The Basic Device Structure

See the kerneldoc for the struct device.

Programming Interface

The bus driver that discovers the device uses this to register the device with the core:

```
int device_register(struct device * dev);
```

The bus should initialize the following fields:

- parent
- name
- bus_id
- bus

A device is removed from the core when its reference count goes to 0. The reference count can be adjusted using:

```
struct device * get_device(struct device * dev);
void put_device(struct device * dev);
```

get_device() will return a pointer to the struct device passed to it if the reference is not already 0 (if it's in the process of being removed already).

A driver can access the lock in the device structure using:

```
void lock_device(struct device * dev);
void unlock_device(struct device * dev);
```

Attributes

```
struct device_attribute {
      struct attribute        attr;
      ssize_t (*show)(struct device *dev, struct device_attribute *attr,
                      char *buf);
      ssize_t (*store)(struct device *dev, struct device_attribute *attr,
                       const char *buf, size_t count);
};
```

Attributes of devices can be exported by a device driver through sysfs.

Please see Documentation/filesystems/sysfs.rst for more information on how sysfs works.

As explained in Documentation/core-api/kobject.rst, device attributes must be created before the KOBJ_ADD uevent is generated. The only way to realize that is by defining an attribute group.

Attributes are declared using a macro called DEVICE_ATTR:

```
#define DEVICE_ATTR(name,mode,show,store)
```

Example::

```
static DEVICE_ATTR(type, 0444, type_show, NULL);
static DEVICE_ATTR(power, 0644, power_show, power_store);
```

Helper macros are available for common values of mode, so the above examples can be simplified to::

```
static DEVICE_ATTR_RO(type);
static DEVICE_ATTR_RW(power);
```

This declares two structures of type struct device_attribute with respective names 'dev_attr_type' and 'dev_attr_power'. These two attributes can be organized as follows into a group:

```
static struct attribute *dev_attrs[] = {
      &dev_attr_type.attr,
      &dev_attr_power.attr,
      NULL,
};

static struct attribute_group dev_group = {
      .attrs = dev_attrs,
};

static const struct attribute_group *dev_groups[] = {
      &dev_group,
```

```
        NULL,
};
```

A helper macro is available for the common case of a single group, so the above two structures can be declared using::

```
ATTRIBUTE_GROUPS(dev);
```

This array of groups can then be associated with a device by setting the group pointer in struct device before device_register() is invoked:

```
dev->groups = dev_groups;
device_register(dev);
```

The device_register() function will use the 'groups' pointer to create the device attributes and the device_unregister() function will use this pointer to remove the device attributes.

Word of warning: While the kernel allows device_create_file() and device_remove_file() to be called on a device at any time, userspace has strict expectations on when attributes get created. When a new device is registered in the kernel, a uevent is generated to notify userspace (like udev) that a new device is available. If attributes are added after the device is registered, then userspace won't get notified and userspace will not know about the new attributes.

This is important for device driver that need to publish additional attributes for a device at driver probe time. If the device driver simply calls device_create_file() on the device structure passed to it, then userspace will never be notified of the new attributes.