HID Sensors Framework

HID sensor framework provides necessary interfaces to implement sensor drivers, which are connected to a sensor hub. The sensor hub is a HID device and it provides a report descriptor conforming to HID 1.12 sensor usage tables.

Description from the HID 1.12 "HID Sensor Usages" specification: "Standardization of HID usages for sensors would allow (but not require) sensor hardware vendors to provide a consistent Plug And Play interface at the USB boundary, thereby enabling some operating systems to incorporate common device drivers that could be reused between vendors, alleviating any need for the vendors to provide the drivers themselves."

This specification describes many usage IDs, which describe the type of sensor and also the individual data fields. Each sensor can have variable number of data fields. The length and order is specified in the report descriptor. For example a part of report descriptor can look like:

```
      INPUT(1)[INPUT]
   ..
      Field(2)
        Physical(0020.0073)
        Usage(1)
          0020.045f
        Logical Minimum(-32767)
        Logical Maximum(32767)
        Report Size(8)
        Report Count(1)
        Report Offset(16)
        Flags(Variable Absolute)
  ..
  ..
```

The report is indicating "sensor page (0x20)" contains an accelerometer-3D (0x73). This accelerometer-3D has some fields. Here for example field 2 is motion intensity (0x045f) with a logical minimum value of -32767 and logical maximum of 32767. The order of fields and length of each field is important as the input event raw data will use this format.

Implementation

This specification defines many different types of sensors with different sets of data fields. It is difficult to have a common input event to user space applications, for different sensors. For example an accelerometer can send X,Y and Z data, whereas an ambient light sensor can send illumination data. So the implementation has two parts:

- Core HID driver
- Individual sensor processing part (sensor drivers)

Core driver

The core driver (hid-sensor-hub) registers as a HID driver. It parses report descriptors and identifies all the sensors present. It adds an MFD device with name HID-SENSOR-xxxx (where xxxx is usage id from the specification).

For example:

HID-SENSOR-200073 is registered for an Accelerometer 3D driver.

So if any driver with this name is inserted, then the probe routine for that function will be called. So an accelerometer processing driver can register with this name and will be probed if there is an accelerometer-3D detected.

The core driver provides a set of APIs which can be used by the processing drivers to register and get events for that usage id. Also it provides parsing functions, which get and set each input/feature/output report.

Individual sensor processing part (sensor drivers)

The processing driver will use an interface provided by the core driver to parse the report and get the indexes of the fields and also can get events. This driver can use IIO interface to use the standard ABI defined for a type of sensor.

Core driver Interface

Callback structure:

```
Each processing driver can use this structure to set some callbacks.
      int (*suspend)(..): Callback when HID suspend is received
      int (*resume)(..): Callback when HID resume is received
      int (*capture_sample)(..): Capture a sample for one of its data fields
      int (*send_event)(..): One complete event is received which can have
                             multiple data fields.
```

Registration functions:

```
int sensor_hub_register_callback(struct hid_sensor_hub_device *hsdev,
                     u32 usage_id,
                     struct hid_sensor_hub_callbacks *usage_callback):
```

Registers callbacks for a usage id. The callback functions are not allowed to sleep:

```
int sensor_hub_remove_callback(struct hid_sensor_hub_device *hsdev,
                       u32 usage_id):
```

Removes callbacks for a usage id.

Parsing function:

```
int sensor_hub_input_get_attribute_info(struct hid_sensor_hub_device *hsdev,
                       u8 type,
                       u32 usage_id, u32 attr_usage_id,
                       struct hid_sensor_hub_attribute_info *info);
```

A processing driver can look for some field of interest and check if it exists in a report descriptor. If it exists it will store necessary information so that fields can be set or get individually. These indexes avoid searching every time and getting field index to get or set.

Set Feature report:

```
int sensor_hub_set_feature(struct hid_sensor_hub_device *hsdev, u32 report_id,
                       u32 field_index, s32 value);
```

This interface is used to set a value for a field in feature report. For example if there is a field report_interval, which is parsed by a call to sensor_hub_input_get_attribute_info before, then it can directly set that individual field:

```
int sensor_hub_get_feature(struct hid_sensor_hub_device *hsdev, u32 report_id,
                       u32 field_index, s32 *value);
```

This interface is used to get a value for a field in input report. For example if there is a field report_interval, which is parsed by a call to sensor_hub_input_get_attribute_info before, then it can directly get that individual field value:

```
int sensor_hub_input_attr_get_raw_value(struct hid_sensor_hub_device *hsdev,
                       u32 usage_id,
                       u32 attr_usage_id, u32 report_id);
```

This is used to get a particular field value through input reports. For example accelerometer wants to poll X axis value, then it can call this function with the usage id of X axis. HID sensors can provide events, so this is not necessary to poll for any field. If there is some new sample, the core driver will call registered callback function to process the sample.

HID Custom and generic Sensors

HID Sensor specification defines two special sensor usage types. Since they don't represent a standard sensor, it is not possible to define using Linux IIO type interfaces. The purpose of these sensors is to extend the functionality or provide a way to obfuscate the data being communicated by a sensor. Without knowing the mapping between the data and its encapsulated form, it is difficult for an application/driver to determine what data is being communicated by the sensor. This allows some differentiating use cases, where vendor can provide applications. Some common use cases are debug other sensors or to provide some events like keyboard attached/detached or lid open/close.

To allow application to utilize these sensors, here they are exported using sysfs attribute groups, attributes and misc device interface.

An example of this representation on sysfs:

```
/sys/devices/pci0000:00/INT33C2:00/i2c-0/i2c-INT33D1:00/0018:8086:09FA.0001/HID-SENSOR-2000e1.6.auto$ tree
.
â”‚Â Â  â”œâ”€â”€ enable_sensor
â”‚Â Â  â”œâ”Â Â  â”œâ”€â”€ feature-0-200316
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ feature-0-200316-maximum
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ feature-0-200316-minimum
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ feature-0-200316-name
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ feature-0-200316-size
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ feature-0-200316-unit-expo
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ feature-0-200316-units
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ feature-0-200316-value
â”‚Â Â  â”œâ”Â Â  â”œâ”€â”€ feature-1-200201
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ feature-1-200201-maximum
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ feature-1-200201-minimum
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ feature-1-200201-name
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ feature-1-200201-size
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ feature-1-200201-unit-expo
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ feature-1-200201-units
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ feature-1-200201-value
â”‚Â Â  â”œâ”Â Â  â”œâ”€â”€ input-0-200201
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ input-0-200201-maximum
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ input-0-200201-minimum
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ input-0-200201-name
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ input-0-200201-size
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ input-0-200201-unit-expo
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ input-0-200201-units
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ input-0-200201-value
â”‚Â Â  â”œâ”Â Â  â”œâ”€â”€ input-1-200202
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ input-1-200202-maximum
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ input-1-200202-minimum
â”‚Â Â  â”œâ”Â Â  â”œâ”Â Â  â”œâ”€â”€ input-1-200202-name
```

```
â”‚Â Â  â”‚Â Â  â”‚Â Â  â”œâ”€â”€ input-1-200202-size
â”‚Â Â  â”‚Â Â  â”‚Â Â  â”œâ”€â”€ input-1-200202-unit-expo
â”‚Â Â  â”‚Â Â  â”‚Â Â  â”œâ”€â”€ input-1-200202-units
â”‚Â Â  â”‚Â Â  â”‚Â Â  â””â”€â”€ input-1-200202-value
```

Here there is a custom sensor with four fields: two feature and two inputs. Each field is represented by a set of attributes. All fields except the "value" are read only. The value field is a read-write field.

Example:

```
/sys/bus/platform/devices/HID-SENSOR-2000e1.6.auto/feature-0-200316$ grep -r . *
feature-0-200316-maximum:6
feature-0-200316-minimum:0
feature-0-200316-name:property-reporting-state
feature-0-200316-size:1
feature-0-200316-unit-expo:0
feature-0-200316-units:25
feature-0-200316-value:1
```

How to enable such sensor?

By default sensor can be power gated. To enable sysfs attribute "enable" can be used:

```
$ echo 1 > enable_sensor
```

Once enabled and powered on, sensor can report value using HID reports. These reports are pushed using misc device interface in a FIFO order:

```
/dev$ tree | grep HID-SENSOR-2000e1.6.auto
â”‚Â Â  â”‚Â Â  â”œâ”€â”€ 10:53 -> ../HID-SENSOR-2000e1.6.auto
â”‚Â Â  â”‚Â Â  â””â”€â”€ HID-SENSOR-2000e1.6.auto
```

Each report can be of variable length preceded by a header. This header consists of a 32-bit usage id, 64-bit time stamp and 32-bit length field of raw data.