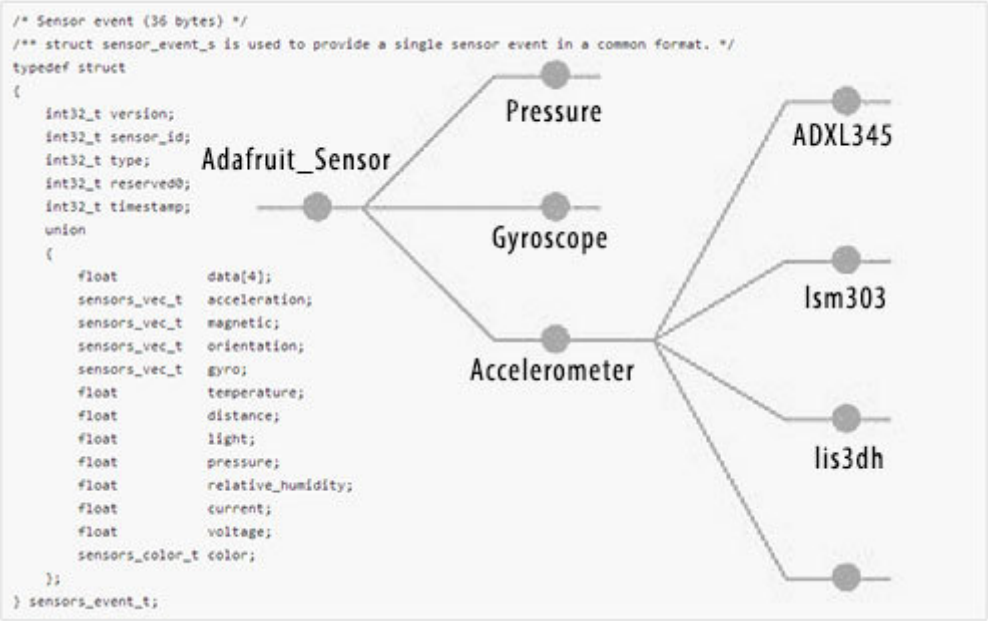
Using the Adafruit Unified Sensor Driver

Created by Kevin Townsend

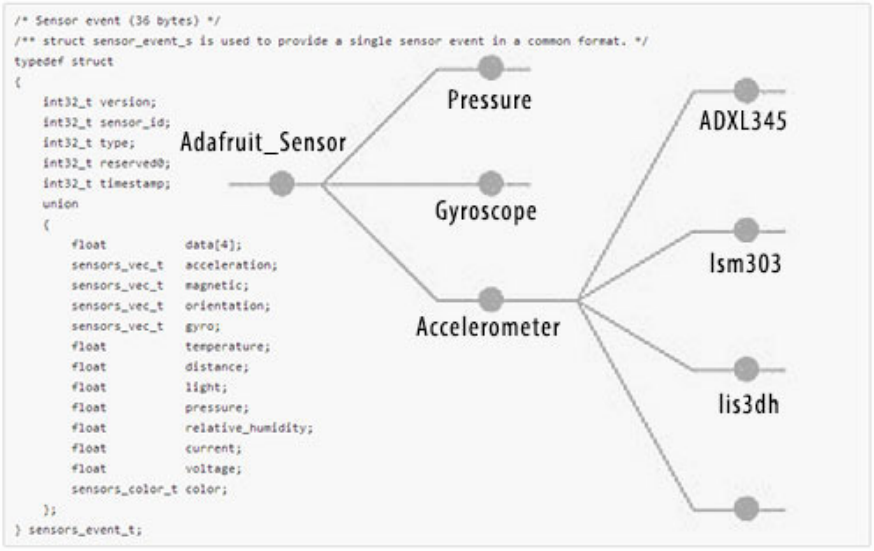


https://learn.adafruit.com/using-the-adafruit-unified-sensor-driver

Last updated on 2024-06-03 01:15:30 PM EDT

Table of Contents

Introduction



One Type to Rule Them All

Adafruit has always prided itself on going the extra mile and providing basic drivers for all of our sensors, but as the number of sensor breakouts we have has steadily grown over the years, it seemed worthwhile to take a step back and ask if we can do things a bit better.

While digging around in the Android source code (https://adafru.it/aZk) for something entirely unrelated, I noticed the intelligent way Android abstracts away all sensor data into a single 'type' (https://adafru.it/aZl) and uses standard, pre-defined SI units of measure.

This makes sense since HW manufacturers are free to put whatever sensors they want in their phone, but Android needs a way to understand sensor data from any source and translate that into something it knows what to do with (i.e., number that make sense).

This is also (conveniently) that exact same problem many people playing with embedded systems have! They may need to swap out one accelerometer for another depending on price, availability, resolution, or any number of factors. The traditional approach of one-off drivers doesn't work well here, though, since every driver probably reports separate units and scales of measure, in different types and sizes, etc., and it can be painful to switch from one sensor to another, even if they offer similar functionality.

The Adafruit Unified Sensor driver follows Android's lead here and tries to harmonize

sensor data into a single (lighter weight than the heavier Android model) 'type', and also abstracts away details about the sensor into another common 'type' that lets us describe sensors in a well-defined manner.

Why Is This a Good Thing?

The biggest advantage is that using drivers based on the Adafruit Unified Sensor Driver (https://adafru.it/aZm) allows you to take any supported sensor, drop it into your system, and it should just work out of the box with the rest of your code.

Any related driver will report the same SI units, using the same scale, and you will call exactly the same functions to read your data.

The ADXL345 (https://adafru.it/aZn) accelerometer isn't available or you need something cheaper? Just grab any other accelerometer (the LSM303DLHC (https://adafru.it/aYU) for example), change the constructor and off you go!

Adafruit_Sensor in Detail

The unified sensor drivers system is based around a single base class named **Adafruit_Sensor** (source code (https://adafru.it/aZo)). This base class defines two functions and and a few shared 'types', which are described below.

Essentially, 'Adafruit_Sensor' is a contract that every driver signs to make sure it plays well with other unified sensor drivers and programs that make use of the system. Anything the inherits from Adafruit_Sensor can obscure away almost all of it's technical particularities, and instead present this common facade.

When a new fancy sensor comes out at half the price of your current one, you don't have to worry about recreating the wheel since the new driver snaps nicely into place because it was created with the same common mold in mind! All you need to care about is that it follows the contract set down by the base class, and when you call those pre-defined functions, you get back the expected results in the expected units and scale.

Standardised SI Units for Sensor Data

A key part of the unified sensor driver system layer is the standardisation of values on SI units of a particular scale. This following SI units and scales are used for the appropriate sensor type:

- **acceleration**: values are in **meter per second per second** (m/s^2)
- **magnetic**: values are in **micro-Tesla** (uT)

- **orientation**: values are in **degrees**
- **gyro**: values are in **rad/s**
- **temperature**: values are in **degrees centigrade** (Celsius)
- **distance**: values are in **centimeters**
- **light**: values are in **SI lux** units
- **pressure**: values are in **hectopascal** (hPa)
- **relative_humidity**: values are in **percent**
- **current**: values are in **milliamps** (mA)
- **voltage**: values are in **volts** (V)
- **color**: values are in 0..1.0 **RGB channel luminosity** and 32-bit RGBA format

This is one of the key benefits of the unified sensor driver system. By using standard SI units in drivers, any accelerometer can be switched out with any other accelerometer with minimal impact on the rest of the system because they all produce exactly the same units and scale (m/s^2)!

No more wondering what 0 or 1023 means, since the necessary conversions and calculations are already done for you behind the scene!

Supported Sensor Types

While this list may expand in the future, the following sensor types are currently supported in the Adafruit Unified Sensor System (based on 'sensors_type_t' in Adafruit_Sensor.h):

```
/** Sensor types */
typedef enum
{
  SENSOR_TYPE_ACCELEROMETER         = (1),   /**&lt; Gravity + linear acceleration
*/
  SENSOR_TYPE_MAGNETIC_FIELD        = (2),
  SENSOR_TYPE_ORIENTATION           = (3),
  SENSOR_TYPE_GYROSCOPE             = (4),
  SENSOR_TYPE_LIGHT                 = (5),
  SENSOR_TYPE_PRESSURE              = (6),
  SENSOR_TYPE_PROXIMITY             = (8),
  SENSOR_TYPE_GRAVITY               = (9),
  SENSOR_TYPE_LINEAR_ACCELERATION   = (10),  /**&lt; Acceleration not including
gravity */
  SENSOR_TYPE_ROTATION_VECTOR       = (11),
  SENSOR_TYPE_RELATIVE_HUMIDITY     = (12),
  SENSOR_TYPE_AMBIENT_TEMPERATURE   = (13),
  SENSOR_TYPE_VOLTAGE               = (15),
  SENSOR_TYPE_CURRENT               = (16),
  SENSOR_TYPE_COLOR                 = (17)
} sensors_type_t;
```

Key Functions in Adafruit_Sensor

Anything that works with the unified sensor driver system uses two basic 'types', with each type having an associated read function for it. These data types are describe in a lot more detail below, but they're summarised below for convenience sake:

- **sensors_event_t** - This type is used to encapsulate a specific sensor reading, called an 'event', and contains a data from the sensor from a specific moment in time.
- **sensor_t** - This type is used to describe some basic details about this specific sensor, including the sensor name, the resolution and range of the sensor (in the SI units described above), as well as a unique ID that you can assign to this sensor so that you can identify dozens of identical sensors if you are logging your sensor data in a central database.

To fill an object with the right data, you simply need to call one of the two mandatory functions in the unified driver system:

void getEvent(sensors_event_t*)

This function will read a new set of values from you sensor (a sensor 'event'), convert them to the appropriate SI units and scale, and then assign the results to a specific sensors_event_t object.

This is the function you call to 'read' your sensor!

A pointer to sensors_event_t object is used here to save memory, so that you only ever need one sensor_event_t object since they use 36 bytes of memory each.

After creating an instance of your driver ('tsl' below), you would call this function as follows:

```
/* Create an instance of our sensor (only do this once!) */
Adafruit_TSL2561 tsl = Adafruit_TSL2561(TSL2561_ADDR_FLOAT, 12345);

...

/* Create a sensors_event_t object in memory to hold our results */
sensors_event_t event;

/* Get a new sensor event, passing in our 'event' placeholder */
tsl.getEvent(&amp;event);

/* Now do something with the sensor event data ... */

/* Display the results (light is measured in lux) */
if (event.light)
{
```

```
  Serial.print(event.light); Serial.println(" lux");
}
else
{
  /* If event.light = 0 lux the sensor is probably saturated
     and no reliable data could be generated! */
  Serial.println("Sensor overload");
}
```

void getSensor(sensor_t*)

This function returns information about the sensor itself, such as the sensor name, the min and max values that the sensor can return, and a unique ID for this specific sensor that can help you distinguish different copies of the exact same sensor in your sensor network or project. If you have 50 light sensors on 50 tomato plants, it's useful to know which data is from which plant, and the unique ID is a great way to solve this problem in your data logs.

You can call this function with the following code (using the TSL2561 from above as an example):

```
/* Create a sensor_t object in memory to filled below */
sensor_t sensor;

/* Get the sensor details and place them in 'sensor' */
tsl.getSensor(&amp;sensor);

/* Display the sensor details */
Serial.println("------------------------------------");
Serial.print  ("Sensor:       "); Serial.println(sensor.name);
Serial.print  ("Driver Ver:   "); Serial.println(sensor.version);
Serial.print  ("Unique ID:    "); Serial.println(sensor.sensor_id);
Serial.print  ("Max Value:    "); Serial.print(sensor.max_value); Serial.println("
lux");
Serial.print  ("Min Value:    "); Serial.print(sensor.min_value); Serial.println("
lux");
Serial.print  ("Resolution:   "); Serial.print(sensor.resolution);
Serial.println(" lux");
Serial.println("------------------------------------");
Serial.println("");
```

Data Type Implementation Details

The two functions defined above manipulate specific data 'types'. These types (**sensor s_event_t** and **sensor_t**) allow us to abstract any type of sensor data into a single object that we can easily log, transmit, manipulate, etc., without requiring any knowledge of the type of sensor being used:

- **sensors_event_t** - This 'type' contains sensor data from a single read at a specific moment in time. The same event type is used for any sensor, with details on how this is made possible below. Sensor data is generally defined as

32-bit floating point values, but two exceptions are the data types defined below:

  - **sensors_vec_t** - This is a type used by sensors_event_t to encapsulate X/Y/Z or similar data in a generic way.
  - **sensors_color_t** - This is another type used by sensors_event_t to encapsulate basic color data.

- **sensor_t** - Data about the sensor itself.

sensors_event_t (sensor data)

'sensors_event_t' (defined in Adafruit_Sensor.h) encapsulates sensor data from any type of sensor.

The way this works is that a fancy feature of C called a 'union' is employed to pack a variety of fields into a single struct, and you can simply use the field that is appropriate for your specific requirements and ignore the rest.

The raw source code for the typedef is below:

```
/* Sensor event (36 bytes) */
/** struct sensor_event_s is used to provide a single sensor event in a common
format. */
typedef struct
{
    int32_t version;                        /**&lt; must be sizeof(struct
sensors_event_t) */
    int32_t sensor_id;                      /**&lt; unique sensor identifier */
    int32_t type;                           /**&lt; sensor type */
    int32_t reserved0;                      /**&lt; reserved */
    int32_t timestamp;                      /**&lt; time is in milliseconds */
    union
    {
        float           data[4];
        sensors_vec_t   acceleration;       /**&lt; acceleration values are in
meter per second per second (m/s^2) */
        sensors_vec_t   magnetic;           /**&lt; magnetic vector values are in
micro-Tesla (uT) */
        sensors_vec_t   orientation;        /**&lt; orientation values are in
degrees */
        sensors_vec_t   gyro;               /**&lt; gyroscope values are in rad/s
*/
        float           temperature;        /**&lt; temperature is in degrees
centigrade (Celsius) */
        float           distance;           /**&lt; distance in centimeters */
        float           light;              /**&lt; light in SI lux units */
        float           pressure;           /**&lt; pressure in hectopascal (hPa)
*/
        float           relative_humidity;  /**&lt; relative humidity in percent
*/
        float           current;            /**&lt; current in milliamps (mA) */
        float           voltage;            /**&lt; voltage in volts (V) */
        sensors_color_t color;              /**&lt; color in RGB component values
*/
```

```
    };
} sensors_event_t;
```

The advantage of handling things this way is that you can completely abstract away the different sensor types. If you are using a temperature sensor, you simply read the 'event.temperature' field, for a light sensor you read the 'event.light' field, etc.

Certain sensor types also use the **sensors_vec_t** or **sensors_color_t** data types. These are used for sensors that return more than one numeric value, such as X/Y/Z for an accelerometer or gyroscope, or r/g/b color data for a color sensor.

sensors_vec_t

```
/** struct sensors_vec_s is used to return a vector in a common format. */
typedef struct {
    union {
        float v[3];
        struct {
            float x;
            float y;
            float z;
        };
        /* Orientation sensors */
        struct {
            float azimuth;
            float pitch;
            float roll;
        };
    };
    int8_t status;
    uint8_t reserved[3];
} sensors_vec_t;
```

sensors_color_t

```
/** struct sensors_color_s is used to return color data in a common format. */
typedef struct {
    union {
        float c[3];
        /* RGB color space */
        struct {
            float r;
            float g;
            float b;
        };
    };
    uint32_t rgba;          /* 24-bit RGB value */
} sensors_color_t;
```

You can access these extended types in the same manner ... for acceleration data, for example, we could read the data with: 'event.acceleration.x', 'event.acceleration.y', etc.

sensor_t (Sensor details)

The other important type is sensor_t, which encapsulates all of the key information about this sensor, such as the sensor type, the name of this particular sensor, the driver version, the unique ID for this specific sensor instance (in case multiple sensors of the same type are used), etc.

The raw typedef for sensor_t is defined in Adafruit_Sensor.h, but can be seen below:

```
/* Sensor details (40 bytes) */
/** struct sensor_s is used to describe basic information about a specific sensor.
*/
typedef struct
{
    char     name[12];                       /**&lt; sensor name */
    int32_t  version;                        /**&lt; version of the hardware +
driver */
    int32_t  sensor_id;                      /**&lt; unique sensor identifier */
    int32_t  type;                           /**&lt; this sensor's type (ex.
SENSOR_TYPE_LIGHT) */
    float    max_value;                      /**&lt; maximum value of this
sensor's value in SI units */
    float    min_value;                      /**&lt; minimum value of this
sensor's value in SI units */
    float    resolution;                     /**&lt; smallest difference between
two values reported by this sensor */
    int32_t  min_delay;                      /**&lt; min delay in microseconds
between events. zero = not a constant rate */
} sensor_t;
```

The most important field here is **type**, which identifies the type of sensor and allows us to know what type of data to read from the sensors_event_t typedef.

Creating a Unified Sensor Driver

How do you create a new driver based around the unified sensor driver system, or port an existing driver over?

Thankfully it's not that complicated. You still have to create a basic driver the same way ... mapping out the correct registers, setting up some plumbing to send and receive the commands, etc. The only really different with a 'Unified Sensor Driver' is that there is an additional layer that sits on top of that underlying code that converts the sensor's native units into what Adafruit_Sensor is expecting.

Unified Driver Template

Since a simple code example usually makes the best documentation if you just want to get started quickly, we've put together a very simple template project that shows the minimum functionality that needs to be added to your driver to make it compliant with the unified sensor driver system.

You can get browse the code online or download it on github at https://github.com/adafruit/Adafruit_SensorTester (https://adafru.it/aZp).

They key elements are that your driver must inherit from the Adafruit_Sensor base class, which is handled in the class declaration in your header file:

Driver Header

```
#ifndef _ADAFRUIT_SENSOR_TESTOR_
#define _ADAFRUIT_SENSOR_TESTOR_

#if ARDUINO &gt;= 100
 #include "Arduino.h"
 #include "Print.h"
#else
 #include "WProgram.h"
#endif

#include &lt;Adafruit_Sensor.h&gt;
#include &lt;Adafruit_SensorTester.h&gt;

class Adafruit_SensorTester : public Adafruit_Sensor {

 public:
  /* Constructor */
  Adafruit_SensorTester(int32_t);

  void getEvent(sensors_event_t*);
  void getSensor(sensor_t*);

 private:
   int32_t _sensorID;
};

#endif
```

The other two main requirements here:

- You need to implement **getEvent** and **getSensor**, as defined in the base class, so the function prototypes are declared here in the header
- Every instance of a sensor has a unique ID that's passed in to the constructor. Notice the int32_t in Adafruit_SensorTester(int32_t)? This is what we'll use to store this ID and it needs to have some sort of local placeholder so we also add a private '_sensorID' variables.

Driver Class Implementation

Once you've updated the header, you need to add a couple things to your actual driver to implement the required functions and support the constructor that accepts a sensor ID.

You can see some sample code for this below (or download it directly from github if you want):

```
#include "Adafruit_SensorTester.h"

#include &lt;avr/pgmspace.h&gt;
#include &lt;limits.h&gt;
#include "pins_arduino.h"
#include "wiring_private.h"

Adafruit_SensorTester::Adafruit_SensorTester(int32_t sensorID) {
  _sensorID = sensorID;
}

void Adafruit_SensorTester::getEvent(sensors_event_t *event) {
  /* Clear the event */
  memset(event, 0, sizeof(sensors_event_t));

  event-&gt;version   = sizeof(sensors_event_t);
  event-&gt;sensor_id = _sensorID;
  event-&gt;type      = SENSOR_TYPE_PRESSURE;
  event-&gt;timestamp = 0;
  event-&gt;pressure = 123.0F;
}

void Adafruit_SensorTester::getSensor(sensor_t *sensor) {
  /* Clear the sensor_t object */
  memset(sensor, 0, sizeof(sensor_t));

  /* Insert the sensor name in the fixed length char array */
  strncpy (sensor-&gt;name, "TESTER", sizeof(sensor-&gt;name) - 1);
  sensor-&gt;name[sizeof(sensor-&gt;name)- 1] = 0;
  sensor-&gt;version     = 1;
  sensor-&gt;sensor_id   = _sensorID;
  sensor-&gt;type        = SENSOR_TYPE_PRESSURE;
  sensor-&gt;min_delay   = 0;
  sensor-&gt;max_value   = 300.0F;               // 300..1100 hPa
  sensor-&gt;min_value   = 1100.0F;
  sensor-&gt;resolution  = 0.01F;                // 0.01 hPa resolution

  /* Clear the reserved section */
  memset(sensor-&gt;reserved, 0, sizeof(sensor-&gt;reserved));
}
```

You can see here that we accept the sensor ID in the constructor and assign it to **_sendorID**. This value is then used in both **getEvent** and **getSensor** so that we always know which sensor data or sensor details are coming from (important if you're logging 30 different TSL2561 (https://adafru.it/aZ9) sensors!).

The key lines you'll need to changed are things like **event->type** and **sensor->type** to point to the correct sensor type (SENSOR_TYPE_PRESSURE is used here as an example).

You can also assign a small name to this sensor type ("BMP085", etc.).

Values assigned to sensor->name must be 12 characters or less. Anything longer than this will be truncated!

The third important point to noticed is '**event->pressure**'. This is where you need to assign the correct value for the latest sensor reading, but it first needs to be converted to the expected SI units (hPa in the case of pressure sensors). The exact field that you assign your value to will depend on the sensor->type.

- Acceleration sensors use: event->acceleration.x, event->acceleration.y, event->acceleration.z
- magnetic sensors use: event->magnetic.x, event->magnetic.y, event->magnetic.z
- Pressure sensors use: event->pressure
- etc.

You can see the supported sensors by looking at the **sensors_event_t** typedef in Adafruit_Sensor.h (https://adafru.it/aZo).

Existing Drivers

The easiest way to learn how to use the system, though, is to look at some complete examples. We've linked to some existing drivers on the next page of this tutorial ...

Where To From Here?

Where is Adafruit trying to go with this approach, you ask? The real goal is just to make things easier for customers, but also easier for use to support. It takes a lot of time and energy to constantly create example code, drivers, etc., and something like the Adafruit unified sensor driver seems like a win-win solution for everyone.

- Customers get much better data since they're working with pre-defined, standard SI units of measure and have a good idea of how get data out of a sensor, as well as a backup plan if one sensor type isn't avaialable.
- Adafruit benefits since it's so easy to reuse sample code from one accelerometer once you've created the first driver for the sensor type.

And hey ... it's just a nice, elegant solution to a complex problem in my opinion!

The current unified sensor system is far from perfect -- I'm not a big fan of floats, and is bigger than a barebones fixed-point sensor driver -- but I really think this is a huge step forward, and the right approach in the long term. I wish I had done this years ago, to be honest, and hope other people find it useful as well!

In any case ... if you have any thoughts or suggestions, either post them up in the forums, or suggest some concrete code changes on github in the

Adafruit_Sensor (https://adafru.it/aZm) repository!

Related Links

- Adafruit_Sensor (https://adafru.it/aZm) - The base class and core typedefs for the entire system
- Adafruit_SensorTester (https://adafru.it/aZp) - A simple template project to create a unified sensor driver

Existing Drivers

The easiest way to see how this all works, though, is to simply look at some of the existing drivers that support the unified sensor sytem. Some good examples can be found below:

Accelerometers

- Adafruit_ADXL345 (https://adafru.it/aZn)
- Adafruit_LSM303DLHC (https://adafru.it/aYU)

Light

- Adafruit_TSL2561 (https://adafru.it/aZ9)

Magnetometers

- Adafruit_LSM303DLHC (https://adafru.it/aYU)

Barometric Pressure

- Adafruit_BMP085_Unified (https://adafru.it/aZq)