,
    uint32_t * lastCalDate )
```

Retrieve the last calibration date as the seconds since epoch.

Parameters

in	<i>handle</i>	Device handle.
out	<i>lastCalDate</i>	Calibration date as seconds since epoch.

Returns

4.1.4.13 vsgReadTemperature()

```
VSG_API VsgStatus vsgReadTemperature (
    int handle,
    float * temp )
```

Retrive the device temperature in Celcius. If the device is not idle, the last read temperature is returned, otherwise the device temperature is queried before returning.

Parameters

in	<i>handle</i>	Device handle.
out	<i>temp</i>	Device temperature in C.

Returns**4.1.4.14 vsgSetRFOutputState()**

```
VSG_API VsgStatus vsgSetRFOutputState (
    int handle,
    VsgBool enabled )
```

Use this function to disable the RF output of the VSG60. Even when the VSG is not transmitting, it might still be emitting spurious energy related to clock frequencies and the DC offset. Disabling the RF output will eliminate most spurious signals. The RF output is enabled by default. When the RF output is disabled the only way to enable it again is to call this function. Until then, all actions will continue to be performed but with the RF output disabled. Additionally, the [vsgAbort](#) function is called when this function is called. This means any signal actively being generated will be stopped when this function is called.

Parameters

in	<i>handle</i>	Device handle.
in	<i>enabled</i>	Set to vsgFalse to disable the RF output.

Returns**4.1.4.15 vsgGetRFOutputState()**

```
VSG_API VsgStatus vsgGetRFOutputState (
    int handle,
    VsgBool * enabled )
```

Retrive the RF output state.

Parameters

in	<i>handle</i>	Device handle.
out	<i>enabled</i>	Returns vsgTrue if the RF output is enabled.

Returns

4.1.4.16 vsgSetTimebase()

```
VSG_API VsgStatus vsgSetTimebase (
    int handle,
    VsgTimebaseState state )
```

Specify whether the VSG60 should use its internal 10MHz reference or one provided on the 10MHz reference input port. If external reference is selected and no reference is provided, the frequency error may be off by several PPM. If the state provided matches the current state, the function returns immediately. Any active waveforms are paused, and the stream is flushed before this operation takes place.

Parameters

<i>handle</i>	Device handle.
<i>state</i>	New timebase state.

Returns

4.1.4.17 vsgGetTimebase()

```
VSG_API VsgStatus vsgGetTimebase (
    int handle,
    VsgTimebaseState * state )
```

Retrieve the current timebase state.

Parameters

<i>handle</i>	Device handle.
<i>state</i>	Returns current timebase state.

Returns

4.1.4.18 vsgSetTimebaseOffset()

```
VSG_API VsgStatus vsgSetTimebaseOffset (
    int handle,
    double ppm )
```

Adjust the VSG60 timebase. The adjustment is provided in parts per million (PPM) This adjustment will only last until the device is closed via [vsgCloseDevice](#) or the program is terminated. If the value provided matches the currently set value, this function returns immediately. Any active waveforms are paused, and the stream is flushed before this operation takes place.

Parameters

<i>handle</i>	Device handle.
<i>ppm</i>	New timebase offset. The value will be clamped between [-2,+2] ppm.

Returns

4.1.4.19 vsgGetTimebaseOffset()

```
VSG_API VsgStatus vsgGetTimebaseOffset (
    int handle,
    double * ppm )
```

Retrieve the current user configured timebase offset in parts per million (PPM).

Parameters

<i>handle</i>	Device handle.
<i>ppm</i>	Returns current user PPM offset.

Returns

4.1.4.20 vsgSetFrequency()

```
VSG_API VsgStatus vsgSetFrequency (
    int handle,
    double frequency )
```

Set the center frequency of the signal generator. This function is used for both basic and streaming operation. In streaming operation, this operation takes 200us to complete. This operation will occur even if the provided frequency matches the current frequency. This operation may configure a recalibration (at no time penalty).

Parameters

<i>handle</i>	Device handle.
<i>frequency</i>	New center frequency in Hz.

Returns

4.1.4.21 vsgGetFrequency()

```
VSG_API VsgStatus vsgGetFrequency (
    int handle,
    double * frequency )
```

Retrieve the current center frequency. This function will return the last user configured center frequency even if the hardware has yet to update to the new value.

Parameters

<i>handle</i>	Device handle.
<i>frequency</i>	Returns the center frequency in Hz.

Returns

4.1.4.22 vsgSetSampleRate()

```
VSG_API VsgStatus vsgSetSampleRate (
    int handle,
    double sampleRate )
```

Sets the I/Q (DAC) sample rate of the signal generator. The streaming queue is flushed via [vsgFlushAndWait](#) before the sample rate is updated. If the supplied sample rate is the same as the current rate, this function returns immediately and no operation occurs. A full sample rate change takes approximately 200-250ms.

Parameters

<i>handle</i>	Device handle.
<i>sampleRate</i>	New sample rate in Hz.

Returns

4.1.4.23 vsgGetSampleRate()

```
VSG_API VsgStatus vsgGetSampleRate (
    int handle,
    double * sampleRate )
```

Retrieve the current I/Q (DAC) sample rate.

Parameters

<i>handle</i>	Device handle.
<i>sampleRate</i>	Returns the sample rate in Hz.

Returns

4.1.4.24 vsgSetLevel()

```
VSG_API VsgStatus vsgSetLevel (
    int handle,
    double level )
```

Set the output level of the signal generator. The output level represents the output power at the RF output port when I/Q samples of magnitude 1.0 are transmitted. Hardware attenuation, amplification, and digital scaling are used to achieve the requested output level. What values are used depend on the temperature calibration coefficients for individual device and the frequency of the output. In streaming operation, this operation takes 10us to complete. This operation will occur even if the provided level matches the current level. This operation may configure a recalibration (at no time penalty).

Parameters

<i>handle</i>	Device handle.
<i>level</i>	New output level in dBm.

Returns

4.1.4.25 vsgGetLevel()

```
VSG_API VsgStatus vsgGetLevel (
    int handle,
    double * level )
```

Retrieve the current output level.

Parameters

<i>handle</i>	Device handle.
<i>level</i>	Output level in dBm.

Returns

4.1.4.26 vsgSetAtten()

```
VSG_API VsgStatus vsgSetAtten (
    int handle,
    int atten )
```

This function allows the customer to guarantee the configuration of the internal attenuator and amplifier directly by specifying a fixed system attenuation. Values that are positive utilize the amplifier to achieve the desired setting. Calling this function overrides the output level configured via [vsgSetLevel](#). A digital I/Q scale of 0.5 is used when attenuation is manually set. See [vsgGetIQScale](#).

Parameters

<i>handle</i>	Device handle.
<i>atten</i>	Attenuator value between [-50, 20] in 2dB steps. Must be an even number.

Returns

4.1.4.27 vsgGetIQScale()

```
VSG_API VsgStatus vsgGetIQScale (
    int handle,
    double * iqScale )
```

Returns the currently used digital scale applied to the I/Q data before transmitting. The digital scaling is used in conjunction with the hardware amplifier and attenuator to achieve the desired output level. This function does not interrupt any active waveforms or streaming generation.

Parameters

<i>handle</i>	Device handle.
<i>iqScale</i>	Returns floating point value between [0.0, 1.0].

Returns

4.1.4.28 vsgSetIQOffset()

```
VSG_API VsgStatus vsgSetIQOffset (
    int handle,
    int16_t iOffset,
    int16_t qOffset )
```

Specify an additional I/Q offset applied to the I/Q data before transmit. Used to fine improve carrier feedthrough. The offset lasts until the device is closed or the program is terminated. [vsgFlushAndWait](#) is called at the beginning of this function. If the supplied value matches the current value, this function returns immediately.

Parameters

<i>handle</i>	Device handle.
<i>iOffset</i>	I channel offset between [-1024,1024].
<i>qOffset</i>	Q channel offset between [-1024,1024].

Returns

4.1.4.29 vsgGetIQOffset()

```
VSG_API VsgStatus vsgGetIQOffset (
    int handle,
    int16_t * iOffset,
    int16_t * qOffset )
```

Retrieve the user configured I/Q offsets.

Parameters

<i>handle</i>	Device handle.
<i>iOffset</i>	Returns I channel offset.
<i>qOffset</i>	Returns Q channel offset.

Returns

4.1.4.30 vsgSetDigitalTuning()

```
VSG_API VsgStatus vsgSetDigitalTuning (
    int handle,
    VsgBool enabled )
```

If the value is provided is, this function returns immediately with no effect. [vsgFlushAndWait](#) is called before the operation occurs. See the [VSG60 Product Manual](#) for a description of digital tuning.

Parameters

<i>handle</i>	Device handle.
<i>enabled</i>	Set to vsgTrue to enable digital tuning.

Returns

4.1.4.31 vsgGetDigitalTuning()

```
VSG_API VsgStatus vsgGetDigitalTuning (
    int handle,
    VsgBool * enabled )
```

Retrieve whether digital tuning is enabled.

Parameters

<i>handle</i>	Device handle.
<i>enabled</i>	Returns vsgTrue if digital tuning is enabled.

Returns

4.1.4.32 vsgSetTriggerLength()

```
VSG_API VsgStatus vsgSetTriggerLength (
    int handle,
    double seconds )
```

Set the length of time the output trigger port remains high when a trigger is output. Default is 10us. (10.0e-6) The range of acceptable values is [100ns, 1s] This function does not interrupt any active waveforms or streaming operation.

Parameters

<i>handle</i>	Device handle.
<i>seconds</i>	Trigger length in seconds.

Returns**4.1.4.33 vsgGetTriggerLength()**

```
VSG_API VsgStatus vsgGetTriggerLength (
    int handle,
    double * seconds )
```

Retrieve the output trigger length.

Parameters

<i>handle</i>	Device handle.
<i>seconds</i>	Returns the trigger length in seconds.

Returns**4.1.4.34 vsgSubmitIQ()**

```
VSG_API VsgStatus vsgSubmitIQ (
    int handle,
    float * iq,
    int len )
```

Submit an array of I/Q samples to be generated with the current configuration. If an ARB waveform is currently being transmitted via the [vsgRepeatWaveform](#) function, it is aborted, and the device starts operating in the streaming configuration. This function should only be used for streaming operation. For generating simple waveforms, use the [vsgOutputWaveform](#) and [vsgRepeatWaveform](#) functions. This function will block until there is room in the processing and command queue. See [Complex Signal Generation \(Streaming\)](#) for more information.

Parameters

<i>handle</i>	Device handle.
<i>iq</i>	Pointer to array of interleaved I/Q values. The array must be len*2 floating point values.
<i>len</i>	Number of I/Q samples in the iq array

Returns

4.1.4.35 vsgSubmitTrigger()

```
VSG_API VsgStatus vsgSubmitTrigger (
    int handle )
```

Submit a streaming trigger event. If an ARB waveform is currently being transmitted, it is aborted, and the device starts operating in the streaming configuration. If a trigger is already active when the new trigger is output, it is first toggled low before re-toggling high, resetting the trigger high period in the process. See the [Basic Signal Generation](#) section for more information.

Parameters

<i>handle</i>	Device handle.
---------------	----------------

Returns

4.1.4.36 vsgFlush()

```
VSG_API VsgStatus vsgFlush (
    int handle )
```

Pushes all pending operations in a stream out to the device. This function should be called after a sequence of streaming events to ensure there are no gaps in the output. Also see [vsgFlushAndWait](#).

Parameters

<i>handle</i>	Device handle.
---------------	----------------

Returns

4.1.4.37 vsgFlushAndWait()

```
VSG_API VsgStatus vsgFlushAndWait (
    int handle )
```

Pushes all pending operations in a stream out to the device and waits for all operations to complete. This function should be called after a sequence of streaming events to ensure there are no gaps in the output. When this function returns, the device will be idle.

Parameters

<i>handle</i>	Device handle.
---------------	----------------

Returns

4.1.4.38 vsgOutputWaveform()

```
VSG_API VsgStatus vsgOutputWaveform (
    int handle,
    float * iq,
    int len )
```

Output the I/Q waveform once and returns. This function has the same effect as calling [vsgSubmitIQ](#) followed by [vsgFlushAndWait](#). This function returns once the waveform has been transmitted. The device will be in an idle state when returned. If an ARB waveform is active, it is aborted prior to transmission.

Parameters

<i>handle</i>	Device handle.
<i>iq</i>	Pointer to array of interleaved I/Q values. The array must be len*2 floating point values.
<i>len</i>	The number of I/Q samples in the iq array.

Returns

4.1.4.39 vsgRepeatWaveform()

```
VSG_API VsgStatus vsgRepeatWaveform (
    int handle,
    float * iq,
    int len )
```

This function instructs the API to continually generate the provided waveform. A full copy of the waveform is made before generation occurs. This function blocks until signal generation has begun. The repeated waveform is only stopped after calling the [vsgAbort](#) function or by calling a function that interrupts operation. Setting any other configuration value, such as frequency, level, etc. will pause the waveform until the configuration completes and then generation starts again from the beginning. See [vsgIsWaveformActive](#).

Parameters

<i>handle</i>	Device handle.
<i>iq</i>	Pointer to array of interleaved I/Q values. The array must be len*2 floating point values.
<i>len</i>	The number of I/Q samples in the iq array.

Returns

4.1.4.40 vsgOutputCW()

```
VSG_API VsgStatus vsgOutputCW (
    int handle )
```

Convenience function which outputs a CW signal with the current frequency, level, and sample rate. This function has the same effect as calling [vsgRepeatWaveform](#) with a single I/Q value of {1,0}.

Parameters

<i>handle</i>	Device handle.
---------------	----------------

Returns

4.1.4.41 vsgIsWaveformActive()

```
VSG_API VsgStatus vsgIsWaveformActive (
    int handle,
    VsgBool * active )
```

Detect whether a waveform is currently being generated via the [vsgRepeatWaveform](#) function.

Parameters

<i>handle</i>	Device handle.
<i>active</i>	Returns vsgTrue if a waveform is currently being transmitted.

Returns

4.1.4.42 vsgGetUSBStatus()

```
VSG_API VsgStatus vsgGetUSBStatus (
    int handle )
```

Return the USB status warning.

Parameters

<i>handle</i>	Device handle.
---------------	----------------

Returns**4.1.4.43 vsgEnablePowerSavingCpuMode()**

```
VSG_API void vsgEnablePowerSavingCpuMode (
    VsgBool enabled )
```

Enable power saving CPU mode See [Power Saving CPU Mode](#) for more information.

Parameters

<i>enabled</i>	Set to vsgTrue to enable power saving CPU mode.
----------------	---

4.1.4.44 vsgGetErrorString()

```
VSG_API const char * vsgGetErrorString (
    VsgStatus status )
```

Retrieve an ascii description string for a given VsgStatus. The string is useful for printing an debugging purposes.

Parameters

<i>status</i>	Status returned from an API function.
---------------	---------------------------------------

Returns

A pointer to a null terminated string. The memory should not be freed/deallocated/modified.

4.2 vsg_api.h

[Go to the documentation of this file.](#)

```
1 // Copyright (c) 2024, Signal Hound, Inc.
2 // For licensing information, please see the API license in the software_licenses folder
3
13 #ifndef VSG_API_H
14 #define VSG_API_H
15
16 #if _WIN32
17 #ifdef VSG_EXPORT
18 #define VSG_API __declspec(dllexport)
```



```

19 #else
20 #define VSG_API
21 #endif
22
23 // bare minimum stdint typedef support
24 #if _MSC_VER < 1700 // For VS2010 or earlier
25     typedef signed char      int8_t;
26     typedef short            int16_t;
27     typedef int               int32_t;
28     typedef long long         int64_t;
29     typedef unsigned char     uint8_t;
30     typedef unsigned short    uint16_t;
31     typedef unsigned int      uint32_t;
32     typedef unsigned long long uint64_t;
33 #else
34 #include <stdint.h>
35 #endif
36 #else // Linux
37 #include <stdint.h>
38 #define VSG_API __attribute__((visibility("default")))
39 #endif
40
41 #define VSG_MAX_DEVICES (8)
42 #define VSG60_MIN_FREQ (30.0e6)
43 #define VSG200_MIN_FREQ (100.0e3)
44 #define VSG60_MAX_FREQ (6.0e9)
45 #define VSG200_MAX_FREQ (20.0e9)
46 #define VSG_MIN_SAMPLE_RATE (12.5e3)
47 #define VSG_MAX_SAMPLE_RATE (54.0e6)
48 #define VSG_MIN_LEVEL (-120.0)
49 #define VSG_MAX_LEVEL (10.0)
50 #define VSG_MIN_IQ_OFFSET (-1024)
51 #define VSG_MAX_IQ_OFFSET (1024)
52 #define VSG_MIN_TRIGGER_LENGTH (0.1e-6)
53 #define VSG_MAX_TRIGGER_LENGTH (0.1)
54
55 typedef enum VsgDeviceType {
56     VsgDeviceType60 = 0,
57     VsgDeviceType200 = 1
58 } VsgDeviceType;
59
60 typedef enum VsgTimebaseState {
61     vsgTimebaseStateInternal = 0, // default
62     vsgTimebaseStateExternal = 1
63 } VsgTimebaseState;
64
65 typedef enum VsgBool {
66     vsgFalse = 0,
67     vsgTrue = 1
68 } VsgBool;
69
70 typedef enum VsgStatus {
71     // Internal use only
72     vsgFileIOErr = -1000,
73     vsgMemErr = -999,
74
75     vsgInvalidOperationErr = -11,
76
77     vsgWaveformAlreadyActiveErr = -10,
78     vsgWaveformNotActiveErr = -9,
79
80     vsgUsbXferErr = -5,
81     vsgInvalidParameterErr = -4,
82     vsgNullPtrErr = -3,
83     vsgInvalidDeviceErr = -2,
84     vsgDeviceNotFoundErr = -1,
85
86     vsgNoError = 0,
87
88     vsgAlreadyFlushed = 1,
89     vsgSettingClamped = 2
90 } VsgStatus;
91
92 #ifdef __cplusplus
93 extern "C" {
94 #endif
95
96 VSG_API const char* vsgGetAPIVersion();
97
98 VSG_API VsgStatus vsgGetDeviceList(int *serials, int *count);
99
100 //TODO:
101 VSG_API VsgStatus vsgGetDeviceList2(int *serials, VsgDeviceType *deviceTypes, int *deviceCount);
102
103 VSG_API VsgStatus vsgOpenDevice(int *handle);
104
105 VSG_API VsgStatus vsgOpenDeviceBySerial(int *handle, int serialNumber);

```

```

197
209 VSG_API VsgStatus vsgCloseDevice(int handle);
210
221 VSG_API VsgStatus vsgPreset(int handle);
222
235 VSG_API VsgStatus vsgRecal(int handle);
236
248 VSG_API VsgStatus vsgAbort(int handle);
249
259 VSG_API VsgStatus vsgGetSerialNumber(int handle, int *serial);
260
270 VSG_API VsgStatus vsgGetDeviceType(int handle, VsgDeviceType *deviceType);
271
281 VSG_API VsgStatus vsgGetFirmwareVersion(int handle, int *version);
282
292 VSG_API VsgStatus vsgGetCalDate(int handle, uint32_t *lastCalDate);
293
305 VSG_API VsgStatus vsgReadTemperature(int handle, float *temp);
306
324 VSG_API VsgStatus vsgSetRFOutputState(int handle, VsgBool enabled);
325
335 VSG_API VsgStatus vsgGetRFOutputState(int handle, VsgBool *enabled);
336
351 VSG_API VsgStatus vsgSetTimebase(int handle, VsgTimebaseState state);
352
362 VSG_API VsgStatus vsgGetTimebase(int handle, VsgTimebaseState *state);
363
379 VSG_API VsgStatus vsgSetTimebaseOffset(int handle, double ppm);
380
391 VSG_API VsgStatus vsgGetTimebaseOffset(int handle, double *ppm);
392
406 VSG_API VsgStatus vsgSetFrequency(int handle, double frequency);
407
419 VSG_API VsgStatus vsgGetFrequency(int handle, double *frequency);
420
434 VSG_API VsgStatus vsgSetSampleRate(int handle, double sampleRate);
435
445 VSG_API VsgStatus vsgGetSampleRate(int handle, double *sampleRate);
446
464 VSG_API VsgStatus vsgSetLevel(int handle, double level);
465
475 VSG_API VsgStatus vsgGetLevel(int handle, double *level);
476
492 VSG_API VsgStatus vsgSetAtten(int handle, int atten);
493
506 VSG_API VsgStatus vsgGetIQScale(int handle, double *iqScale);
507
523 VSG_API VsgStatus vsgSetIQOffset(int handle, int16_t iOffset, int16_t qOffset);
524
536 VSG_API VsgStatus vsgGetIQOffset(int handle, int16_t *iOffset, int16_t *qOffset);
537
551 VSG_API VsgStatus vsgSetDigitalTuning(int handle, VsgBool enabled);
552
562 VSG_API VsgStatus vsgGetDigitalTuning(int handle, VsgBool *enabled);
563
576 VSG_API VsgStatus vsgSetTriggerLength(int handle, double seconds);
577
587 VSG_API VsgStatus vsgGetTriggerLength(int handle, double *seconds);
588
608 VSG_API VsgStatus vsgSubmitIQ(int handle, float *iq, int len);
609
622 VSG_API VsgStatus vsgSubmitTrigger(int handle);
623
633 VSG_API VsgStatus vsgFlush(int handle);
634
645 VSG_API VsgStatus vsgFlushAndWait(int handle);
646
663 VSG_API VsgStatus vsgOutputWaveform(int handle, float *iq, int len);
664
684 VSG_API VsgStatus vsgRepeatWaveform(int handle, float *iq, int len);
685
695 VSG_API VsgStatus vsgOutputCW(int handle);
696
707 VSG_API VsgStatus vsgIsWaveformActive(int handle, VsgBool *active);
708
716 VSG_API VsgStatus vsgGetUSBStatus(int handle);
717
723 VSG_API void vsgEnablePowerSavingCpuMode(VsgBool enabled);
724
734 VSG_API const char* vsgGetErrorString(VsgStatus status);
735
736 #ifdef __cplusplus
737 } // Extern C
738 #endif
739
740 #endif

```

Index

VSG200_MAX_FREQ
vsg_api.h, 13

VSG200_MIN_FREQ
vsg_api.h, 13

VSG60_MAX_FREQ
vsg_api.h, 13

VSG60_MIN_FREQ
vsg_api.h, 13

vsg_api.h, 11

VSG200_MAX_FREQ, 13

VSG200_MIN_FREQ, 13

VSG60_MAX_FREQ, 13

VSG60_MIN_FREQ, 13

VSG_MAX_DEVICES, 13

VSG_MAX_IQ_OFFSET, 14

VSG_MAX_LEVEL, 14

VSG_MAX_SAMPLE_RATE, 13

VSG_MAX_TRIGGER_LENGTH, 14

VSG_MIN_IQ_OFFSET, 14

VSG_MIN_LEVEL, 13

VSG_MIN_SAMPLE_RATE, 13

VSG_MIN_TRIGGER_LENGTH, 14

vsgAbort, 18

vsgAlreadyFlushed, 16

VsgBool, 15

vsgCloseDevice, 17

vsgDeviceNotFoundErr, 16

VsgDeviceType, 14

VsgDeviceType200, 15

VsgDeviceType60, 15

vsgEnablePowerSavingCpuMode, 34

vsgFalse, 15

vsgFlush, 31

vsgFlushAndWait, 31

vsgGetAPIVersion, 16

vsgGetCalDate, 20

vsgGetDeviceList, 16

vsgGetDeviceType, 19

vsgGetDigitalTuning, 29

vsgGetErrorString, 34

vsgGetFirmwareVersion, 20

vsgGetFrequency, 25

vsgGetIQOffset, 28

vsgGetIQScale, 27

vsgGetLevel, 26

vsgGetRFOutputState, 22

vsgGetSampleRate, 25

vsgGetSerialNumber, 19

vsgGetTimebase, 23

vsgGetTimebaseOffset, 24

vsgGetTriggerLength, 30

vsgGetUSBStatus, 33

vsgInvalidDeviceErr, 16

vsgInvalidParameterErr, 15

vsgIsWaveformActive, 33

vsgNoError, 16

vsgNullPtrErr, 15

vsgOpenDevice, 16

vsgOpenDeviceBySerial, 17

vsgOutputCW, 33

vsgOutputWaveform, 32

vsgPreset, 18

vsgReadTemperature, 20

vsgRecal, 18

vsgRepeatWaveform, 32

vsgSetAtten, 27

vsgSetDigitalTuning, 28

vsgSetFrequency, 24

vsgSetIQOffset, 28

vsgSetLevel, 26

vsgSetRFOutputState, 22

vsgSetSampleRate, 25

vsgSetTimebase, 23

vsgSetTimebaseOffset, 23

vsgSettingClamped, 16

vsgSetTriggerLength, 29

VsgStatus, 15

vsgSubmitIQ, 30

vsgSubmitTrigger, 31

VsgTimebaseState, 15

vsgTimebaseStateExternal, 15

vsgTimebaseStateInternal, 15

vsgTrue, 15

vsgUsbXferErr, 15

vsgWaveformAlreadyActiveErr, 15

vsgWaveformNotActiveErr, 15

VSG_MAX_DEVICES
vsg_api.h, 13

VSG_MAX_IQ_OFFSET
vsg_api.h, 14

VSG_MAX_LEVEL
vsg_api.h, 14

VSG_MAX_SAMPLE_RATE
vsg_api.h, 13

VSG_MAX_TRIGGER_LENGTH
vsg_api.h, 14

VSG_MIN_IQ_OFFSET
vsg_api.h, 14

VSG_MIN_LEVEL
 vsg_api.h, 13
 VSG_MIN_SAMPLE_RATE
 vsg_api.h, 13
 VSG_MIN_TRIGGER_LENGTH
 vsg_api.h, 14
 vsgAbort
 vsg_api.h, 18
 vsgAlreadyFlushed
 vsg_api.h, 16
 VsgBool
 vsg_api.h, 15
 vsgCloseDevice
 vsg_api.h, 17
 vsgDeviceNotFoundErr
 vsg_api.h, 16
 VsgDeviceType
 vsg_api.h, 14
 VsgDeviceType200
 vsg_api.h, 15
 VsgDeviceType60
 vsg_api.h, 15
 vsgEnablePowerSavingCpuMode
 vsg_api.h, 34
 vsgFalse
 vsg_api.h, 15
 vsgFlush
 vsg_api.h, 31
 vsgFlushAndWait
 vsg_api.h, 31
 vsgGetAPIVersion
 vsg_api.h, 16
 vsgGetCalDate
 vsg_api.h, 20
 vsgGetDeviceList
 vsg_api.h, 16
 vsgGetDeviceType
 vsg_api.h, 19
 vsgGetDigitalTuning
 vsg_api.h, 29
 vsgGetErrorString
 vsg_api.h, 34
 vsgGetFirmwareVersion
 vsg_api.h, 20
 vsgGetFrequency
 vsg_api.h, 25
 vsgGetIQOffset
 vsg_api.h, 28
 vsgGetIQScale
 vsg_api.h, 27
 vsgGetLevel
 vsg_api.h, 26
 vsgGetRFOutputState
 vsg_api.h, 22
 vsgGetSampleRate
 vsg_api.h, 25
 vsgGetSerialNumber
 vsg_api.h, 19
 vsgGetTimebase
 vsg_api.h, 23
 vsgGetTimebaseOffset
 vsg_api.h, 24
 vsgGetTriggerLength
 vsg_api.h, 30
 vsgGetUSBStatus
 vsg_api.h, 33
 vsgInvalidDeviceErr
 vsg_api.h, 16
 vsgInvalidParameterErr
 vsg_api.h, 15
 vsgIsWaveformActive
 vsg_api.h, 33
 vsgNoError
 vsg_api.h, 16
 vsgNullPtrErr
 vsg_api.h, 15
 vsgOpenDevice
 vsg_api.h, 16
 vsgOpenDeviceBySerial
 vsg_api.h, 17
 vsgOutputCW
 vsg_api.h, 33
 vsgOutputWaveform
 vsg_api.h, 32
 vsgPreset
 vsg_api.h, 18
 vsgReadTemperature
 vsg_api.h, 20
 vsgRecal
 vsg_api.h, 18
 vsgRepeatWaveform
 vsg_api.h, 32
 vsgSetAtten
 vsg_api.h, 27
 vsgSetDigitalTuning
 vsg_api.h, 28
 vsgSetFrequency
 vsg_api.h, 24
 vsgSetIQOffset
 vsg_api.h, 28
 vsgSetLevel
 vsg_api.h, 26
 vsgSetRFOutputState
 vsg_api.h, 22
 vsgSetSampleRate
 vsg_api.h, 25
 vsgSetTimebase
 vsg_api.h, 23
 vsgSetTimebaseOffset
 vsg_api.h, 23
 vsgSettingClamped
 vsg_api.h, 16
 vsgSetTriggerLength
 vsg_api.h, 29
 VsgStatus
 vsg_api.h, 15

vsgSubmitIQ
 vsg_api.h, [30](#)
vsgSubmitTrigger
 vsg_api.h, [31](#)
VsgTimebaseState
 vsg_api.h, [15](#)
vsgTimebaseStateExternal
 vsg_api.h, [15](#)
vsgTimebaseStateInternal
 vsg_api.h, [15](#)
vsgTrue
 vsg_api.h, [15](#)
vsgUsbXferErr
 vsg_api.h, [15](#)
vsgWaveformAlreadyActiveErr
 vsg_api.h, [15](#)
vsgWaveformNotActiveErr
 vsg_api.h, [15](#)