people.csail.mit.edu

Advanced usage

Although the techniques described in this chapter so far should be sufficient for most Bluetooth applications with simple and straightforward requirements, some applications may require more advanced functionality or finer control over the Bluetooth system resources. This section describes asynchronous device detection and the `_bluetooth` module.

3.5.1. Asynchronous device discovery

The device discovery and remote name request methods described earlier are both synchronous methods in that they don't return until the requests are complete, which can often taken a long time. During this time, the controlling thread blocks and can't do anything else, such as responding to user input or displaying other information. To avoid this, PyBluez provides the `DeviceDiscoverer` class for asynchronous device discovery and name lookup.

Example 3-8. asynchronous-inquiry.py

```
import bluetooth
import select

class MyDiscoverer(bluetooth.DeviceDiscoverer):

    def pre_inquiry(self):
        self.done = False

    def device_discovered(self, address, device_class, name):
        print "%s - %s" % (address, name)

    def inquiry_complete(self):
        self.done = True

d = MyDiscoverer()
d.find_devices(lookup_names = True)

readfiles = [ d, ]

while True:
    rfds = select.select( readfiles, [], [] )[0]

    if d in rfds:
        d.process_event()

    if d.done: break
```

To asynchronously detect nearby bluetooth devices, create a subclass of `DeviceDiscoverer` and override the `pre_inquiry`, `device_discovered`, and `inquiry_complete` methods. To start the discovery process, invoke `find_devices`, which returns immediately. `pre_inquiry` is called immediately before the actual inquiry process begins, and `inquiry_complete` is called as soon as the process completes.

`MyDiscoverer` exposes a `fileno` method, which allows it to be used with the `select` module. This provides a way for a single thread of control to wait for events on many open files at once, and greatly simplifies event-driven programs.

Call `process_event` to have the `DeviceDiscoverer` process pending events, which can be either a discovered device or the inquiry completion. When a nearby device is detected, `device_discovered` is invoked, with the address and device class of the detected device. If `lookup_names` was set in the call to `find_devices`, then `name` will also be set to the user-friendly name of the device. For more information about device classes, see https://www.bluetooth.org/foundry/assignnumb/document/baseband. The `DeviceDiscoverer` class can be used directly with the `select` module, and can easily be integrated into event loops of existing applications.

3.5.2. The `_bluetooth` module

The `bluetooth` module provides classes and utility functions useful for the most common Bluetooth programming tasks. More advanced functionality can be found in the `_bluetooth` extension module, which is little more than a thin wrapper around the BlueZ C API described in the next chapter. Lower level Bluetooth operations, such as establishing a connection with the actual Bluetooth microcontroller on the local machine and reading signal strength information, can be performed with the `_bluetooth` module in most cases without having to resort to the C API. An overview of the classes and methods available in `_bluetooth` is beyond the scope of this chapter, but the module documentation and examples are provided with the PyBluez distribution.