

VAUNIX TECHNOLOGY CORPORATION



# **Lab Brick® LSW Series RF Switch**

---

## **Linux Ethernet API User Manual**

Revision B

**8/20/2025**

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

## 1. Overview

The Lab Brick RF Switch Linux SDK supports developers who want to control Lab Brick RF Switch over Ethernet from Linux programs. For maximum compatibility, the SDK includes source code for C functions to initialize and control the switch, 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 C program, a sample C program (lswtestapp.c) 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 -xv <lsw_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 (lswdrv.h) to your project and include it with the other header files in your program. Modify the make file by replacing 'lswtestapp.c' with your program name. Or simply compile your program with the command line "`gcc -o LSWtest -lm -lpthread -lusb <yourprogram>.c LSWhid.c`". In this case, the compiler will send the final output to LSWtest, link with the math, thread and other necessary libraries, and for source will use your program and the SDK source file, lswsocket.c'.

## 3. Programming

### 3.1 Overall Strategy and API Architecture

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

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 RF switch 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 `fnLSW_Init()` and only needs to be done once.

Be sure to call `fnLSW_SetTestMode(FALSE)`, unless of course you want the API to operate in its test mode. In test mode, 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 LabBrick RF Switch with them, but still want to be able to work on an applications program that uses the LabBrick. Of course, it is important to make sure that the API is in its normal mode in order to access the actual hardware!

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

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

For variables you want to get from the LSW 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 switch RF output, you could allocate an integer value rfOutput to receive the respdata:

```
int rfOutput;
```

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

```
fnLSW_GetSwitchRFoutput(myLSW, swindex, &rfOutput);
```

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

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

You can call `fnLSW_GetModelName(char* deviceip, char* respdata)`, using a buffer that can hold `MAX_MODELNAME` chars to obtain the model name to identify the type of switch.

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

To change one of the settings of the switch, use the corresponding set function. For example, to set the output state for a specified switch index, call `fnLSW_SetSwitchRFoutput(char* deviceip, int swindex, LSW_SWPORT_T swport)`. The first argument is the device ID of the RF switch, the second argument is the switch index corresponding to the expansion bus index of the desired switch (1 for single-unit devices), and the third argument is the RF outputs. The RF outputs are numbered sequentially from 1 to numSwitches, where numSwitches is the number of switches for the device returned by the `fnLSW_GetNumSwitches` function.

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

### 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.

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

### 3.3 Functions – Setting up the Environment & Housekeeping

`void fnLSW_Init(void)`

This function will be used to initialize LSW device data structures.

`void fnLSW_SetTestMode(bool testmode)`

Set testmode to FALSE for normal operation. If testmode is TRUE the library does not communicate with the actual hardware but simulates the basic operation of the library functions.

`char* fnLSW_perror(LVSTATUS status)`

Useful for debugging your user program, `fnLSW_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* fnLSW_LibVersion(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.

### 3.4 Functions – Selecting the Device

`int fnLSW_InitDevice(char* deviceip)`

This function is used to open the device interface socket connection over ethernet to the RF Switch and initialize the library's copy of the device's settings. If the `fnLSW_InitDevice` function succeeds, then you can use the various `fnLSW_Get*` functions to read the RF Switch's settings. This function will fail, and return an error status if the RF Switch has already been opened by another program.

`int fnLSW_CloseDevice(char* deviceip)`

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

`int fnLSW_CheckDeviceReady(char* deviceip)`

This function will be used to check whether the device is ready to get/set the parameters of the LSW RF Switch device.

### 3.5 Functions – Setting parameters

```
int fnLSW_SetSwitchRFoutput(char* deviceip, int swindex, LSW_SWPORT_T swport)
```

This function is used to set the Switch RF output state for the corresponding switch index. The first argument is the device ID of the RF switch, the second argument is the switch index corresponding to the expansion bus index of the desired switch (1 for single-unit devices), and the third argument is the RF outputs. The RF outputs are numbered sequentially from 1 to numSwitches, where numSwitches is the number of switches for the device returned by the fnLSW\_GetNumSwitches function.

```
int fnLDA_SaveSettings(char* deviceip)
```

The LabBrick RF switch can save their settings 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

All Get function calls take two arguments one pointer pointing to the device IP string and the other is the response data pointer.

```
int fnLSW_GetModelName(char* deviceip, char *respdata)
```

This function is used to get the model name of the RF Switch.

```
int fnLSW_GetSerialNumber(char* deviceip, int* respdata)
```

This function is used to get the serial number of the RF Switch.

```
int fnLSW_GetSoftwareVersion(char* deviceip, char* respdata)
```

This function is used to read the software version of the device.

```
int fnLSW_GetIPMode(char* deviceip, int* respdata)
```

This function is used to read the IP mode configuration of the device. Response data “0” represents the “Static” mode, “1” represents the “DHCP” mode.

```
int fnLSW_GetIPAddress(char* deviceip, char* respdata)
```

This function is used to read the IP address of the device.

```
int fnLSW_GetNetmask(char* deviceip, char* respdata)
```

This function is used to read the netmask of the device.

`int fnLSW_GetGateway(char* deviceip, char* respdata)`

This function is used to read the gateway address of the device.

`int fnLSW_GetNumSwitches(char* deviceip, int* respdata)`

This function returns the base number of switches in the selected device. This is a read only value.

`int fnLSW_GetMaxSwitchDevices(char* deviceip, int* respdata)`

This function is used to read the max RF switches device.

`int fnLSW_GetSwitchRFoutput(char* deviceip, int swindex, int* respdata)`

This function is used to get the Switch RF output state for the corresponding switch index. The first argument is the device ID of the RF switch, the second argument is the switch index corresponding to the expansion bus index of the desired switch (1 for single-unit devices).