

VAUNIX TECHNOLOGY CORPORATION



# **Lab Brick® LPS Series Programmable Phase Shifter**

---

## **Windows USB API User Manual**

**Revision -B**

**9/5/2025**

### **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>3</b>
<b>2. Using the SDK .....</b>	<b>3</b>
<b>3. Programming.....</b>	<b>4</b>
3.1 Overall Strategy and API Architecture .....	4
3.2 Status Codes.....	5
3.3 Functions – Setting up the Environment .....	6
3.4 Functions – Selecting the Device .....	6
3.5 Functions – Setting parameters .....	8
3.6 Functions – Reading parameters .....	12

## 1. Overview

The Lab Brick Programmable Phase Shifter Windows SDK supports developers who want to control Lab Brick LPS series devices from Windows programs, or who want to control the Phase shifters from LabVIEW or other National Instruments programming environments. The SDK includes a dll which provides a Win32/Win64 API to find, initialize, and control the phase shifters, along with header files and an example Win32/Win64 C program which demonstrates the use of the API.

## 2. Using the SDK

The SDK consists of both 32- and 64-bit dll files along with this documentation, a C style header file, a library file for linking to the dll, and example programs. Unzip the SDK into a convenient place on your hard disk and then copy the dll and library file into the directory of the executable program you are creating. Add the header file (VNX\_LPS\_api.h) to your project and include it with the other header files in your program. Make sure that the linker directives include the path of the library file.

### 3. Programming

#### 3.1 Overall Strategy and API Architecture

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

The API can be operated in test mode, where the functions will simulate many aspects of 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 programmable phase shifter 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 in order to access the actual hardware.

Be sure to call `fnLPS_SetTestMode(FALSE)`, unless of course you want the API to operate in its test mode. In test mode there will be one LPS-802 device.

The first step is to identify the phase shifters connected to the system. Call the function `fnLPS_GetNumDevices()` to get the number of LPS phase shifters 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 `fnLPS_GetNumDevices()` to see if any new devices have been attached.<sup>1</sup>

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 `VNX_LPS_api.h` it's just an array of uints at this point. You may want to just 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 `fnLPS_GetDevInfo(DEVID *ActiveDevices)`, which will fill in the array with the device ids for each connected phase shifter. The function returns an integer, which is the number of devices present on the machine.

The next step is to call `fnLPS_GetModelNameA(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 phase shifter in the event that other models of the LPS phase shifters are offered in the future.

---

<sup>1</sup> Usually it is a good idea to call `fnLPS_GetNumDevices()` at around 1 to 5 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.

Call `fnLPS_GetSerialNumber(DEVID deviceID)` to get the serial number of the phase shifter. Based on that information, your program can determine which device to open.

Once you have identified the phase shifter you want to send commands to, call `fnLPS_InitDevice(DEVID deviceID)` to actually open the device and get its various parameters like phase angle setting, ramp parameters, etc. After the `fnLPS_InitDevice` function has completed you can use any of the get functions to read the settings of the phase shifter. It is best to use the `InitDevice` function once, at the beginning of operation, and then close the device when your program has completed its operations with the Lab Brick device.

To change one of the settings of the phase shifter, use the corresponding set function. Before selecting a phase shift, or starting a ramp or profile, set the operating frequency of the phase shifter based on the center frequency of the signal you want to phase shift.

For example, to set a 30 degree phase shift for a 5.6 GHz signal, first call `fnLPS_SetWorkingFrequency(DEVID deviceID, int frequency)` with the frequency in 100KHz units (56000 in this example). Then call `fnLPS_SetPhaseAngle(DEVID deviceID, int phase)` with the desired phase shift (30 in this example).

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

### 3.2 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.

A separate function, `fnLPS_GetDeviceStatus(DEVID deviceID)` provides access to a set of status bits describing the operating state of the phase shifter. 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 `VNX_LPS_api.h` header file.

### 3.3 Functions – Setting up the Environment

`char* fnLPS_perror(LVSTATUS status)`

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

`int fnLPS_GetLibVersion(void)`

Returns an integer 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.

`void fnLPS_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 ramps or profile operations generated by the actual hardware, but it does simulate the behavior of the functions used to get and set the parameters for the ramps and profile operations.

### 3.4 Functions – Selecting the Device

For multi-channel LPS devices, the functions act on the currently selected channel, except for `fnLPS_SetPhaseAngleQ`, which combines channel selection and setting the phase angle value in one function, and `fnLPS_SetPhaseAngleMC`, `fnLPS_StartRampMC`, and `fnLPS_StartProfileMC` functions which act on the channels selected in their channel selection masks.

VNX\_LPS\_API `void fnLPS_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 ramps or profile operations generated by the actual hardware, but it does simulate the behavior of the functions used to get and set the parameters for the ramps and profile operations.

VNX\_LPS\_API `int fnLPS_GetNumDevices()`

This function returns a count of the number of connected LPS phase shifters.

VNX\_LPS\_API int fnLPS\_GetDevInfo(DEVID \*ActiveDevices)

This function fills in the ActiveDevices array with the device ids for the connected phase shifters. Note that the array must be large enough to hold a device id for the number of devices returned by fnLPS\_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 fnLPS\_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.

VNX\_LPS\_API int fnLPS\_GetModelNameA(DEVID deviceID, char \*ModelName)

This function retrieves the model name of the phase shifter in ANSI (single-byte) character format. If the function is called with a null pointer, it returns the length of the model name string. If it is called with a non-null string pointer, it copies the model name into the provided buffer and returns the length of the string. The returned length will never exceed the constant MAX\_MODELNAME defined in VNX\_LPS\_api.h. This function can be used whether or not the phase shifter has been initialized with the fnLPS\_InitDevice function.

VNX\_LPS\_API int fnLPS\_GetModelNameW(DEVID deviceID, wchar\_t \*ModelName)

This function retrieves the model name of the phase shifter in Unicode (wide character) format. If the function is called with a null pointer, it returns the length of the model name string. If it is called with a non-null string pointer, it copies the model name into the provided buffer and returns the length of the string. The returned length will never exceed the constant MAX\_MODELNAME defined in VNX\_LPS\_api.h. This function can be used whether or not the phase shifter has been initialized with the fnLPS\_InitDevice function.

VNX\_LPS\_API int fnLPS\_GetSerialNumber(DEVID deviceID)

This function is used to get the serial number of the phase shifter. It can be called regardless of whether or not the phase shifter has been initialized with the fnLPS\_InitDevice function. If your system has multiple Lab Brick LPS phase shifters, 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 phase shifters will typically be found in the same order, but there is no guarantee that this will occur.

VNX\_LPS\_API int fnLPS\_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 dynamic activity of the device, such as whether a ramp or profile is active is not guaranteed to be available before the device is initialized.)

VNX\_LPS\_API int fnLPS\_InitDevice(DEVID deviceID)

This function is used to open the device interface to the phase shifter and initialize the dll's copy of the device's settings. If the fnLPS\_InitDevice function succeeds, then you can use the various fnLPS\_Get\* functions to read the phase shifter's settings. This function will fail, and return an error code if the phase shifter has already been opened by another program.

VNX\_LPS\_API int fnLPS\_CloseDevice(DEVID deviceID)

This function closes the device interface to the phase shifter. It should be called when your program is done using the Lab Brick.

### 3.5 Functions – Setting parameters

VNX\_LPS\_API LVSTATUS fnLPS\_SetChannel(DEVID deviceID, int channel)

This function is used to set the channel to be controlled. The channel defaults to channel 1 in the absence of a setting.

VNX\_LPS\_API LPSTATUS fnLPS\_SetPhaseAngle(DEVID deviceID, int phase)

This function is used to set the desired phase shift at the current working frequency. The first argument is the device id of the phase shifter, the second is the desired phase angle in degrees.

VNX\_LPS\_API LVSTATUS fnLPS\_SetPhaseAngleQ(DEVID deviceID, int phase, int channel)

This function uses the same deviceID as all of the other API functions, a phase angle value in degrees, and a channel number. Note that this function will leave the channel set to whatever value it is called with, so if you want to use other functions on other channels you need to call the SetChannel function before doing so.

VNX\_LPS\_API LPSTATUS fnLPS\_SetPhaseAngleMC(DEVID deviceID, int phase, unsigned long long channelmask)

This function is used to set the desired phase shift across multiple channels simultaneously. The first argument is the device id of the phase shifter, the second is the desired phase angle in degrees, and the third is a bitmask specifying which channels to apply the phase to.



VNX\_LPS\_API LPSTATUS fnLPS\_SetWorkingFrequency(DEVID deviceId, int frequency)

This function is used to select the working frequency for the phase shifter. The frequency is specified as an integer in 100 KHz units. A working frequency of 1 GHz, for example, would be represented as 10,000. The working frequency range of an LPS phase shifter can be determined by using the fnLPS\_GetMaxWorkingFrequency and fnLPS\_GetMinWorkingFrequency functions.

VNX\_LPS\_API LPSTATUS fnLPS\_SetRampStart(DEVID deviceId, int rampstart)

This function sets the starting phase angle for a ramp, rampstart is in degrees. The ramp can be unidirectional, like a sawtooth, or bi-directional. A bi-directional ramp returns to the starting value at its conclusion. A uni-directional ramp has one section, a bi-directional ramp has two sections.

VNX\_LPS\_API LPSTATUS fnLPS\_SetRampEnd(DEVID deviceId, int rampstop)

This function sets the ending phase angle for a ramp, rampstop is in degrees. For a uni-directional ramp, the rampstop angle is the phase angle at the end of the ramp. For a bi-directional ramp, the rampstop angle is the phase angle at the end of the first section of the ramp and becomes the starting angle for the second section of the ramp after the hold time between the first and second sections.

VNX\_LPS\_API LPSTATUS fnLPS\_SetPhaseAngleStep(DEVID deviceId, int phasestep)

This function sets the magnitude of the step in phase angle which occurs during the first section of the ramp. The phase step is always a positive value, ranging from 1 to the maximum phase shift possible for the device, even when the direction of a ramp section causes the slope of the ramp to be negative.

VNX\_LPS\_API LPSTATUS fnLPS\_SetPhaseAngleStepTwo(DEVID deviceId, int phasestep2)

This function sets the magnitude of the step in phase angle which occurs during the second section of the ramp. The phase step is always a positive value, ranging from 1 to the maximum phase shift possible for the device, even when the direction of a ramp section causes the slope of the ramp to be negative.

VNX\_LPS\_API LPSTATUS fnLPS\_SetDwellTime(DEVID deviceId, int dwelltime)

This function sets the length of time that the phase shifter will dwell on each phase angle step while it is generating the first section of the ramp. The dwelltime variable is encoded as the number of milliseconds to dwell at each phase angle. The minimum dwell time is 1 millisecond.

VNX\_LPS\_API LPSTATUS fnLPS\_SetDwellTimeTwo(DEVID deviceID, int dwelltime2)

This function sets the length of time that the phase shifter will dwell on each phase angle step while it is generating the second section of the ramp. The dwelltime variable is encoded as the number of milliseconds to dwell at each phase angle. The minimum dwell time is 1 millisecond.

VNX\_LPS\_API LPSTATUS fnLPS\_SetIdleTime(DEVID deviceID, int idletime)

This function sets the length of time that the phase shifter will wait after completing a ramp before beginning another ramp when the ramp mode is set to continuous operation. The idletime variable is encoded as the number of milliseconds to wait. The minimum idle time is 0 milliseconds.

VNX\_LPS\_API LPSTATUS fnLPS\_SetHoldTime(DEVID deviceID, int holdtime)

This function sets the length of time that the phase shifter will wait after completing the first section of a ramp before beginning the second section of the ramp in a bi-directional ramp. The holdtime variable is encoded as the number of milliseconds to wait. The minimum hold time is 0 milliseconds.

VNX\_LPS\_API LPSTATUS fnLPS\_SetRampDirection(DEVID deviceID, bool up)

This function is used to set the direction of the first section of the ramp, with up TRUE for an increasing slope. The direction is automatically reversed in the second section for a bi-directional ramp.

VNX\_LPS\_API LPSTATUS fnLPS\_SetRampMode(DEVID deviceID, bool mode)

This function is used to select whether the device produces a single ramp, or a repeating series of ramps. For a single ramp, set mode to FALSE. This setting should be modified before starting a ramp.

VNX\_LPS\_API LPSTATUS fnLPS\_SetRampBidirectional(DEVID deviceID, bool  
bidir\_enable)

This function is used to select bi-directional ramps, when bidir\_enable is TRUE, or unidirectional ramps, when bidir\_enable is FALSE. This setting should be modified before starting a ramp.

VNX\_LPS\_API LPSTATUS fnLPS\_StartRamp(DEVID deviceID, bool go)

This function is used to start a ramp or stop a ramp which is in operation. Calling the function with go TRUE will start a ramp, calling the function with go FALSE will stop a ramp.

LPSTATUS fnLPS\_StartRampMC(DEVID deviceID, int mode, int chmask, bool deferred)

This function is used to start or stop a ramp on multiple channels. The second argument (mode) is composed of the sweep command byte flags SWP\_DIRECTION, SWP\_CONTINUOUS, SWP\_ONCE, and SWP\_BIDIR. The third argument is a bitmask specifying which channels to apply the profile to. The fourth argument (deferred) is not currently used by the function.

VNX\_LPS\_API LPSTATUS fnLPS\_SetProfileElement(DEVID deviceID, int index, int phase)

The Lab Brick phase shifter can generate a phase shift profile of up to 50 (PROFILE\_MAX) or 1,000 (PROFILE\_MAX\_RAM) phase angle values depending on LPS model. This function is used to load the profile elements into the Lab Brick. The index variable specifies the location in the profile array, and ranges from 0, the start of the profile, to PROFILE\_MAX - 1, the end of the profile. For LPS-802-8 and LPS-402-8 devices, the index variable ranges from 0 to PROFILE\_MAX\_RAM - 1, as these devices have 1,000 element RAM based profiles. The phase variable specifies the phase angle in degrees for the selected element in the profile.

VNX\_LPS\_API LPSTATUS fnLPS\_SetProfileCount(DEVID deviceID, int profilecount)

This function sets the number of elements in the profile that will be used. It must be greater than zero and less than PROFILE\_MAX, the maximum profile length. For LPS-802-8 and LPS-402-8 devices, the maximum profile length is PROFILE\_MAX\_RAM.

VNX\_LPS\_API LPSTATUS fnLPS\_SetProfileDwellTime(DEVID deviceID, int dwelltime)

This function sets the length of time that the phase shifter will dwell on each element in the profile. The dwelltime variable is encoded as the number of milliseconds to dwell at each element. The minimum dwell time is 1 millisecond.

VNX\_LPS\_API LPSTATUS fnLPS\_SetProfileIdleTime(DEVID deviceID, int idletime)

This function sets the length of time that the phase shifter will wait after completing a profile before beginning another profile when the profile is set to repeat operation. The idletime variable is encoded as the number of milliseconds to wait. The minimum idle time is 0 milliseconds.

VNX\_LPS\_API LPSTATUS fnLPS\_StartProfile(DEVID deviceID, int mode)

This function is used to start or stop a profile. If mode is 0, the profile will be stopped. If mode is 1 the profile will be generated once. If mode is 2, the profile will repeat.

VNX\_LPS\_API LPSTATUS fnLPS\_StartProfileMC(DEVID deviceID, int mode, int chmask, bool delayed)

This function is used to start or stop a profile on multiple channels. If mode is 0, the profile will be stopped. If mode is 1, the profile will be generated once. If mode is 2, the profile will repeat. The third argument is a bitmask specifying which channels to apply the profile to. The fourth argument (delayed) is not currently used by the function.

VNX\_LPS\_API LVSTATUS fnLPS\_SaveSettings(DEVID deviceID)

The Lab Brick phase shifters 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.

### 3.6 Functions – Reading parameters

VNX\_LPS\_API int fnLPS\_GetPhaseAngle(DEVID deviceID)

This function returns the last reported phase angle of the phase shifter. In general, the phase shifter reports its phase angle when a new phase angle is generated in a ramp or profile, as long as the dwell time is longer than approximately 50 milliseconds. Phase angle is reported in degrees. Note that the updating of the phase angle variable is asynchronous with respect to calls to this function.

VNX\_LPS\_API int fnLPS\_GetWorkingFrequency(DEVID deviceID)

This function returns the current working frequency in 100KHz units.

VNX\_LPS\_API int fnLPS\_GetRampStart(DEVID deviceID)

This function returns the ramp starting phase angle in degrees.

VNX\_LPS\_API int fnLPS\_GetRampEnd(DEVID deviceID)

This function returns the ramp ending phase angle in degrees.

VNX\_LPS\_API int fnLPS\_GetDwellTime(DEVID deviceID)

This function returns the dwell time for the first section of a ramp in milliseconds.

VNX\_LPS\_API int fnLPS\_GetDwellTimeTwo(DEVID deviceID)

This function returns the dwell time for the second section of a ramp in milliseconds.

VNX\_LPS\_API int fnLPS\_GetIdleTime(DEVID deviceID)

This function returns the idle time between repetitions of a ramp in milliseconds.

VNX\_LPS\_API int fnLPS\_GetHoldTime(DEVID deviceID)

This function returns the hold time between the first and second sections of a ramp in milliseconds.

VNX\_LPS\_API int fnLPS\_GetPhaseAngleStep(DEVID deviceID)

This function returns the magnitude of each step in the first section of the ramp.

VNX\_LPS\_API int fnLPS\_GetPhaseAngleStepTwo(DEVID deviceID)

This function returns the magnitude of each step in the second section of the ramp.

VNX\_LPS\_API int fnLPS\_GetProfileElement(DEVID deviceID, int index)

This function returns the profile array element selected by the index variable. The index variable ranges from 0 to PROFILE\_MAX - 1. For new LPS devices, the index variable ranges from 0 to PROFILE\_MAX\_RAM - 1. The returned value is in degrees.

VNX\_LPS\_API int fnLPS\_GetProfileCount(DEVID deviceID)

This function returns the length of the profile. The returned value ranges from 1 to PROFILE\_MAX. For new LPS devices, the return value ranges from 1 to PROFILE\_MAX\_RAM.

VNX\_LPS\_API int fnLPS\_GetProfileDwellTime(DEVID deviceID)

This function returns the dwell time for each element of a profile in milliseconds.

VNX\_LPS\_API int fnLPS\_GetProfileIdleTime(DEVID deviceID)

This function returns the idle time between repetitions of a profile in milliseconds.

VNX\_LPS\_API int fnLPS\_GetProfileIndex(DEVID deviceID)

This function returns the last reported index for the active element in a profile. In general, the phase shifter reports the index when a new element in a profile is generated, as long as the dwell time is longer than approximately 50 milliseconds. Note that the updating of the index variable is asynchronous with respect to calls to this function.

VNX\_LPS\_API int fnLPS\_GetMaxPhaseShift(DEVID deviceID)

This function returns the maximum phase shift that the Lab Brick is capable of, in degrees.

VNX\_LPS\_API int fnLPS\_GetMinPhaseShift(DEVID deviceID)

This function returns the minimum phase shift that the Lab Brick is capable of, in degrees.

VNX\_LPS\_API int fnLPS\_GetMinPhaseStep(DEVID deviceID)

This function returns the smallest phase shift increment that the Lab Brick is capable of, in degrees.

VNX\_LPS\_API int fnLPS\_GetMaxWorkingFrequency (DEVID deviceID)

This function returns the maximum working frequency for the Lab Brick phase shifter in 100KHz units.

VNX\_LPS\_API int fnLPS\_GetMinWorkingFrequency (DEVID deviceID)

This function returns the minimum working frequency for the Lab Brick phase shifter in 100KHz units.