

Event Dispatcher Framework Based LED Controller with Push Button and Web Based Interface

Objective: The purpose of this mini project is to demonstrate how the Event Dispatcher Framework Library can be used to easily create software that can switch an LED on and off in response to input from a push button and through a web based interface (a webpage). The project will use the Raspberry Pi computer as the hardware platform. The Raspberry Pi computer will be wired directly to the LED that it is controlling as well as a push button.

Figure #1 below shows the complete LED control system including the web based interface running on a tablet on the left, the Raspberry Pi computer running the Apache server and control software on the right and the a breadboard with the LED and pushbutton circuit in the center.



figure #1

Theory: In this project a Raspberry Pi computer is used to control the on/off state of an LED. There are two pieces of software used to implement the functionality of the system, these are the Apache server and the control software.

The Apache Server

The role of the Apache web server software is to serve a web page to desktop computers and mobile devices that is used as a web interface to control the LED state across a network, see figure #2.

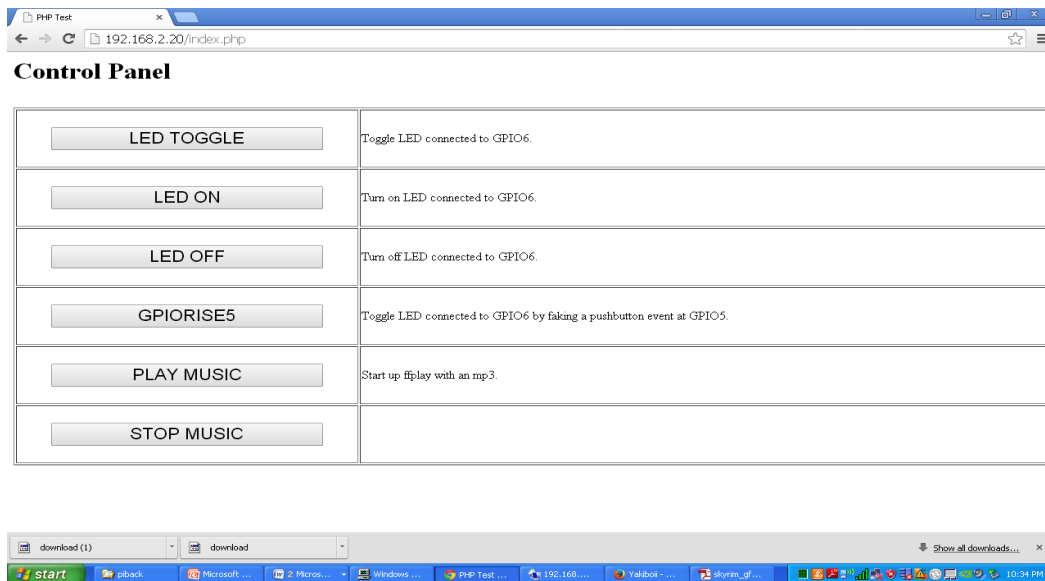


figure #2

Whenever a button on the web interface is pressed, a server side PHP script is executed on the Apache server that sends a command to the second piece of software (control software) using a UDP packet.

Below is a list of all possible event message strings that can be sent to the control software by the PHP script in response to button submit events that occur on the web page (web interface) in the browser.

- LED TOGGLE - UDP event string that causes light to toggle.
- LED ON - UDP event string that causes light to turn on.
- LED OFF - UDP event string that causes light to turn off.
- PLAY MUSIC - UDP event string that causes an mp3 file to play.
- STOP MUSIC - UDP event string that causes an mp3 file to stop (not implemented).

The Control Software

The control software receives UDP packets containing “event messages” from the Apache server. Event messages are short strings that tell the event dispatcher framework that events such as pressed button on web interface or physical button wired directly to the Raspberry Pi computer are pressed. The event

dispatcher framework responds to each event message by selecting the appropriate event handler (call the correct function) to actually switch the LED on or off.

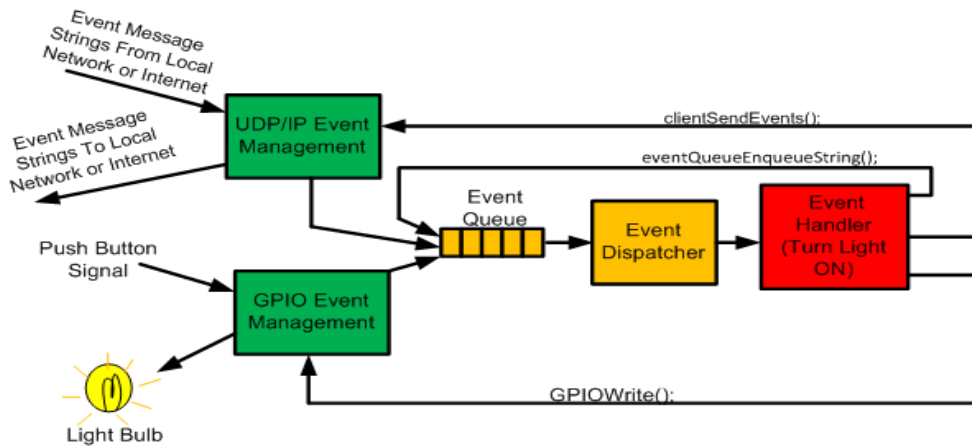


Figure #3

Figure #3 shows how event messages, the event dispatcher and the event handler (function) are related. Note that only the event handlers need to be created by the user, all other blocks are implemented in the Event Dispatcher Framework Library (see Event Dispatcher Framework Library manual).

System Overview:

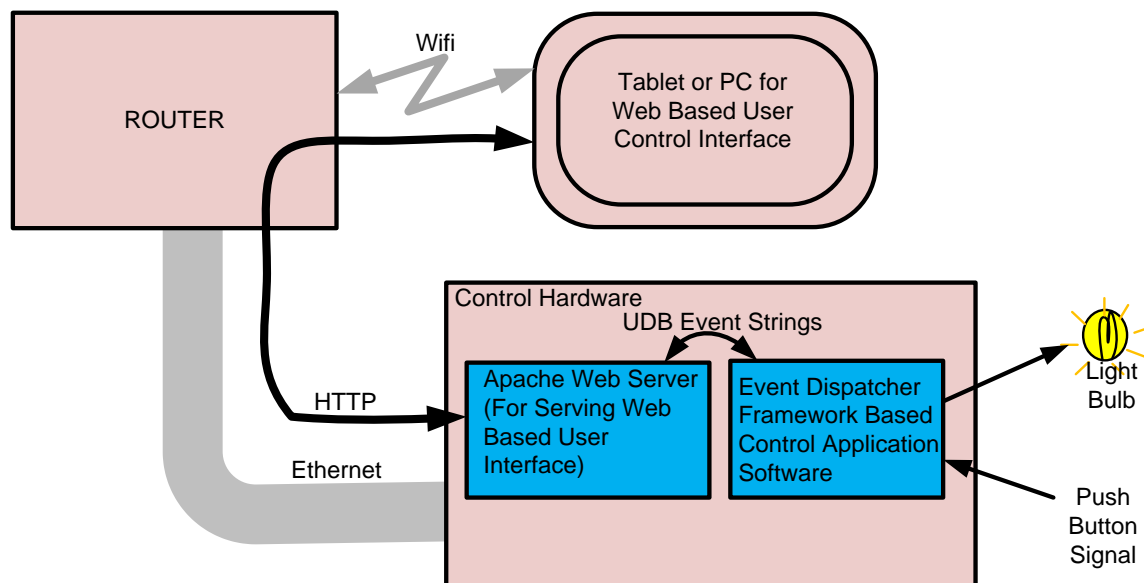


figure #4

The overall system is illustrated in figure #4. A tablet or other computer that is used as HMI (human machine interface, where the web interface is browsed/operated) communicates through a router to the Apache interface which in turns sets command to turn the LED on/off or start up some music

playing. The diagram also shows how the physical pushbutton and the LED are related to the event dispatcher framework based control software.

Schematic:

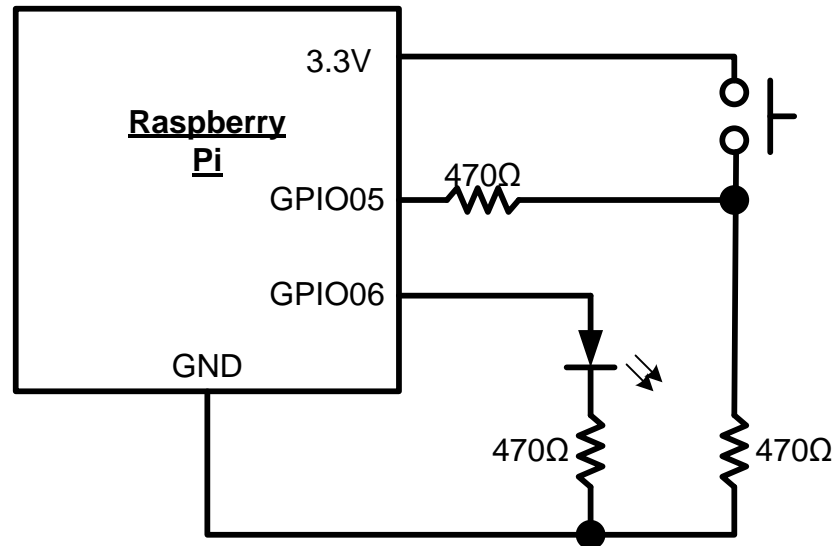


figure #5

The schematic in figure #5 shows how the LED and the pushbutton should be connected to Raspberry Pi computer. It does not show the USB power supply to the computer or the RJ45 network cable connection; these are implied (necessary, but goes without saying).

The Control Software Source Code:

The source code below is a short program that implements the control software, it is written in C language. Since all of the UDP event management, GPIO management and the event dispatcher are already implemented in the Event Dispatcher Framework Library, the user defined portion of the software code only needs to initialize the libraries and provide the user define event handlers.

```
#include <stdio.h>
#include <string.h>
#include "ef.h"
#include "UDPEvents.h"

void GPIO5RisingEventHandler(); /*Handler that toggles LED (or button signal rises) . */
void LEDOff(); /* Handler that turns LED off. */
void LEDOn(); /* Handler that turns LED on. */
void playMusic(); /* Handler that starts music. */

int main()
{
    char tempString[EVENTSIZEMAX];
```

```

char testString[64]
int i;

printf("Event queue init\n");
eventQueueInit(); /* Starts up event queue. */

printf("Event dispatcher init.\n");
eventHandlerFrameworkInit(); /* Starts up event dispatcher. */

printf("GPIO init\n");
GPIOInit(); /* Starts up GPIO management. */

printf("UDP Event init\n");
UDPEventsInit(8888); /*Starts up UDP event management. */

/* Sends test event packet to event logger, if available locally and on port 45000 */
clientSendEvents("127.0.0.1", 45000, "TESTEVENT");

GPIOOutputMode(6); /* Initializes GPIO06 as output (for LED) and sets to off state. */
GPIOWrite(6, 0);

GPIOInputMode(5); /* Initializes GPIO05 as digital input (from pushbutton). */

printf("Start event dispatcher test.\n");

/* Tell the event dispatcher to pair each type of event message string type with their */
/* respective event handler. Ex. An "LED ON" event causes LEDOn() to be called. */
addEventHandler("GPIO RISE5", GPIO5RisingEventHandler);
addEventHandler("LED TOGGLE", GPIO5RisingEventHandler);
addEventHandler("LED ON", LEDOn);
addEventHandler("LED OFF", LEDOff);
addEventHandler("PLAY MUSIC", playMusic);

/* Loop here forever to keep processing events. */
while(1)
{
    /* Check to see if an input or output pin has changed state. */
    GPIOEdgeEventDetect();

    /* Check to see if event messages were received at UDP socket. */
    serverRecieveEvents();

    /* Dispatch each event in the event queue until it is empty. */
    while(!eventQueueEmpty())
    {
        eventHandlerDispatcher(); /* Call event handler. */
        printf("Event log: %s\n", auxEventString); /* Print any event messages in the queue. */
        fflush(stdout);
    }
}

```

```

        }
        delay(10);    /* Let this program rest so that the CPU can do other things. */
    }

    printf("End event dispatcher test.\n");

    return(1);
}

/*Handler that toggles LED (or button signal rises).*/
void GPIO5RisingEventHandler()
{
    if(GPIORead(6))
    {
        GPIOWrite(6, 0);
        clientSendEvents("127.0.0.1", 45000, "LED OFF");
    }
    else
    {
        GPIOWrite(6, 1);
        clientSendEvents("127.0.0.1", 45000, "LED ON");
    }
    return;
}

/* Handler that turns LED on. */
void LEDOn()
{
    GPIOWrite(6, 1);
    return;
}

/* Handler that turns LED off. */
void LEDOff()
{
    GPIOWrite(6, 0);
    return;
}

/* Handler that starts music. */
void playMusic()
{
    system("ffplay Thirtyfive.mp3 &"); /* Start up music player in background. */
    return;
}

```

Compiling the Source:

Place the source code for the control program (maintest3.c) and the source from the library into a directory where you plan to compile the control software. Type the line below at the command prompt to compile the program in the source code directory. The result will be an executable program named test3:

```
gcc -o test3 ef.c gpio.c UDPEvent.c maintest3.c -lwiringPi
```

To startup the control software type the following at to command prompt:

```
./test3
```

The PHP Web Interface Script:

The PHP script (index.php) which is executed by the web server (Apache) serves the dual purpose of emitting (generating) the web interface (web page) seen in the browser and for producing and sending a UDP based packet to the control software whenever a button on the web interface ins pressed:

```
<html>
<head>
  <title>PHP Test</title>
</head>
<body>
<h1>
  <?php echo '<p>Control Panel</p>'; ?>
</h1>

<?php
  $sock = socket_create(AF_INET, SOCK_DGRAM, SOL_UDP);

  $msg = $_POST[submit];
  $len = strlen($msg);

  // Send event message to control software
  socket_sendto($sock, $msg, $len, 0, '127.0.0.1', 8888);
  socket_close($sock);
?>

<form action="index.php" method="post">
  <table border="1" width="100%">
    <tr>
      <td align="center" width="25%" height = 80 style="font-size: 25px;">
        <p><input type="submit" style="width:80%; height:100%;font-size:22px;"
name="submit" value="LED TOGGLE"></p>
      </td>
      <td width="50%"> Toggle LED connected to GPIO6.</td>
    </tr>
    <tr>
      <td align="center" width="25%" height = 80 style="font-size: 25px;">
```

```

                <p><input type="submit" style="width:80%; height:100%;font-size:22px;"
name="submit" value="LED ON"></p>
            </td>
            <td> Turn on LED connected to GPIO6. </td>
        </tr>
    <tr>
        <td align="center" width="25%" height = 80 style="font-size: 25px;">
            <p><input type="submit" style="width:80%; height:100%;font-size:22px;"
name="submit" value="LED OFF"></p>
        </td>
        <td> Turn off LED connected to GPIO6.</td>
    </tr>
    <tr>
        <td align="center" width="25%" height = 80 style="font-size: 25px;">
            <p><input type="submit" style="width:80%; height:100%;font-size:22px;"
name="submit" value="GPIORISE5"></p>
        </td>
        <td> Toggle LED connected to GPIO6 by faking a pushbutton event at GPIO5.</td>
    </tr>

    <tr>
        <td align="center" width="25%" height = 80 style="font-size: 25px;">
            <p><input type="submit" style="width:80%; height:100%;font-size:22px;"
name="submit" value="PLAY MUSIC"></p>
        </td>
        <td> Start up ffmpeg with an mp3.</td>
    </tr>

    <tr>
        <td align="center" width="25%" height = 80 style="font-size: 25px;">
            <p><input type="submit" style="width:80%; height:100%;font-size:22px;"
name="submit" value="STOP MUSIC"></p>
        </td>
        <td> </td>
    </tr>
</table>
</form>
</body>
</html>

```

This file (index.php) should be copied into the following directory:

/var/www

To View and operate the PHP based web interface, type in the IP address and the name of the script in the URL box of a browser running on a computer or tablet on the same network as the R Pi Computer:

Example URL: **192.168.2.20/index.php**

The web interface that appears in the browser is should appear the same as the one shown in figure #2.

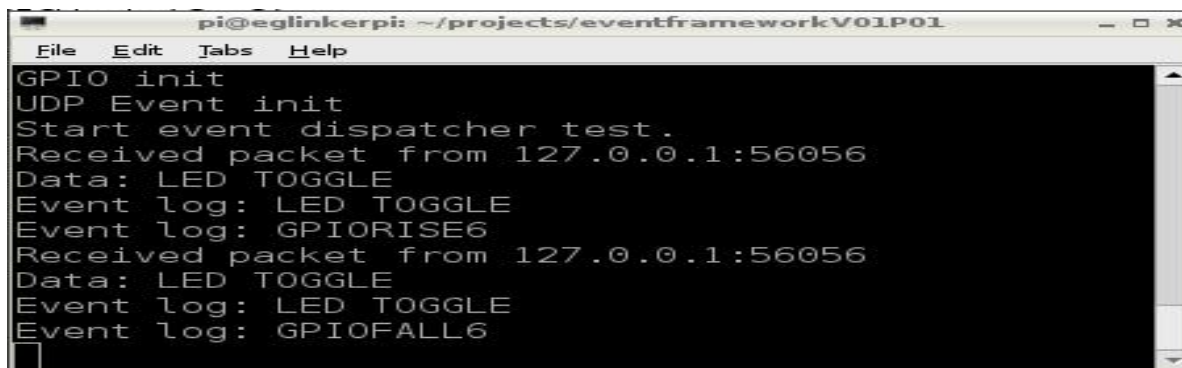
Testing:

From the web interface press the following sequence of buttons on the browser window:

LED TOGGLE

LED TOGGLE

The following should be observed in the controller software window running on the Raspberry Pi:



```
pi@eglinkerpi: ~/projects/eventframeworkV01P01
File Edit Tabs Help
GPIO init
UDP Event init
Start event dispatcher test.
Received packet from 127.0.0.1:56056
Data: LED TOGGLE
Event log: LED TOGGLE
Event log: GPIORISE6
Received packet from 127.0.0.1:56056
Data: LED TOGGLE
Event log: LED TOGGLE
Event log: GPIOFALL6
```

Figure #6

As is seen in figure #6 and figure #7, the two LED TOGGLE events cause GPIO6 to rise and then fall, accompanied by the corresponding observation that LED wired to the Raspberry Pi computer is turned on, then off:

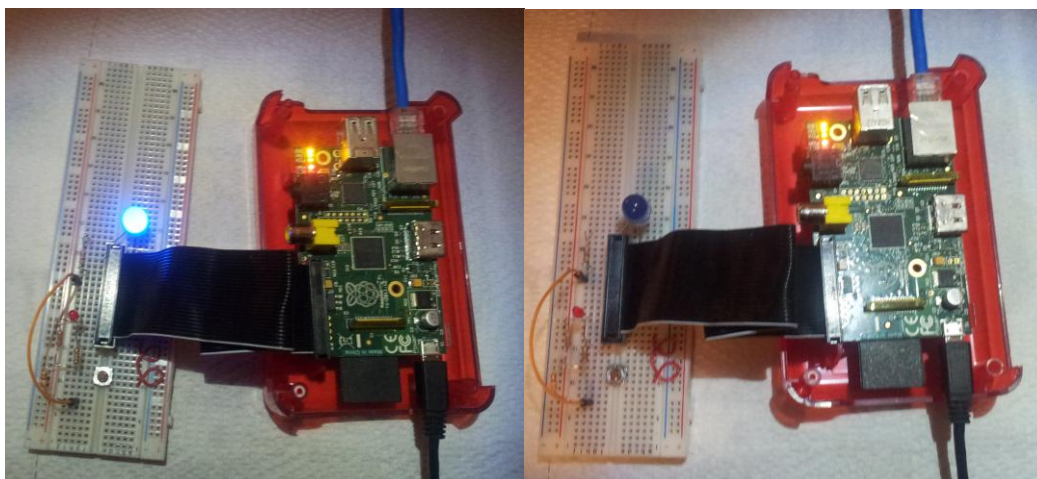
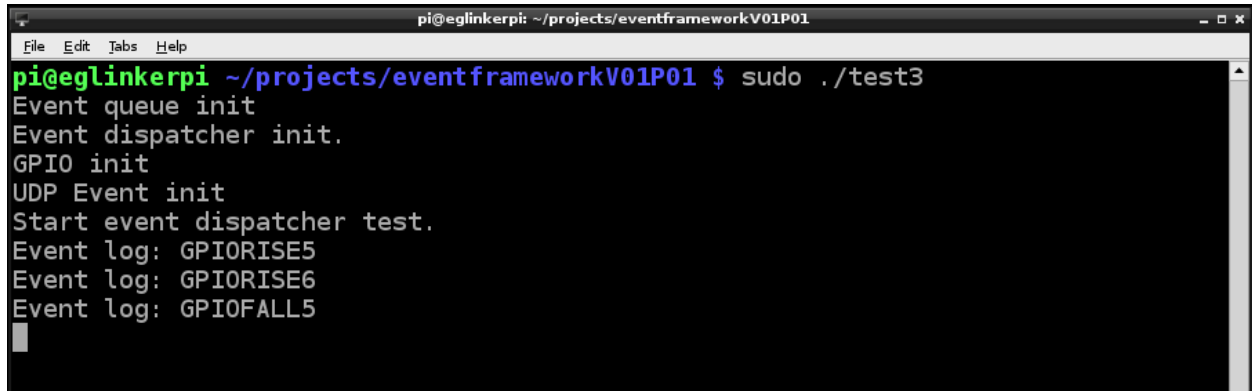


Figure #7

When the physical pushbutton that is wired to the Raspberry Pi computer is pressed twice it should also be observed that the LED will toggle on, then off (see figure #8). A physical push button event (pushing the button electrically connected to GPIO5) should also appear in the control software window of the R Pi computer as a GPIORISE5 event followed by a GPIPFALL5 event:

A terminal window titled 'pi@eglinkerpi: ~/projects/eventframeworkV01P01' with a menu bar (File, Edit, Tabs, Help). The terminal output shows the execution of 'sudo ./test3' and subsequent initialization steps: 'Event queue init', 'Event dispatcher init.', 'GPIO init', 'UDP Event init', and 'Start event dispatcher test.'. It then logs three events: 'Event log: GPIORISE5', 'Event log: GPIORISE6', and 'Event log: GPIOFALL5'.

```
pi@eglinkerpi ~/projects/eventframeworkV01P01 $ sudo ./test3
Event queue init
Event dispatcher init.
GPIO init
UDP Event init
Start event dispatcher test.
Event log: GPIORISE5
Event log: GPIORISE6
Event log: GPIOFALL5
```

figure #8

Conclusions:

This mini project demonstrates how the Event Dispatcher Framework Library can be use to create a simple control application where an LED can be controlled using a web interface or through a physical button without having to go in-depth into the intern working of the TCP/IP/UDP and GPIO programming.