

VAUNIX TECHNOLOGY CORPORATION



# Lab Brick® Programmable Attenuator

---

## Linux Ethernet API User Manual

Library Versions 1.1+

6/8/2026

### 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>3</b>
<b>2. Using the SDK .....</b>	<b>3</b>
<b>3. Programming.....</b>	<b>3</b>
3.1 Overall Strategy and API Architecture .....	3
3.2 Status Codes.....	4
3.3 Functions – Setting up the Environment .....	5
3.4 Functions – Selecting the Device .....	5
3.5 Functions – Setting parameters on the Attenuator.....	7
3.6 Functions – Reading parameters from the Attenuator .....	11

## 1. Overview

The LabBrick Digital Attenuator SDK for Linux supports developers who want to control LabBrick Digital Attenuators from Linux programs through the Ethernet port. For maximum compatibility, the SDK includes source code for C functions to initialize and control the attenuators, along with header files and an example C program which demonstrates the use of the API. These functions are written to use the standard Ethernet Sockets library which comes with most Linux distributions.

## 2. Using the SDK

The SDK consists of source code for the SDK functions, a .H header file for your programs, sample C programs (LDA\_Eth\_CLI.c and ldatestapp.c), a Python example script (ldaccli.py), 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 (tar -xvf <lda\_sdk\_name>.tar), 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 (ldadrvr.h) to your project, and include it with the other header files in your program. Modify the make file by replacing 'test' with your program name. Or simply compile your program with the command line “gcc -o test -lm -lpthread -lusb <yourprogram>.c ldasocket.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, 'ldasocket.c'.

## 3. Programming

### 3.1 Overall Strategy and API Architecture

The API provides functions for initializing attenuators so that you can send them commands and read their state, functions to control the operation of the attenuators, and finally a function to close the software connection to the attenuator when you no longer need to communicate with the device.

The API can be operated in a 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 attenuator 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!

Call fnLDA\_Init(void) to prepare the library for use.

Be sure to call fnLDA\_SetTestMode(FALSE), unless of course you want the API to operate in its test mode. In test mode there will be 2 attenuator devices.

For each of the LDA devices that you have, determine its IP address, and create a string with the IP address in the form:

```
char myLDA[] = {"192.168.100.10"};
```

For variables you want to get from the LDA device allocate a variable to receive the result from the library (this API uses a call by reference for parameters). For example, to get the maximum number of channels you could allocate an integer value MaxChannels to receive the respdata:

```
int MaxChannels;
```

And then pass the address of that variable to the library function:

```
fnLDA_GetMaxChannels(myLDA, &MaxChannels);
```

Once you have selected the attenuator you want to send commands to, call `fnLDA_InitDevice(char* deviceip)` to actually open the device and get its various parameters like attenuation setting, attenuation ramp parameters, etc. After the `fnLDA_InitDevice` function has completed, you can use any of the get functions to read the settings of the attenuator. This function generates a lot of traffic to the device, so it should be used once at the beginning of your use of the attenuator.

The next step is to call `fnLDA_CheckDeviceReady(char* deviceip)` to check if the device is ready and available.

You can call `fnLDA_GetModelName(char* deviceip, char *respdata)`, using a buffer that can hold MAX\_MODELNAME chars to obtain the model name to identify the type of attenuator.

Call `fnLDA_GetSerialNumber(char* deviceip, int* respdata)` to get the serial number of the attenuator. Based on that information, your program can determine which device to open.

To change one of the settings of the attenuator, use the corresponding set function. For example, to set the attenuation level, call `fnLDA_SetAttenuation(char* deviceip, int attenuation)`. The first argument is the IP address of the attenuator, the second is the value of the attenuation you want to set. For this command, the attenuation is specified in .05 dB units, so 10 dB of attenuation is represented as 200, 6 dB of attenuation is represented as 120, and .1 dB, the minimum attenuation increment, is represented as 2.

When you are done with the device, call `fnLDA_CloseDevice(char* deviceip)`.

### 3.2 Status Codes

All of the functions return a status code indicating whether an error occurred. See the `ldadrvr.h` file for its definition.

### 3.3 Functions – Setting up the Environment

`void fnLDA_Init(void)`

This function prepares the library for use. Be sure to call this first at the start of your program before using any of the other functions available in this API.

`char* fnLDA_LibVersion(void)`

Returns a string which contains the version number of the library.

`int fnLDA_GetLibVersion(void)`

This function returns the version of the library, encoded as Major. Minor version in the lower 16 bits. Example: LDA\_DLLVERSION 0x00000101 // Version 1.1

`void fnLDA_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 attenuation ramps generated by the actual hardware, but it does simulate the behavior of the functions used to set the parameters for the ramps.

### 3.4 Functions – Selecting the Device

`STATUS_REPORT_T fnLDA_InitDevice(char* deviceip)`

This function is used to open the device interface to the attenuator and initialize the library's copy of the device's settings. If the `fnLDA_InitDevice` function succeeds, then you can use the various `fnLDA_Get*` functions to read the attenuator's settings. For optimum performance you should open the device interface at the beginning of use of the device, and close it when your program has completed. Repeatedly opening and closing a device will incur a performance penalty.

`STATUS_REPORT_T fnLDA_CloseDevice(char* deviceip)`

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

`STATUS_REPORT_T fnLDA_CheckDeviceReady(char* deviceip)`

This function call checks to see if the device is in a state where it is ready to receive commands. Returns 0 if the device is ready, 1 if not.

STATUS\_REPORT\_T fnLDA\_GetModelName(char\* deviceip, char \*respdata)

This function is used to get the model name of the attenuator. The function copies the model name into the string pointed to by respdata.

STATUS\_REPORT\_T fnLDA\_GetSerialNumber(char\* deviceip, int\* respdata )

This function is used to get the serial number of the attenuator.

STATUS\_REPORT\_T fnLDA\_GetIPMode(DEVID deviceID, int\* respdata)

This function retrieves the IP mode of the synthesizer and stores it in respdata. 0 = Static, 1 = DHCP.

STATUS\_REPORT\_T fnLDA\_GetIPAddress(DEVID deviceID, char\* respdata)

Gets the device's IP address string and stores it in respdata.

STATUS\_REPORT\_T fnLDA\_GetNetmask(DEVID deviceID, char\* respdata)

Gets the device's subnet mask string and stores it in respdata.

STATUS\_REPORT\_T fnLDA\_GetGateway(DEVID deviceID, char\* respdata)

Gets the device's gateway address string and stores it in respdata.

### 3.5 Functions – Setting parameters on the Attenuator

For multi-channel LDA devices, the functions act on the currently selected channel, except for `fnLDA_SetAttenuationQ`, which combines channel selection and setting the attenuation value in one function, and the `fnLDA_StartRampMC` and `fnLDA_StartProfileMC` functions which act on the channels selected in their channel selection masks.

`STATUS_REPORT_T fnLDA_SetChannel(char* deviceip, 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.

`STATUS_REPORT_T fnLDA_SetWorkingFrequency(char* deviceip, int frequency)`

This function is used to set the midband working frequency of the attenuator to optimize attenuation accuracy. The frequency setting is encoded as an integer using 100 kHz units. The encoding is:

$$\text{Frequency (MHz)} / 10$$

For example, to specify a working frequency of 1500 MHz, `frequency = 15,000`

`STATUS_REPORT_T fnLDA_SetAttenuation(char* deviceip, int attenuation)`

This function is used to set the attenuation level of the programmable attenuator. The attenuation setting is encoded as an integer where each increment represents .05db of attenuation. The encoding is:

$$\text{attenuation} * .05\text{db} = \text{Attenuation in dB}$$

For example, `attenuation = 100` for 5 dB of attenuation, 2 for .1 dB of attenuation, and 2400 for 120 dB attenuation.

`STATUS_REPORT_T fnLDA_SetAttenuationQ(char* deviceip, int Attenuation, int Channel)`

This function uses the same deviceID as all the other API functions, an attenuation value in .05 dB steps, and a channel number (1 to N). 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. The attenuation setting is encoded as an integer where each increment represents .05db of attenuation. The encoding is:

$$\text{attenuation} * .05\text{db} = \text{Attenuation in dB}$$

For example, `attenuation = 100` for 5 dB of attenuation, 2 for .1 dB of attenuation, and 2400 for 120 dB attenuation.

STATUS\_REPORT\_T fnLDA\_SetRampStart(char\* deviceip, int rampstart)

This function sets the attenuation level at the beginning of an attenuation ramp or sweep. The encoding of rampstart, the attenuation level, is the same as the fnLDA\_SetAttenuation function.

STATUS\_REPORT\_T fnLDA\_SetRampEnd(char\* deviceip, int rampstop)

This function sets the attenuation level at the end of an attenuation ramp or sweep. The encoding of rampstop, the attenuation level, is the same as the fnLDA\_SetAttenuation function.

STATUS\_REPORT\_T fnLDA\_SetAttenuationStep(char\* deviceip, int attenuationstep)

This function sets the size of the attenuation step that will be used to generate the attenuation ramp or sweep. The encoding of attenuationstep, is the same as the fnLDA\_SetAttenuation function.

STATUS\_REPORT\_T fnLDA\_SetAttenuationStepTwo(char\* deviceip, int attenuationstep2)

This function sets the size of the attenuation step that will be used to generate the attenuation ramp or sweep during the second phase of a bidirectional sweep. The encoding of attenuationstep2, is the same as the fnLDA\_SetAttenuation function.

STATUS\_REPORT\_T fnLDA\_SetDwellTime(char\* deviceip, int dwelltime)

This function sets the length of time that the attenuator will dwell on each attenuation step while it is generating the attenuation ramp. The dwelltime variable is encoded as the number of milliseconds to dwell at each level. The minimum dwell time is 1 millisecond.

STATUS\_REPORT\_T fnLDA\_SetDwellTimeTwo(char\* deviceip, int dwelltime2)

This function sets the length of time that the attenuator will dwell on each attenuation step while it is generating the attenuation ramp. The dwelltime2 variable is encoded as the number of milliseconds to dwell at each level. The minimum dwell time is 1 millisecond.

STATUS\_REPORT\_T fnLDA\_SetIdleTime(char\* deviceip, int idletime)

This function sets the length of time that the attenuator will wait at the end of an attenuation ramp before beginning the ramp again when the ramp mode is set to SWP\_REPEAT. The idletime variable is encoded as the number of milliseconds to dwell at each level. The minimum idle time is 0 milliseconds.



STATUS\_REPORT\_T fnLDA\_SetHoldTime(char\* deviceip, int holdtime)

This function sets the time delay between the first and second phases of a ramp. The holdtime variable is encoded as the number of milliseconds between phases. The minimum hold time is 0 milliseconds.

STATUS\_REPORT\_T fnLDA\_SetRampDirection(char\* deviceip, bool up)

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

STATUS\_REPORT\_T fnLDA\_SetRampMode(char\* deviceip, bool mode)

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

STATUS\_REPORT\_T fnLDA\_SetRampBidirectional(char\* deviceip, bool bidir\_enable)

This function selects bidirectional ramps. For a bidirectional ramp the attenuation changes from the start to end value in the first phase, and then back to the start value in the second phase.

STATUS\_REPORT\_T fnLDA\_StartRamp(char\* deviceip, bool go)

This function is used to start and stop the attenuation ramps. If go = TRUE the attenuator will begin sweeping, FALSE stops the sweep.

STATUS\_REPORT\_T fnLDA\_StartRampMC(char\* deviceip, int mode, int chmask, bool deferred)

This function is used to start and stop the attenuation ramps on the specified channel. The channels are selected based on the bits set in chmask, with channel 1 corresponding to the least significant bit.

For example, a chmask value of 0x000000001A will start profiles on channels 2, 4, and 5.

If go = TRUE the attenuator will begin ramping, FALSE stops the ramp.

STATUS\_REPORT\_T fnLDA\_SetProfileElement(char\* deviceip, int index, int attenuation)

This function sets the value of a profile element. The index runs from zero to the maximum profile length minus 1. The attenuation value is encoded in .05db steps.

STATUS\_REPORT\_T fnLDA\_SetProfileCount(char\* deviceip, 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.

STATUS\_REPORT\_T fnLDA\_SetProfileIdleTime(char\* deviceip, int idletime)

This function sets the idle time after a profile is played before the profile is played again in repeating profile mode.

STATUS\_REPORT\_T fnLDA\_SetProfileDwellTime(char\* deviceip, int dwelltime)

This function sets the time duration of each element in the profile during playback. The dwelltime is specified in milliseconds.

STATUS\_REPORT\_T fnLDA\_StartProfile(char\* deviceip, int mode)

This function starts the playback of a profile. A mode value of 1 plays the profile once, a mode value of 2 plays the profile repeatedly.

STATUS\_REPORT\_T fnLDA\_StartProfileMC(char\* deviceip, int mode, int chmask, bool delayed)

This function starts the playback of a profile on one or more specified channels. The channels are selected based on the bits set in chmask, with channel 1 corresponding to the least significant bit. For example, a chmask value of 0x000000001A will start profiles on channels 2, 4, and 5. A mode value of 1 plays the profile once, a mode value of 2 plays the profile repeatedly. The delayed argument is reserved for future use.

STATUS\_REPORT\_T fnLDA\_SetRFOn(char\* deviceip, bool on)

This function allows rapid switching of the attenuator from its set value “on” (on = TRUE) to its maximum attenuation (on = FALSE).

STATUS\_REPORT\_T fnLDA\_SaveSettings(char\* deviceip)

The Lab Brick attenuators 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 from the Attenuator

`STATUS_REPORT_T fnLDA_GetChannel(char* deviceip, int* respdata)`

This function returns the current active channel of the selected device.

`STATUS_REPORT_T fnLDA_GetMaxChannels(char* deviceip, int* respdata);`

This function returns the number of attenuation channels available.

`STATUS_REPORT_T fnLDA_GetWorkingFrequency(char* deviceip, int* respdata)`

This function returns the current working frequency of the selected device.

`STATUS_REPORT_T fnLDA_GetMinWorkingFrequency(char* deviceip, int* respdata)`

This function returns the Minimum working frequency of the selected device in 100 kHz units.

`STATUS_REPORT_T fnLDA_GetMaxWorkingFrequency(char* deviceip, int* respdata)`

This function returns the Maximum working frequency of the selected device in 100 kHz units.

`STATUS_REPORT_T fnLDA_GetAttenuation(char* deviceip, int* respdata)`

This function returns the current attenuation setting of the selected device. When an attenuation ramp is active this value will change dynamically to reflect the current setting of the device. The return value is in .05 dB units.

`STATUS_REPORT_T fnLDA_GetMinAttenuation(char* deviceip, int* respdata)`

This function returns the minimum attenuation value that the device can provide. In general, this value is 0 dB for the programmable attenuators. Since future products may have different capabilities, your software should use this function to obtain the minimum attenuation possible.

`STATUS_REPORT_T fnLDA_GetMaxAttenuation(char* deviceip, int* respdata)`

This function returns the maximum attenuation value that the device can provide. For the LDA-802 series programmable attenuators this value is 120 dB, which is 2400 .05 dB units. Since future products may have different maximum attenuation capabilities your software should use this function to obtain the maximum attenuation possible.

STATUS\_REPORT\_T fnLDA\_GetRampStart(char\* deviceip, int\* respdata)

This function returns the current attenuation ramp start value setting of the selected device. The return value is in .05 dB units.

STATUS\_REPORT\_T fnLDA\_GetRampEnd(char\* deviceip, int\* respdata)

This function returns the current attenuation ramp end setting of the selected device. The return value is in .05 dB units.

STATUS\_REPORT\_T fnLDA\_GetDwellTime(char\* deviceip, int\* respdata)

This function returns the current dwell time for each step on the attenuation ramp in milliseconds. A one second dwell time, for example, would be returned as 1000.

STATUS\_REPORT\_T fnLDA\_GetDwellTimeTwo(char\* deviceip, int\* respdata)

This function returns the current dwell time for each step on the second phase of a bidirectional attenuation ramp in milliseconds. A one second dwell time, for example, would be returned as 1000.

STATUS\_REPORT\_T fnLDA\_GetIdleTime(char\* deviceip, int\* respdata)

This function returns the idle time, which is the delay between attenuation ramps when the device is in the repeating ramp mode, in milliseconds.

STATUS\_REPORT\_T fnLDA\_GetHoldTime(char\* deviceip, int\* respdata)

This function returns the hold time, which is the delay between attenuation ramps when the device is in the bidirectional ramp mode, in milliseconds.

STATUS\_REPORT\_T fnLDA\_GetAttenuationStep(char\* deviceip, int\* respdata)

This function returns the current attenuation step size setting of the selected device. The return value is in .05 dB units, so for example an attenuation step of 5db would be represented by a return value of 100.

STATUS\_REPORT\_T fnLDA\_GetAttenuationStepTwo(char\* deviceip, int\* respdata)

This function returns the current attenuation step size setting of the selected device during the second phase of a bidirectional ramp. The return value is in .05 dB units, so for example an attenuation step of 5db would be represented by a return value of 100.

`STATUS_REPORT_T fnLDA_GetRF_On(char* deviceip, int* respdata)`

This function returns an integer value which is 1 when the attenuator is “on”, or 0 when the attenuator has been set “off” by the `fnLDA_SetRFOn` function. Note that the function does not attempt to interpret attenuation settings as either “on” or “off”, so if you set the attenuation level to 120 dB, (attenuation = 2400) the output signal level would be the same as if you had used the `fnLDA_SetRFOn` function with the `on = FALSE`, but this function would not return 0.

`STATUS_REPORT_T fnLDA_GetProfileElement(char* deviceip, int index, int* respdata);`

This function gets the value of a profile element. The index runs from zero to the maximum profile length minus 1. `PROFILE_MAX` is currently 100. The attenuation value is encoded in .05db steps.

`STATUS_REPORT_T fnLDA_GetProfileCount(char* deviceip, int* respdata);`

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.

`STATUS_REPORT_T fnLDA_GetProfileMaxLength(char* deviceip, int* respdata);`

This function returns the maximum length profile available for the programmable attenuator.

`STATUS_REPORT_T fnLDA_GetProfileDwellTime(char* deviceip, int* respdata);`

This function gets the time duration of each element in the profile during playback. The dwelltime is specified in milliseconds.

`STATUS_REPORT_T fnLDA_GetProfileIdleTime(char* deviceip, int* respdata);`

This function gets the idle time after a profile is played before the profile is played again in repeating profile mode.

`STATUS_REPORT_T fnLDA_GetFeatures(char* deviceip, int* respdata);`

This function returns a bit vector with bits set to indicate the available features. See `ldadrvr.h` for definitions. Legacy devices have a zero value for the feature vector.