

Vaunix Technology Corporation



# Lab Brick Family of RF Switches

---

## API INSTRUCTIONS

**Certification**

Vaunix Technology Corporation certifies that this product met its published specifications at the time of shipment from the factory.

**Warranty**

Lab Brick Digital Attenuators are warranted against defects in material and workmanship for a period of one year from the date of shipment.

**LIMITATION OF WARRANTY**

The foregoing warranty does not apply to connectors that have failed due to normal wear. Also, the warranty does not apply to defects resulting from improper or inadequate maintenance by the Buyer, unauthorized modification or misuse, or operation outside of the environmental specifications of the product.

No other warranty is expressed or implied, and the remedies provided herein are the Buyer's sole and exclusive remedies. Vaunix Technology Corporation shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory.

**NOTICE**

Vaunix has prepared this manual for use by Vaunix Company personnel and customers as a guide for the proper installation, operation, and maintenance of Vaunix equipment and computer programs. 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.

<b>API Instructions</b>	
<b>Lab Brick RF Switch API</b>	

## LabBrick RF Switch API

### Overview

The LabBrick RF Switch SDK for Linux supports developers who want to control LabBrick RFSwitches from Linux programs. For maximum compatibility, the SDK includes source code for C functions to find, initialize, and control the switches, 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. Many distributions which use a kernel 2.4 or newer already have this library installed.

### Setting up for 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.

### 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 (LSWtest.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 Vaunix_LSW_Vxx.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 (LSWhid.h) to your project and include it with the other header files in your program. Modify the make file by replacing 'LSWtest' 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 LSW'test', link with the math, thread and usb libraries, and for source will use your program and the SDK source file, 'LSWhid.c'.

### Overall Strategy and API architecture

The API provides functions for identifying how many and what type of LabBrick RF Switches are connected to the system, initializing the switches so that you can send them

commands and read their state, functions to control the operation of the switches, and finally a function to close the software connection to each switch when you no longer need to communicate with it.

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 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!

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 there will be 2 devices, an LSW-602PDT and an LSW-602P4T.

The first step in talking to the devices is to identify the RF Switches connected to the system. Call the function `fnLSW_GetNumDevices()` to get the number of RF Switches 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 `fnLSW_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 `LSWhid.h` it's just an array of unsigned ints 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 `fnLSW_GetDevInfo(DEVID *ActiveDevices)`, which will fill in the array with the device ids for each connected RF Switch. 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 `fnLSW_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 `fnLSW_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 RF Switch. Call `fnLSW_GetSerialNumber(DEVID deviceID)` to get the serial number of the RF Switch. Based on that information, your program can determine which device to open.

Once you have identified the RF Switch you want to send commands to, call `fnLSW_InitDevice(DEVID deviceID)` to actually open the device and get its various parameters like the number of switches it has, etc. After the `fnLSW_InitDevice` function has completed you can use any of the get functions to read the settings of the RF Switch.

To change one of the settings of the RF Switch, use the corresponding set function. For example, to set a switch selection, call `fnLSW_SetSwitch (DEVID deviceID, int inputselect)`. The first argument is the device id of the RF Switch the second is the desired switch selection. The RF outputs are numbered sequentially from 1 to 4 for the LSW-602P4T. For the LSW-602PDT the output 1 or 2 can be selected.

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

### 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, `fnLSW_GetDeviceStatus(DEVID deviceID)` provides access to a set of status bits describing the operating state of the RF Switch. 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 `LSWhid.h` header file.

### Functions – Setting up the environment & housekeeping

`void fnLSW_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* 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* fnDA_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.

### Functions – Selecting the Device

`void fnLSW_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 dynamic operation of the actual hardware, but it does simulate the behavior of the functions used to set and get the parameters in the device. Thus API calls which start switch patterns, or pulsed mode switching, will not cause the same changes in status variables as actual hardware would.

`int fnLSW_GetNumDevices()`

This function returns a count of the number of connected LabBrick RF Switch devices.

`int fnLSW_GetDevInfo(DEVID *ActiveDevices)`

This function fills in the ActiveDevices array with the device ids for the connected RF Switches. Note that the array must be large enough to hold a device id for the number of devices returned by `fnLSW_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 `fnLSW_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 fnLSW_GetModelName(DEVID deviceID, char *ModelName)`

This function is used to get the model name of the RF Switch. 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 `LSWhid.h`. This function can be used regardless of whether or not the RF Switch has been initialized with the `fnLSW_InitDevice` function.

`int fnLSW_GetSerialNumber(DEVID deviceID)`

This function is used to get the serial number of the RF Switch. It can be called regardless of whether or not the RF Switch has been initialized with the `fnLSW_InitDevice` function. If your system has multiple RF Switches, 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 RF Switches will typically be found in the same order, but there is no guarantee that this will occur.

`int fnLSW_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 is not guaranteed to be available before the device is initialized.)

`int fnLSW_InitDevice(DEVID deviceID)`  
This function is used to open the device interface 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 code if the RF Switch has already been opened by another program.

`int fnLSW_CloseDevice(DEVID deviceID)`  
This function closes the device interface to the RF Switch. It should be called when your program is done using the RF Switch.

### Functions – Setting parameters on the switch

`int fnLSW_SetSwitch (DEVID deviceID, int select)`  
This function is used to set the position of the switch. The first argument is the device id of the RF Switch, the second is the desired switch selection. The RF outputs are numbered sequentially from 1 to 4 for the LSW-602P4T. For the LSW-602PDT the output 1 or 2 can be selected.

`int fnLSW_SetUseExternalControl (DEVID deviceID, bool external);`  
This function is used to select internal or external control of the RF Switches. If external is TRUE, then the Lab Brick RF Switch will be controlled by the external control signal input or inputs.

`int fnLSW_SetPattern(DEVID deviceID, int num_entries, int sw_select[], int holdtime[])`  
This function sets the parameters for a switch pattern. A switch pattern consists of a set of pattern elements, where each element defines a switch setting and a hold time. When the pattern is activated the Lab Brick RF Switch steps through the pattern elements, waiting for the specified hold time at each step. Hold times are specified in milliseconds, with the minimum being 1 millisecond. Currently, a pattern can have at most four entries, so the maximum value for `num_entries` is 4. The array of switch selections, `sw_select`, has one element for each step in the pattern, and that element holds a switch number from 1 to 4. To start or stop a pattern use the `fnLSW_StartPattern` function.

`int fnLSW_StartPattern(DEVID deviceID, bool go)`  
Calling this function with `go` set to TRUE starts a switch pattern sequence at the beginning. To stop the pattern, call this function with `go` set to FALSE.

`int fnLSW_SetPatternType(DEVID deviceID, bool continuous )`

Calling this function with continuous set to TRUE before starting the pattern in order to have the pattern repeat. If continuous is set to FALSE the pattern will only run once when it is started.

```
int fnLSW_SetPatternEntry(DEVID deviceID, int sw_select, int holdtime, int index, bool last_entry)
```

This function can be used to set individual elements of the pattern. The argument `sw_select` is the switch setting, from 1 to 4. The argument `holdtime` is the length of time that the pattern will hold each switch setting, expressed as an integer number of milliseconds. The argument `index` is the zero based position in the pattern, ranging from 0 to 3. The `last_entry` argument should be set to TRUE only for the final element in the pattern. For example, the following set of calls define a pattern with three steps, where switch 1 is active for 1 second, switch 2 is active for .1 seconds, and switch 3 is active for 10 seconds on device 5:

```
result = fnLSW_SetPatternEntry(5, 1, 1000, 0, FALSE);
result = fnLSW_SetPatternEntry(5, 2, 100, 1, FALSE);
result = fnLSW_SetPatternEntry(5, 3, 10000, 2, TRUE);
```

```
int fnLSW_SetFastPulsedOutput(DEVID deviceID, float pulseontime, float pulseretime, bool on)
```

This function is the preferred way to control the internal pulse switching option. The `pulseontime` parameter is the length of the pulse on time (switch 1 active) 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.

```
int fnLSW_SetPulseOnTime(DEVID deviceID, float pulseontime)
```

This function is used to set the length of the RF pulse on time of the device's internal pulse switching. The `pulseontime` parameter is the length of the pulse on time (switch 1 active) in seconds, with a 100 nanosecond minimum. This function is not recommended for general use. Instead use the `fnLSW_SetFastPulsedOutput` function.

```
int fnLSW_SetPulseOffTime(DEVID deviceID, float pulseofftime)
```

This function is used to set the length of the RF pulse off time of the device's internal pulse switching. The `pulseofftime` parameter is the length of the pulse off time (switch 2 active) 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 `fnLSW_SetFastPulsedOutput` function.

```
int fnLSW_EnableInternalPulseMod(DEVID deviceID, bool on)
```

This function is used to turn on and off the internal pulse switching. If `on = TRUE` the switch will switch the RF output between switch 1 and switch 2 according to the values set for the pulse on time and pulse off time using either the

fnLSW\_SetFastPulsedOutput function or the functions to set pulse on and off time directly . To stop the internal pulse switching, set on = FALSE. Always disable internal pulse switching before setting the pulse on and off time using the fnLSW\_SetPulseOnTime and fnLSW\_SetPulseOffTime functions.

int fnLSW\_SaveSettings(DEVID deviceID)

The Lab Brick RF Switches 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.

### Functions – Reading parameters from the RF Switch

int fnLSW\_GetNumSwitches (DEVID deviceID)

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

int fnLSW\_GetActiveSwitch (DEVID deviceID)

This function returns the current switch connection of the selected device. This value may differ from the current switch setting when an external signal is used to control the switch, or when a switch pattern is running, or during pulse mode operation. Note that for rapidly changing switch connections due to an external signal, switch patterns or pulse mode operation the value returned by the GetActiveSwitch function may not be a useful indicator of the actual switch connection since the value returned represents the switch connection at the last status report which is asynchronous with respect to the call to the GetActiveSwitch function.

int fnLSW\_GetSwitchSetting (DEVID deviceID)

This function returns the current switch setting of the selected device. In normal operation, this value is the same as the active switch, except in the conditions described above.

int fnLSW\_GetUseExternalControl (DEVID deviceID)

This function returns a non-zero value if the Lab Brick RF Switch has been set to use an external signal to control the switches.

float fnLSW\_GetPulseOnTime(DEVID deviceID)

This function returns the pulse on time, which is the length of time that RF input is connected to output switch 1 when internal pulse modulation is operating, in seconds.

float fnLSW\_GetPulseOffTime(DEVID deviceID)

This function returns the pulse off time, which is the length of time that RF output is connected to switch 2 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 fnLSW\_GetPulseMode(DEVID deviceID)

This function returns an integer value which is 1 when the RF Switch's internal pulse modulation is active, or 0 when the internal pulse modulation is off.

int fnLSW\_GetHasFastPulseMode(DEVID deviceID)

This function is included for compatibility with software developed for other Lab Brick products. All Lab Brick RF Switches have fast pulse mode switching.

int fnLSW\_GetPatternLength (DEVID deviceID);

This function returns an integer value which is the number of elements in the switch pattern. Currently, the maximum pattern length is 4. A pattern length of 0 indicates that no pattern has been loaded into the Lab Brick RF Switch.

int fnLSW\_GetPatternType (DEVID deviceID);

This function returns the current pattern type. A value of 1 indicates that a single shot pattern was selected, and a value of 2 indicates that a repeating pattern was selected.

int fnLSW\_GetPatternEntrySwitch (DEVID deviceID, int index);

This function returns the switch setting for a particular element in the array of switch settings that define the switch pattern. The index ranges from 0 to 3. A value of zero indicates the end of the pattern, while values of 1 to 4 indicate the switch setting for that step in the pattern.

int fnLSW\_GetPatternEntryTime (DEVID deviceID);

This function returns the hold time for a particular element in the array of switch settings that define the switch pattern. The index ranges from 0 to 3. The integer value returned is the length of time, in 1 millisecond increments that the switch will remain at that step in the pattern