

VAUNIX TECHNOLOGY CORPORATION



## **Lab Brick® LMS/BLX Signal Generators**

---

### **Linux USB API User Manual**

**Library Versions 1.09+**

**6/10/2026**

#### **NOTICE**

Vaunix has prepared this manual for use by Vaunix Company personnel and customers as a guide for the customized programming of Lab Brick products. The drawings, specifications, and information contained herein are the property of Vaunix Technology Corporation, and any unauthorized use or disclosure of these drawings, specifications, and information is prohibited; they shall not be reproduced, copied, or used in whole or in part as the basis for manufacture or sale of the equipment or software programs without the prior written consent of Vaunix Technology Corporation.

## Table of Contents

<b>1. Overview .....</b>	<b>2</b>
<b>2. Setting up the SDK .....</b>	<b>2</b>
<b>3. Using the SDK .....</b>	<b>2</b>
<b>4. Programming.....</b>	<b>3</b>
4.1 Overall Strategy and API Architecture .....	3
4.2 Status Codes.....	5
4.3 Functions – Setting up the environment & housekeeping .....	5
4.4 Functions – Selecting the Device .....	6
4.5 Functions – Setting parameters.....	7
4.6 Functions – Reading parameters .....	12

## 1. Overview

The Lab Brick LMS Series Signal Generators SDK for Linux supports developers who want to control Lab Brick LMS Series Signal Generators from Linux programs. For maximum compatibility, the SDK includes source code for C functions to find, initialize, and control the Signal Generators, along with header files and an example C program which demonstrates the use of the API. These functions are written to use the 'libusb' library which comes with most Linux distributions or is easily installed.

## 2. Setting up the SDK

Before you can use the SDK or try the sample program, you need to make sure you have libusb installed. You can retrieve source from the developer's site at <http://www.libusb.org>, or use your distribution's package installer. Look for a package that contains "libusb-dev" in the package name. For Debian and Ubuntu, "libusb-dev" should work. For Redhat and Fedora, look for "libusb-devel". If you have the library installed, "locate usb.h" should turn up an include file in some appropriate location (perhaps '/usr/include') and that file should have declarations for `usb_init()`, `usb_set_debug()`, and `usb_find_devices()` among others. Help forums exist for most distributions and someone on one of these forums can probably help you find the appropriate library. Contact us if you get stuck.

The SDK also uses the Posix thread functions found in the 'pthread' library. Again, most recent distributions will have this library preinstalled.

## 3. Using the SDK

The SDK consists of source code for the SDK functions, a .H header file for your C program, a sample C program, and a Makefile which demonstrates how to build your code to use the functions. Untar the SDK into a convenient place on your hard disk, and then copy these files into the directory of the executable program you are creating. Start by trying to build the sample (make all). If the build is successful, you're ready to add these functions to your own program. Add the header file (`lmsdrv.h`) to your project, and include it with the other header files in your program. Modify the make file by replacing the test program name with your program name. Or simply compile your program with the command line "`gcc -o test -lm -lpthread -lusb .c lmsusb.c`". In this case, the compiler will send the final output to 'test', link with the math, thread and usb libraries, and for source will use your program and the SDK source file, `lmsusb.c`.

## 4. Programming

### 4.1 Overall Strategy and API Architecture

The API provides functions for identifying how many and what type of Lab Brick LMS Signal Generators are connected to the system, initializing Signal Generators so that you can send them commands and read their state, functions to control the operation of the Signal Generators, and finally a function to close the software connection to the Signal Generators when you no longer need to communicate with it.

The API can be operated in test mode, where the functions will simulate normal operation but will not actually communicate with the hardware devices. This feature is provided as a convenience to software developers who may not have a Lab Brick Signal Generators with them but still want to be able to work on an applications program that uses the Lab Brick. Of course, it is important to make sure that the API is in its normal mode to access the actual hardware!

Before you do anything else, you MUST clear the SDK's internal structures. This is simply a call to `fnLMS_Init()` and only needs to be done once.

Be sure to call `fnLMS_SetTestMode(FALSE)`, unless of course you want the API to operate in its test mode. In test mode there will be 2 devices, an LMS-103 and an LMS-123.

The first step is to identify the Signal Generators connected to the system. Call the function `fnLMS_GetNumDevices()`<sup>1</sup> to get the number of Signal Generators attached to the system. Note that USB devices can be attached and detached by users at any time. If you are writing a program which needs to handle the situation where devices are attached or detached while the program is operating, you should periodically call `fnLMS_GetNumDevices()` to see if any new devices have been attached.

Allocate an array big enough to hold the device ids for the number of devices present. While you should use the `DEVID` type declared in `lmsdrv.h` it's just an array of units at this point. You may want to allocate an array large enough to hold `MAXDEVICES` device ids, so that you do not have to handle the case where the number of attached devices increases.

Call `fnLMS_GetDevInfo(DEVID *ActiveDevices)`, which will fill in the array with the device ids for each connected Signal Generator. The function returns an integer, which is the number of devices present on the machine.

---

<sup>1</sup> Usually it is a good idea to call `fnLMS_GetNumDevices()` at around 1 second intervals. While a short interval reduces the chances, it is still possible that the user will remove one device and replace it with another however, so to completely handle all the cases which can result from users hot plugging devices your application needs to check to see not only if the number of devices is different, but if the same number of devices are present, that they are not different devices.

The next step is to call `fnLMS_GetModelName(DEVID deviceID, char *ModelName)` with a null `ModelName` pointer to get the length of the model name, or just use a buffer that can hold `MAX_MODELNAME` chars. You can use the model name to identify the type of Signal Generators. Call `fnLMS_GetSerialNumber(DEVID deviceID)` to get the serial number of the Signal Generators. Based on that information, your program can determine which device to open.

Once you have identified the Signal Generators you want to send commands to, call `fnLMS_InitDevice(DEVID deviceID)` to actually open the device and get its various parameters like frequency setting, frequency sweep parameters, etc. After the `fnLMS_InitDevice` function has been completed you can use any of the get functions to read the settings of the Signal Generators.

To change one of the settings of the Signal Generators, use the corresponding set function. For example, to set the Signal Generators frequency, call `fnLMS_SetFrequencyEx(DEVID deviceID, unsigned int frequency)`. The first argument is the device id of the Signal Generator, the second is the desired output frequency. Frequency is specified in 10 Hz increments, where:

$\text{frequency} = \text{Frequency (Hz)} / 10$

For example, to specify an output frequency of 5.5 GHz,  $\text{frequency} = 550000000$ .

To set the output power level, call `fnLMS_SetPowerLevel(DEVID deviceID, int powerlevel)` with the output power level you want. The power level is encoded as the number of .25dB increments, with a resolution of .5dB. To set a power level of +5 dBm, for example, `powerlevel` would be 20. To set a power level of -20 dBm, `powerlevel` would be -80.

Note that the Lab Brick Signal Generators have a maximum and minimum settable power level. You can query the limits with calls to `fnLMS_GetMaxPwr(DEVID deviceID)` and `fnLMS_GetMinPwr(DEVID deviceID)`. Both functions use the same encoding of the `powerlevel` as the `SetPowerLevel` function.

When you are done with the device, call `fnLMS_CloseDevice(DEVID deviceID)`.

## 4.2 Status Codes

All of the set functions return a status code indicating whether an error occurred. The get functions normally return an integer value, but in the event of an error they will return an error code. The error codes can be distinguished from normal data by their numeric value, since all error codes have their high bit set, and they are outside of the range of normal data.

Functions that return a floating point result use specific, negative numeric values to indicate if an error occurred.

A separate function, `fnLMS_GetDeviceStatus(DEVID deviceID)` provides access to a set of status bits describing the operating state of the Signal Generators. This function can be used to check if a device is currently connected or open.

The values of the status codes are defined in the `lmsdrv.h` header file.

## 4.3 Functions – Setting up the environment & housekeeping

`void fnLMS_Init(void)`

Must be called once at the beginning of the user program to clear out the SDK's data structures and initialize the USB library functions.

`char* fnLMS_perror(LVSTATUS status)`

Useful for debugging your user program, `fnLMS_perror()` takes a returned LVSTATUS value from another function and returns a pointer to a descriptive string you can display on screen or log.

`char* fnLMS_pFloatError(float status)`

Useful for debugging your user program, `fnLMS_pFloatError()` takes a returned status value from another function and returns a pointer to a descriptive string you can display on screen or log for functions that return a floating point value.

`char* fnLMS_GetLibVersion(void)`

Returns a string which contains the version number of the SDK. If possible, call this function once when your program starts so you know the version number – that way, if you have questions or problems, you can include this version information in your question to us.

#### 4.4 Functions – Selecting the Device

`void fnLMS_SetTestMode(bool testmode)`

Set testmode to FALSE for normal operation. If testmode is TRUE the dll does not communicate with the actual hardware but simulates the basic operation of the dll functions. It does not simulate the operation of frequency sweeps generated by the actual hardware, but it does simulate the behavior of the functions used to set the parameters for sweeps.

`int fnLMS_GetNumDevices()`

This function returns a count of the number of connected Signal Generators.

`int fnLMS_GetDevInfo(DEVID *ActiveDevices)`

This function fills in the ActiveDevices array with the device ids for the connected Signal Generators. Note that the array must be large enough to hold a device id for the number of devices returned by `fnLMS_GetNumDevices`. The function also returns the number of active devices, which can, under some circumstances, be less than the number of devices returned in the previous call to `fnLMS_GetNumDevices`.

The device ids are used to identify each device and are used in the rest of the functions to select the device. Note that while the device ids may be small integers, and may, in some circumstances appear to be numerically related to the devices present, they should only be used as opaque handles.

`int fnLMS_GetModelName(DEVID deviceID, char *ModelName)`

This function is used to get the model name of the Signal Generators for ASCII clients. If the function is called with a null pointer, it returns just the length of the model name string. If the function is called with a non-null string pointer it copies the model name into the string and returns the length of the string. The string length will never be greater than the constant `MAX_MODELNAME` which is defined in `lmsdrv.h`. This function can be used regardless of whether or not the Signal Generator has been initialized with the `fnLMS_InitDevice` function.

`int fnLMS_GetSerialNumber(DEVID deviceID)`

This function is used to get the serial number of the Signal Generators. It can be called regardless of whether or not the Signal Generators has been initialized with the `fnLMS_InitDevice` function. If your system has multiple Signal Generators, your software should use each device's serial number to keep track of each specific device. Do not rely upon the order in which the devices appear in the table of active devices. On a typical system the individual Signal Generators will typically be found in the same order, but there is no guarantee that this will occur.

`int fnLMS_GetDeviceStatus(DEVID deviceID)`

This function can be used to obtain information about the status of a device, even before the device is initialized. (Note that information on the sweep activity of the device is not guaranteed to be available before the device is initialized.)

`int fnLMS_InitDevice(DEVID deviceID)`

This function is used to open the device interface to the Signal Generators and initialize the dll's copy of the device's settings. If the `fnLMS_InitDevice` function succeeds, then you can use the various `fnLMS_Get*` functions to read the Signal Generators's settings. This function will fail and return an error code if the Signal Generators has already been opened by another program.

`int fnLMS_CloseDevice(DEVID deviceID)`

This function closes the device interface to the Signal Generators. It should be called when your program is done using the Signal Generators.

#### 4.5 Functions – Setting parameters

`LVSTATUS fnLMS_SetFrequencyEx(DEVID deviceID, unsigned int frequency)`

This function is used to set the output frequency of the Signal Generators. Frequency is encoded as an unsigned integer number of 10 Hz steps:

$\text{frequency} = \text{Frequency (Hz)} / 10$

For example, to specify an output frequency of 6 GHz,  $\text{frequency} = 6000000$ . The value of frequency must be within the range of the attached Signal Generators, or an error will be returned.

`LVSTATUS fnLMS_SetPowerLevel(DEVID deviceID, int powerlevel);`

This function is used to set the output power level of the programmable Signal Generators. The power level is specified in .25dB units. The encoding is:  $\text{powerlevel} = \text{desired output power in dBm} / .25\text{dB}$

For example, if you want -7.5 dBm output power then you should set powerlevel to -30.

`LVSTATUS fnLMS_SetStartFrequencyEx(DEVID deviceID, unsigned int startfrequency)`

This function sets the frequency at the beginning of a frequency sweep. The encoding of startfrequency is the same as the `fnLMS_SetFrequencyEx` function. Note that the start frequency should be less than the end frequency when you want the frequency to step upwards during the sweep. For a sweep where the frequency decreases, then the start frequency should be larger than the end frequency.

LVSTATUS fnLMS\_SetEndFrequencyEx(DEVID deviceID, unsigned int endfrequency)

This function sets the frequency at the end of a frequency sweep. The encoding of endfrequency is the same as the fnLMS\_SetFrequencyEx function.

LVSTATUS fnLMS\_SetSweepTime(DEVID deviceID, int sweeptime)

This function sets the time duration of the frequency sweep for LMS devices. The sweeptime variable is encoded as a number of milliseconds. The minimum sweep time is 1 millisecond.

LVSTATUS fnLMS\_SetFrequencyStepEx(DEVID deviceID, unsigned int freqstep)

This function is used to set the frequency step size of the frequency sweep. The encoding of freqstep is the same as the fnLMS\_SetFrequencyEx function.

LVSTATUS fnLMS\_SetDwellTime(DEVID deviceID, int dwelltime)

This function is used to set the dwell time of the frequency sweep for BLX devices. The dwell time variable is encoded as a number of milliseconds. The minimum dwell time is 10 ms.

LVSTATUS fnLMS\_SetIdleTime(DEVID deviceID, int dwelltime)

This function is used to set the idle time of the frequency sweep for BLX devices. The idle time variable is encoded as a number of milliseconds. The minimum idle time is 0 ms.

LVSTATUS fnLMS\_SetRFOn(DEVID deviceID, bool on)

This function turns the RF stages of the Signal Generators on (on = TRUE) or off (on = FALSE).

LVSTATUS fnLMS\_SetUseInternalRef(DEVID deviceID, bool internal);

This function configures the Signal Generators to use the internal reference if internal = 1. If internal = 0, then the Signal Generators is configured to use an external frequency reference.

LVSTATUS fnLMS\_SetSweepDirection(DEVID deviceID, bool up)

This function is used to set the direction of the frequency sweep. To create a sweep with increasing frequency, set up = TRUE. Note that the sweep start frequency value must be less than the sweep end frequency value for a sweep with increasing frequency. For a sweep that decreases in frequency, the sweep start value must be greater than the sweep end value.

LVSTATUS fnLMS\_SetSweepMode(DEVID deviceID, bool mode)

This function is used to select either a single frequency sweep, or a repeating series of sweeps. If mode = TRUE then the sweep will be repeated, if mode = FALSE the sweep will only happen once.

LVSTATUS fnLMS\_SetSweepType(DEVID deviceID, bool swptype)

This function is used to select between a single directional frequency sweep, or a sweep which returns to its original frequency after each sweep. If swptype = TRUE then the sweep will be bidirectional, if swptype = FALSE the sweep will only go in one direction. For a bi-directional sweep a graph of frequency vs. time for a repeating sweep will appear like a triangle wave, for a non-bidirectional sweep, the graph of frequency vs. time will appear like a sawtooth wave.

LVSTATUS fnLMS\_StartSweep(DEVID deviceID, bool go)

This function is used to start and stop the frequency sweeps. If go = TRUE the Signal Generators will begin sweeping, FALSE stops the sweep. You must set the sweep parameters before calling this function to start the sweep.

LVSTATUS fnLMS\_SetFastPulsedOutput(DEVID deviceID, float pulseontime, float pulseretime, bool on)

This function is the preferred way to control the internal pulse modulation option. The pulseontime parameter is the length of the pulse on time in seconds. The pulseretime parameter is the length of the repetition period in seconds. Both values can range from 100 nanoseconds (0.100e-6) to 1000 seconds (1.0e3). Set on = TRUE to start the pulsed output modulation.

LVSTATUS fnLMS\_SetPulseOnTime(DEVID deviceID, float pulseontime)

This function is used to set the length of the RF pulse on time of the device's internal modulation for devices that support pulsed output modulation. The pulseontime parameter is the length of the pulse on time in seconds, with a 100-nanosecond minimum. This function is not recommended for general use. Instead use the fnLMS\_SetFastPulsedOutput function.

LVSTATUS fnLMS\_SetPulseOffTime(DEVID deviceID, float pulseofftime)

This function is used to set the length of the RF pulse off time of the device's internal modulation. The pulseofftime parameter is the length of the pulse off time in seconds, with a 100-nanosecond minimum. The repetition period of the pulse modulation is equal to pulseontime + pulseofftime. This function is not recommended for general use. Instead use the fnLMS\_SetFastPulsedOutput function.

LVSTATUS fnLMS\_EnableInternalPulseMod(DEVID deviceID, bool on)

This function is used to turn on and off the internal output modulation. If on = TRUE the Signal Generators will pulse its RF output on and off according to the values set for the pulse on time and pulse off time using either the fnLMS\_SetFastPulsedOutput function or the functions to set pulse on and off time directly. To stop the internal pulse modulation, set on = FALSE. Always disable internal pulse modulation before setting the pulse on and off time using the fnLMS\_SetPulseOnTime and fnLMS\_SetPulseOffTime functions.

LVSTATUS fnLMS\_SetUseExternalPulseMod(DEVID deviceID, bool external)

This function configures the Signal Generators to use the external pulse modulation input signal if external = TRUE. If external = FALSE, then the Signal Generators is configured to use the internal pulse modulation. Not all hardware configurations support an external pulse modulation input. Both the internal and external pulse modulation can operate at the same time, allowing more complex modulation patterns.

LVSTATUS fnLMS\_SetChirpMode(DEVID deviceID, bool mode)

This function enables chirp generation if mode is TRUE, and disables chirp generation if mode is FALSE. The StartSweep function is used to start the generation of a chirp or a repeated set of chirps.

The function fnLMS\_SetUseExternalSweepTrigger(DEVID deviceID, bool external) can be used to select the external trigger input as the source of triggers to start the chirps.

LVSTATUS fnLMS\_SetUseExternalSweepTrigger(DEVID deviceID, bool external)

Select the trigger input source to start the chirps. Uses external trigger input if external is TRUE, uses internal trigger input if external is FALSE.

LVSTATUS fnLMS\_SetChirpLength(DEVID deviceID, float chirplength)

This function sets the length of the chirp, with a minimum of 1 microsecond (1.0e-6), and a maximum of 50 milliseconds (50.0e-3).

LVSTATUS fnLMS\_SetChirpRepeatRate(DEVID deviceID, float chirp\_repeat)

This function sets the repeat rate of repeating chirps. The parameter is the time between chirp triggers, with a minimum of 10 microseconds (10.0e-6).

LVSTATUS fnLMS\_SetSequenceElement(DEVID deviceID, int index, unsigned int frequency, int powerlevel, bool pwr\_control)

This function is used to set an element in the sequence. The index runs from 0 to 49, frequency is an unsigned integer in 10Hz units, powerlevel is in 0.25 db units, and the pwr\_control flag is true when the sequence element controls both frequency and power level. If the pwr\_control flag is false then only the frequency is controlled.

LVSTATUS fnLMS\_SetSequenceStart(DEVID deviceID, int start)

This function is used to set the starting element of the sequence. The index runs from 0 to 49, and start is zero based.

LVSTATUS fnLMS\_SetSequenceCount(DEVID deviceID, int count)

This function is used to set the length of the sequence. The length of the sequence can be set to an integer number of elements from 0 to 50.

LVSTATUS fnLMS\_SetSequenceDwellTime(DEVID deviceID, int dwelltime)

This function is used to set the dwell time in milliseconds for each element of the sequence. The minimum dwell time for sequence elements is 10 ms.

LVSTATUS fnLMS\_SetSequenceIdleTime(DEVID deviceID, int idletime)

This function is used to set the idle time in milliseconds for the sequence. The minimum idle time for sequences is 0 ms.

LVSTATUS fnLMS\_StartSequence(DEVID deviceID, int control)

This function is used to control the operation of sequences. StartSequence takes one of five control values defined in lmsdrv.h: STOP\_SEQUENCE to stop the sequence, START\_SEQUENCE to play the sequence once, REPEAT\_SEQUENCE to start a repeating sequence, PAUSE\_SEQUENCE to pause the sequence, and RESUME\_SEQUENCE to resume the sequence.

LVSTATUS fnLMS\_SaveSettings(DEVID deviceID)

The Lab Brick Signal Generators can save their settings and then resume operating with the saved settings when they are powered up. Set the desired parameters, then use this function to save the settings.

## 4.6 Functions – Reading parameters

unsigned int fnLMS\_GetFrequencyEx(DEVID deviceID)

This function returns the current frequency setting of the selected device. When a sweep is active this value will change dynamically to reflect the current setting of the device. The return value is frequency as an unsigned integer in 10 Hz units.

unsigned int fnLMS\_GetStartFrequencyEx(DEVID deviceID)

This function returns the current frequency sweep starting value setting of the selected device. The return value is frequency as an unsigned integer in 10 Hz units.

unsigned int fnLMS\_GetEndFrequencyEx(DEVID deviceID)

This function returns the current frequency sweep end setting of the selected device. The return value is frequency as an unsigned integer in 10 Hz units.

unsigned int fnLMS\_GetFrequencyStepEx(DEVID deviceID, unsigned int freqstep)

This function is used to get the frequency step size of the frequency sweep. The return value is frequency step as an unsigned integer in 10 Hz units.

int fnLMS\_GetSweepTime(DEVID deviceID)

This function returns the current frequency sweep time in milliseconds. A one second sweep time, for example, would be returned as 1000.

int fnLMS\_GetSweepmode(DEVID deviceID)

This function returns the sweep mode from the Signal Generator as an integer. This returns the value 2 when the sweep is in repeat sweep mode, and 1 when the sweep is in single sweep mode.

int fnLMS\_GetSweepbidirectionalmode(DEVID deviceID)

This function returns an integer value which is 1 when bidirectional sweep mode is enabled for the Signal Generator, or 0 when bidirectional sweep mode is disabled.

int fnLMS\_GetDwellTime(DEVID deviceID)

This function returns the dwell time for sweep steps in milliseconds. A one second dwell time, for example, would be returned as 1000.

int fnLMS\_GetIdleTime(DEVID deviceID)

This function returns the dwell time for sweep steps in milliseconds. A one second dwell time, for example, would be returned as 1000.

int fnLMS\_GetPulseOnTime(DEVID deviceID)

This function returns the pulse on time, which is the length of time that RF output is enabled when internal pulse modulation is operating, in seconds.

int fnLMS\_GetPulseOffTime(DEVID deviceID)

This function returns the pulse off time, which is the length of time that RF output is disabled when internal pulse modulation is operating, in seconds. The pulse repetition period is equal to the pulse on time added to the pulse off time.

int fnLMS\_GetPulseMode(DEVID deviceID)

This function returns an integer value which is 1 when the Signal Generators internal pulse modulation is active, or 0 when the internal pulse modulation is off.

int fnLMS\_GetUseInternalPulseMod(DEVID deviceID)

This function returns an integer value which is 1 when the Signal Generators is configured to use its internal pulse modulation, or 0 when the external pulse modulation input is selected to control the output.

int fnLMS\_GetHasFastPulseMode(DEVID deviceID)

This function returns an integer value which is 1 when the Signal Generators has the internal pulse modulation option, or 0 when the option is not installed.

int fnLMS\_GetRF\_On(DEVID deviceID)

This function returns an integer value which is 1 when the Signal Generators is “on”, or 0 when the Signal Generators has been set “off” by the fnLMS\_SetRFOn function.

int fnLMS\_GetUseInternalRef(DEVID deviceID)

This function returns an integer value which is 1 when the Signal Generators is configured to use its internal frequency reference. It returns a value of 0 when the Signal Generators is configured to use an external frequency reference.

`int fnLMS_GetUseExtSweepTrigger(DEVID deviceID)`

This function returns an integer value which is 1 when the external sweep trigger is enabled for the Signal Generator. It returns a value of 0 when the external sweep trigger is disabled for the signal generator.

`int fnLMS_GetPLLLock(DEVID deviceID)`

This function returns an integer value which is 1 when the Signal Generator is phase locked to the reference, 0 when it is not phase locked to the reference.

`int fnLMS_GetPowerLevel(DEVID deviceID)`

This function returns the current power level setting as an integer number of .25 dB units. As an example, an output power level of +3 dBm would result in the value 12 being returned, while an output power level of +3.5 dBm would result in the value 14 being returned. The output power resolution is .5 dB.

`int fnLMS_GetMaxPwr(DEVID deviceID)`

This function returns the maximum output power level that the Signal Generators can provide, encoded in the same format as the `fnLMS_GetPowerLevel` function. For Signal Generators with +10 dBm maximum output power level this function returns the integer value 40. This is a read only value.

`int fnLMS_GetMinPwr(DEVID deviceID)`

This function returns the minimum output power level that the Signal Generators can provide, encoded in the same format as the `fnLMS_GetPowerLevel` function. Typically, this value is a negative number. For example, a device with -45 dBm minimum output power would return an integer value of -180. This is a read only value.

`unsigned int fnLMS_GetMaxFreqEx(DEVID deviceID)`

This function returns the maximum output frequency that the device can provide. The value is represented as an unsigned integer in 10 Hz units.

`unsigned int fnLMS_GetMinFreqEx(DEVID deviceID)`

This function returns the minimum output frequency that the device can provide. The value is represented as an unsigned integer in 10 Hz units.

int fnLMS\_GetHasChirpMode(DEVID deviceID)

This function returns an integer value which is 1 if the Signal Generator supports chirp generation, 0 otherwise

float fnLMS\_GetChirpLength(DEVID deviceID)

This function gets the length of the chirp in seconds.

float fnLMS\_GetChirpRepeatRate(DEVID deviceID)

This function gets the repeat rate in seconds of repeating chirps.

unsigned int fnLMS\_GetSeqElementFrequency(DEVID deviceID, int index)

This function is used to get the frequency value for an element in the sequence. The index runs from 0 to 49, and the frequency is returned as an unsigned integer in 10Hz units.

int fnLMS\_GetSeqElementPower(DEVID deviceID, int index)

This function is used to get the power value for element in the sequence. The index runs from 0 to 49, and the power level value is returned as an integer in 0.25 db units.

int fnLMS\_GetSeqElementPwrControl(DEVID deviceID, int index)

This function is used to get the power control value for element in the sequence. The index runs from 0 to 49, and the pwr\_control flag is returned as an integer and is 1 when the sequence element controls both frequency and power level. If the pwr\_control flag is 0 then only the frequency is controlled.

int fnLMS\_GetSequenceStart(DEVID deviceID)

This function is used to get the starting element of the sequence. The returned index runs from 0 to 49 and is zero based.

int fnLMS\_GetSequenceCount(DEVID deviceID)

This function is used to get the length of the sequence. The length of the sequence is returned as an integer number of elements from 0 to 50.

int fnLMS\_GetSequenceDwellTime(DEVID deviceID)

This function is used to get the dwell time in milliseconds for each element of the sequence.

int fnLMS\_GetSequenceIdleTime(DEVID deviceID)

This function is used to get the idle time in milliseconds for each element of the sequence.