

## How to use an Xbox Guitar Hero Controller with Linux

### Setup:

In order to use the Xbox Guitar Hero Controller (GHC), the Xbox Gamepad Driver (XbGD) is required to get the data from the GHC. While the XbGD runs, it will continuously output data from the GHC that can then be used by an application.

Installing the Xbox Gamepad Driver for Linux:

If on Ubuntu:

```
sudo add-apt-repository ppa:grumbel/ppa  
sudo apt-get update  
sudo apt-get install xboxdrv
```

If on linux:

Download the latest version of the Xbox Gamepad Driver for Linux:

<http://pingus.seul.org/~grumbel/xboxdrv/>

Extract and install

### Running the Driver:

```
xboxdrv  
xboxdrv --quiet
```

The quiet option disables startup information from being displayed. If this option is not used, there will be a 3 second delay before the driver starts outputting data while it displays the startup information.

## Parsing the Data:

While the XbGD runs, it will continuously read data from the Guitar Hero Controller (GHC).

Example output:

```
X1: 0 Y1: 0 X2: -5796 Y2: 32767 du:0 dd:0 dl:0 dr:0 back:0 guide:0 start:0 TL:0 TR:0 A:0 B:0 X:0 Y:0 LB:0 RB:0 LT:118 RT: 61
```

This data is output as a single string, and needs to be parsed before it can be used by another program.

To parse the data, the XbGD must be opened as a file so it's output can be captured.

```
static const int DRIVER_OUTPUT_LENGTH = 145;
char driverOutput[DRIVER_OUTPUT_LENGTH];
FILE *file = fopen("xboxdrv --quiet", "r");
```

This allows each instance of data output by the XbGD to be converted to a single string that can be parsed:

```
while (fgets(driverOutput, DRIVER_OUTPUT_LENGTH, file) != NULL) {
    parseDriverOutput(driverOutput);
}
```

Each of the data values output by the driver XbGD are separated by a single colon ":". This allows the output data to be parsed by checking the output string for a colon, and reading the data that follows the colon.

```
for(int i = 0; len > i; ++i) {
    if (driverOutput[i] == ':') {
        //process data
    }
}
```

Once the raw data has been pared by colon, blank space before and after the data values must be filtered out before the raw data can be converted.

```
//removes leading white space
while(driverOutput[++i] == ' ') {}

//copies the data value from the driver output
value[0] = driverOutput[i];
valueIndex = 1;
while(driverOutput[++i] != ' ' && len - 1 > i) {
    value[valueIndex] = driverOutput[i];
    ++valueIndex;
}
```

The raw data can now be converted from a string to an integer.

```
data[dataCount] = atoi(value);
++dataCount;
```

### Example Parsing Code:

```
static const int DATA_VALUES_COUNT = 21;
static const int OUTPUT_BUFFER_LENGTH = 6;

static void parseDriverOutput(char *driverOutput) {
    int len = strlen(driverOutput);

    int valueIndex;
    int dataCount = 0;

    char value[OUTPUT_BUFFER_LENGTH];
    bool data[DATA_VALUES_COUNT];

    for(int i = 0; len > i; ++i) {
        if (driverOutput[i] == ':') {

            //removes leading white space
            while(driverOutput[++i] == ' ') {}

            //copies the data value from the driver output
            value[0] = driverOutput[i];
            valueIndex = 1;
            while(driverOutput[++i] != ' ' && len - 1 > i) {
                value[valueIndex] = driverOutput[i];
                ++valueIndex;
            }

            //adds a null character to the end
            value[valueIndex] = 0;

            //converts the output to an integer
            data[dataCount] = atoi(value);
            ++dataCount;
        }
    }

    //parse the converted data
    parseData(data);
}
```

## Using the Data:

Once the output data has been converted from it's raw string form to an integer value, it needs to be further parsed to determine what part of the GHC that data corresponds to.

This can be done by storing all the data in a collection object (array, list, vector, etc.), and then parsing the data in the collection by index. Since the order of the data is always the same, reading from a specific index will always correspond to the same part of the GHC.

```
static void parseData(bool *driverData) {  
    struct GuitarData data;  
  
    data.buttonGreen      = driverData[INDEX_GREEN];  
    data.buttonRed        = driverData[INDEX_RED];  
    data.buttonYellow     = driverData[INDEX_YELLOW];  
    data.buttonBlue       = driverData[INDEX_BLUE];  
    data.buttonOrange     = driverData[INDEX_ORANGE];  
  
    data.buttonStrumUp    = driverData[INDEX_STRUM_UP];  
    data.buttonStrumDown  = driverData[INDEX_STRUM_DOWN];  
  
    data.buttonKeypadLeft = driverData[INDEX_KEYPAD_LEFT];  
    data.buttonKeypadRight = driverData[INDEX_KEYPAD_RIGHT];  
  
    data.selectKey        = driverData[INDEX_KEY_SELECT];  
    data.startKey         = driverData[INDEX_KEY_START];  
}
```

## Reading Button Data:

Because the XbGD outputs data at an extremely high rate (approximately 100 times per second), a user pressing down on a button on the GHC, will cause multiple reads of the same input.

To prevent this from happening, each button input (input that only has 2 values) should have a boolean value associated with it. This boolean should initially be set to false, and set to true the first time the corresponding button is pressed. The boolean should remain true until the button is no longer pressed, in which case the boolean should be set to false.

By ignoring any inputs when the boolean is set to true, multiple reads of the same button input are prevented.

```
static bool *isButtonPressed;
static void buttonPressed(bool buttonValue, int index) {
    if (buttonValue) {
        if (isButtonPressed[index] == 0) {
            //first time read of that button being pressed
            //do stuff...
        }
        isButtonPressed[index] = 1;
    }
    isButtonPressed[index] = 0;
}
```

### Troubleshooting:

Depending on the run environment, the XbGD might have trouble starting. If that is the case try the option:

`--detach-kernel-mode`

When connecting the GHC, make sure to check that it has become activated. This can be done by checking the xbox status icon on the GHC.

Inactive GHC:



Active GHC:



When connecting to the BeagleBone Black through the ZenCape, the GHC can remain inactive. If this happens, connect the GHC to another device. If the GHC becomes activated by the second device, disconnect it, and reconnect it to the BeagleBone Black. This may require several attempts until the GHC becomes active while connected to the BeagleBone Black.