VSG25 Programming Interface (API)

Programmers Reference Manual

Signal Hound VSG25 Application Programming Interface (API)
Programmers Reference Manual

35707 NE 86th Ave
La Center, WA 98629 USA
Phone 360.263.5006 • Fax 360.263.5007

Version 3.0
4/28/2015

Overview

This manual is a reference for the Signal Hound VSG25A vector signal generator (VSG) programming interface. The API provides a set of C routines used to control the VSG25A. The API is C ABI compatible, so it can be called from a number of other languages and environments such as Java, C#, Python, C++, Matlab, and Labview.

This manual will describe the requirements and knowledge needed to program to the API.

Contact Information

Please report any issues or bugs as soon as possible. We are also interested in your feedback and questions.

All programming and API questions should be directed to aj@signalhound.com

All hardware related questions should be directed to justin@teplus.com

Build/Version Notes

Two Windows builds are available in 32 and 64-bit. The builds are compiled with Visual Studio 2012. Distributing an application using this library will require distributing the Visual Studio 2012 redistributable libraries.

Version 1.0.0

Initial release

Requirements

Windows Development Requirements

- Windows 7/8 (The API is untested outside of these operating systems)
- Windows PC with USB 2.0 port.
- Windows C/C++ development tools/environment. Preferable Visual Studio 2008 or later.
- The vsg25_api.h API header file.
- The API files (vsg25_api.lib and vsg25_api.dll)

Theory of Operation

The API provides a set of routines necessary to configure the output of the VSG25 for all available output modulations.

The order of operation for any program interfacing the VSG25 is as follows …

1) Open a VSG25 with the sgOpenDevice or sgOpenDeviceBySerial functions.
2) Set the frequency and amplitude of the device.
3) Set the desired modulation type.
4) Close the device with the sgCloseDevice function.

Opening a Device

The API provides two routines for claiming a VSG25 for operation. The sgOpenDevice function opens the first device discovered connected to the PC via USB 2.0. The sgGetDeviceList and sgOpenDeviceBySerial are used to discover all available VSG25 devices connected to the PC and claim a specific one. Both open routines return a device handle which is used in all subsequent configuration function calls.

Configuring a Device

When a device is opened through the API the output of the device is initially in the off state. You must call a series of configuration functions in the correct order to set the desired state of the device.

Each time a new output state of the device is desired, the full configuration state must be set.

To see a number of examples of this, see the Appendix: Code Examples section.

Querying the Device Configuration

Depending on the parameters provided, the VSG25 API may not be able to set the device to the exact requested configuration. To account for this the API has a number of query operations which can be used to retrieve the exact values used to set the device. Of particular importance is retrieving the symbol clock rate after configuring any digital modulation as well as AM/FM analog modulation. Also some modulations such as pulse may not offer all combinations of configurations, and using the saQueryPulse function allows you to determine the configuration.

API Functions

sgGetDeviceList

```
sgStatus sgGetDeviceList(int deviceList[8], int *length)
```

Parameters

deviceList Pointer to an array of 8 integers. If the function returns successfully, the array will be populated with a list of all discovered device serial numbers.

length Pointer to an integer. If the function returns successfully, the integer length points to will contain the number of devices discovered.

Description

This function is used to discover all unopened devices. This function will return the serial numbers of up to 8 unopened devices. No device will be claimed when this function returns. The function can be used in conjunction with saOpenDeviceBySerial to target a specific device.

saOpenDevice

```
sgStatus sgOpenDevice(int *device)
```

Parameters

device Pointer to an integer. If successful the integer pointed to by *device* will contain a valid device number which can be used to identify a device for successive API function calls.

Description

This function will attempt to open the first VSG25 it detects. If a device is opened successfully a handle to the device will be returned through the *device* pointer which can be used for all future API function calls.

saOpenDeviceBySerial

```
sgStatus sgOpenDeviceBySerial(int *device, int serialNumber)
```

Parameters

device Pointer to an integer. If successful the integer pointed to by *device* will contain a valid device number which can be used to identify a device for successive API function calls.

serialNumber Serial number of a VSG25 unit to open.

Description

This function will attempt to discover and open the VSG25 device specified by the *serialNumber* parameter. If a device cannot be found matching the provided serial number, the function will return unsuccessful. If a device is opened successfully, a handle to the device will be returned through the *device* pointer which can be used to target that device for future API calls.

sgCloseDevice

```
sgStatus sgCloseDevice(int device)
```

Parameters

device Handle of the device to close.

Description

This function is called when you wish to terminate a connection with a device. Any resources the device has allocated will be freed and the USB 2.0 connection to the device will be closed. The device closed will be released and will be able to be opened again.

sgGetSerialNumber

```
sgStatus sgGetSerialNumber(int device, int *serialNumber)
```

Parameters

device Device handle.

serialNumber Pointer to an integer.

Description

This function returns the serial number to an open device. If this function returns successfully, the integer *serialNumber* points to will contain the serial number of the device specified.

sgSetFrequencyAmplitude

```
sgStatus sgSetFrequencyAmplitude(int device, double frequency, double
amplitude)
```

Parameters

device Device handle.

frequency Frequency in Hz.

amplitude RF output amplitude specified in dBm.

Description

Configure the frequency and amplitude of the specified device. If any output mode is active, this function disables the current output. The settings applied here do not take effect until a mode is set.

sgRFOff

```
sgStatus sgRFOff(int device)
```

Parameters

device Device handle.

Description

Disables the RF output of the specified device.

sgSetCW

```
sgStatus sgSetCW(int device)
```

Parameters

device Device handle.

Description

Instruct the device to output a CW signal.

sgSetAM

```
sgStatus sgSetAM(int device, double frequency, double depth, sgShape shape)
```

Parameters

device Device handle.

frequency AM frequency specified in Hz.

depth AM depth specified in percentage %. Can be set from 1% to 99%.

shape Specifies the modulation waveform shape.

Description

Configure the device to output an amplitude modulated signal.

sgSetFM

```
sgStatus sgSetFM(int device, double frequency, double deviation, sgShape
shape)
```

Parameters

device Device handle.

frequency FM frequency specified in Hz.

deviation FM waveform deviation specified in Hz.

shape Specifies the modulation waveform shape.

Description

Configure the device to output a frequency modulated signal.

sgSetPulse

```
sgStatus sgSetPulse(int device, double period, double width)
```

Parameters

device Device handle.

period Specify the time between rising pulse edges in seconds.

width Specifies the pulse width (“on” time) in seconds. *Width* must be less than *period*.

Description

Configure the device to output a pulse modulated signal with a fixed duty cycle where the duty cycle is

$$duty\ cycle = \frac{pulse\ width}{pulse\ period} * 100\%$$

See also saQueryPulse.

sgSetSweep

```
sgStatus sgSetSweep(int device, double time, double span)
```

Parameters

device Device handle.

time Specify the sweep duration in seconds.

span Specify the frequency span of the sweep in Hz.

Description

Configure the device to perform a continuous frequency sweep. The device will sweep a CW signal from the start to stop frequency in *time* seconds where

$$start = \left(center\ frequency - \frac{span}{2}\right) \qquad stop = \left(center\ frequency + \frac{span}{2}\right)$$

where the center frequency is specified in sgSetFrequencyAmplitude. The next sweep is performed immediately after the current sweep is finished.

sgSetMultitone

```
sgStatus sgSetMultitone(int device, int count, double spacing, double
notchWidth, sgMultiTonePhase phase)
```

Parameters

device Device handle.

count Specify the number of tones.

spacing Specify the spacing of the tones in Hz.

notchWidth Specify a notch bandwidth in Hz centered in the middle of the multi-tone signal.

phase Specify the phase relationship of the tones generated, `sgParabolic` for best signal-to-noise, or `sgRandom for` random (required for noise power ratio testing).

Description

The VSG25 is capable of generating complex multi-tone signals. There are 4 main parameters you can use to generate these signals. The *count* and *spacing* parameter specifies the number and spacing of the signals. The *notchWidth* parameter defines a region centered on the center frequency where tones are not generated. The *phase* parameter specifies how the phase of the tones are determined. For best dynamic range, choose a parabolic phase. For repeatability choose the random fixed seed phase.

sgSetASK

```
sgStatus sgSetASK(int device, double symbolRate, sgFilterType filterType,
double filterAlpha, double depth, int *symbols, int symbolCount);
```

Parameters

device Device handle.

symbolRate Specify the ASK symbol clock rate in samples per second.

sgFilterType Specify a filter to be applied to the ASK waveform.

filterAlpha Specify the roll-off factor of the filter selected. Acceptable inputs range from 0.0 to 1.0. This parameter is ignored if no filter is selected.

depth Specify the amplitude modulation depth in percentage. Acceptable values range between 1.0 and 99.0%.

symbols A pointer to an array of *symbolCount* integers representing the digital data to be modulated. Acceptable values for each sample are either 0 or 1. Only the least significant bit is used to determine the bit value. An array that is smaller than *symbolCount* will result in undefined behavior.

symbolCount Specify the number of symbols to be modulated. Specifies the minimum size of the array *symbols* points to. Acceptable inputs range from 1 to 512.

Description

Configure the device to modulate a user defined data set using amplitude-shift keyed modulation.

sgSetFSK

```
sgStatus sgSetFSK(int device, double symbolRate, sgFilterType filterType,
double filterAlpha, double modulationIndex, int *symbols, int symbolCount);
```

Parameters

device Device handle.

symbolRate Specify the FSK symbol clock rate in samples per second.

sgFilterType Specify a filter to be applied to the FSK waveform.

filterAlpha Specify the roll-off factor of the filter selected. Acceptable inputs range from 0.0 to 1.0. This parameter is ignored if no filter is selected.

modulationIndex Specify the occupied bandwidth/frequency deviation of the signal with the modulationIndex parameter. The modulation index is defined as

$$modulation\ index = \frac{2 * frequency\ deviation}{symbol\ clock\ rate}$$

symbols A pointer to an array of *symbolCount* integers representing the digital data to be modulated. Acceptable values for each sample are either 0 or 1. Only the least significant bit is used to determine the bit value. An array that is smaller than *symbolCount* will result in undefined behavior.

symbolCount Specify the number of symbols to be modulated. Specifies the minimum size of the array *symbols* points to. Acceptable inputs range from 1 to 512.

Description

Configure the device to modulate a user defined data set using frequency-shift keyed modulation.

sgSetPSK

```
sgStatus sgSetPSK(int device, double symbolRate, sgModulationType modType,
sgFilterType filterType, double filterAlpha, int *symbols, int symbolCount);
```

Parameters

device Device handle.

symbolRate Specify the FSK symbol clock rate in samples per second.

modType Specify a digital modulation.

sgFilterType Specify a filter to be applied to the FSK waveform.

filterAlpha Specify the roll-off factor of the filter selected. Acceptable inputs range from 0.0 to 1.0. This parameter is ignored if no filter is selected.

symbols A pointer to an array of *symbolCount* integers representing the digital data to be modulated. Only the least 'N' significant bits are used to determine the symbol value, where N is determined by the modulation type selected. An array that is smaller than *symbolCount* will result in undefined behavior.

symbolCount Specify the number of symbols to be modulated. Specifies the minimum size of the array *symbols* points to. Acceptable inputs range from 1 to 512.

Description

Configure the device to modulate a user defined data set using a PSK or QAM modulation.

sgSetCustomIQ

```
sgStatus sgSetCustomIQ(int device, double clockRate, double *IVals, double
*QVals, int length, int period)
```

Parameters

device Device handle.

clockRate Specify the sample clock rate in Hz.

IVals Pointer to an array of I-channel values of at least *length* doubles.

QVals Pointer to an array of Q-channel values of at least *length* doubles.

length Total number of I/Q output pairs.

period The period, in clocks, of the signal, including an optional idle period. This must be equal to or greater than *length*.

Description

Configures the device to generate a custom pattern. Set *period* equal to *length* for continuously looped waveforms. If *period* > *length*, the first (index=0) and last (index=length-1) I/Q samples must match. This value (typically 0, 0) will be held for (*period* – *length*) additional samples.

sgQueryPulse

```
sgStatus sgQueryPulse(int device, double *period, double *width)
```

Parameters

device Device handle.

period Pointer to a 64-bit floating point value.

width Pointer to a 64-bit floating point value.

Description

Retrieve the last pulse period and width used by the VSG25.

sgQuerySymbolClockRate

```
sgStatus sgQuerySymbolClockRate(int device, double *clock)
```

Parameters

device Device handle.

clock Pointer to a 64-bit floating point value.

Description

Retrieve the last symbol clock rate used by the device.

sgQueryClockError

```
sgStatus sgQueryClockError(int device, double *error)
```

Parameters

device Device handle.

error Pointer to a 64-bit floating point value.

Description

Retrieve the clock error, defined as fabs(ideal clock – actual clock) / ideal clock. Typically 0.0, but some combinations of settings will result in a small error. See saQuerySymbolClockRate to retrieve the actual internal clock rate.

sgGetStatusString

```
const char* sgGetStatusString(sgStatus status)
```

Parameters

status A *sgStatus* enumerated value.

Description

Returns an ascii status string for a given status code. This function is useful for debugging and providing readable error messages.

sgGetAPIVersion

```
const char* sgGetAPIVersion()
```

Parameters

None

Description

Returns the API version as a string. The returned string is of the form *major.minor.revision.* *Major/minor/revision* are not guaranteed to be a single decimal digit but are always separated by a '.' character. The string is null terminated. An example string is below …

['1' | '.' | '2' | '.' | '1' | '1' | '\0'] = "1.2.11"

Error Codes

sgNoError The function returned successfully.

sgNullPtrErr One or more pointer parameters passed was null.

sgInvalidDeviceHandle The device handle supplied is invalid. Either the handle supplied is outside the range of acceptable handle values [0-7], or the handle does not refer to an active device in the API.

sgUnableToFindDevice No device was found. Returned when calling the open routines and no devices were discovered or all available devices are already claimed by the API.

sgInvalidParameter One or more parameter passed is outside the range of acceptable inputs. Refer to the specific function in the API manual for more information.

sgSettingClamped One or more parameters was clamped to its acceptable input range. This is performed when the clamp does not change the desired requested functionality.

Appendix

Code Examples

Opening a Device

The following example demonstrates how to open a single device through the API .

```
int handle;
sgStatus openStatus = sgOpenDevice(&handle);
if(openStatus != sgNoError) {
      std::cout << sgGetStatusString(openStatus);
      return -1;
}

// Device is open
// The handle variable can be used for all future
//   API calls
```

Opening a Specific Device

The following example demonstrates how to discover and open a specific device by serial number.

```
int handle;
int serialnumbers[8];
int count;

sgGetDeviceList(serialnumbers, &count);
if(count == 0) {
      return -1;
}

if(serialNumberPresent(serialnumbers, count)) {
      sgOpenDeviceBySerial(&handle, desiredSerial);
```

```
} else {
      return -1;
}
```

Output a CW Signal

The following example demonstrates how to configure the device to output a CW signal. This example and all future examples do not perform error handling in attempt to maintain clarity. All examples show the process of opening the device. Note that this step is only required once per application launch, not every time you want to configure the device.

```
// Configure the device to output a -20 dBm CW signal
//  at 1GHz
int handle;
sgOpenDevice(&handle);
sgSetFrequencyAmplitude(handle, 1000.0e6, -20.0);
sgSetCW(handle);
```

Configure an Analog AM/FM Output

```
// Configure the device to output an FM waveform at
//  91.3MHz
int handle;
sgOpenDevice(&handle);
sgSetFrequencyAmplitude(handle, 91.3e6, -10.0);
// Configure a 12kHz sinusoidal FM signal
//  with a deviation of 100kHz
sgSetFM(handle, 12.0e3, 100.0e3, sgShapeSine);
```

Configure a Pulse Signal

```
// Configure a 40us pulse with a 50% duty cycle
//  in the ISM band at 915MHz
int handle;
sgOpenDevice(&handle);
sgSetFrequencyAmplitude(handle, 915.0e6, 10.0);
sgSetPulse(handle, 80.0e-6, 40.0-6);
// It might be important to determine the exact
//  pulse width and period used by the device
double period, width;
sgQueryPulse(handle, &period, &width);
```

Output a Multi-Tone Signal

```
// Configure a 16 tone multi-tone signal at 2.4GHz
// The tones are separated by 10kHz and no notch filter should be applied
// A parabolic phase is specified for the best dynamic range
int handle;
sgOpenDevice(&handle);
sgSetFrequencyAmplitude(handle, 2400.0e6, -10.0);
sgSetMultitone(handle, 16, 10.0e3, 0.0, sgParabolic);
```

Configure a Digital Modulation Waveform

```
// Configure the device to output a QPSK modulated signal
// Output the signal at 890 MHz, symbol rate of 3.84 MHz
// 128 symbols total with root raised cosine filter with
//   roll-off factor of 0.22
```

```
int handle;
sgOpenDevice(&handle);
sgSetFrequencyAmplitude(handle, 890.0e6, -10.0);
int symbols[128];
// Hypothetical function to set symbol values between 0-3
loadSymbols(symbols);
sgSetPSK(handle, 3840.0e3, sgModQPSK, sgRootRaisedCosine,
         0.22, symbols, 128);
```