people.csail.mit.edu

Communicating with RFCOMM

Bluetooth programming in Python follows the socket programming model. This is a concept that should be familiar to almost all network programmers, and makes the transition from Internet programming to Bluetooth programming much simpler. Example 3-2 and Example 3-3 show how to establish a connection using an RFCOMM socket, transfer some data, and disconnect.

Example 3-2. rfcomm-server.py

```
import bluetooth

server_sock=bluetooth.BluetoothSocket( bluetooth.RFCOMM )

port = 1
server_sock.bind(("",port))
server_sock.listen(1)

client_sock,address = server_sock.accept()
print "Accepted connection from ",address

data = client_sock.recv(1024)
print "received [%s]" % data

client_sock.close()
server_sock.close()
```

Example 3-3. rfcomm-client.py

```
import bluetooth

bd_addr = "01:23:45:67:89:AB"

port = 1

sock=bluetooth.BluetoothSocket( bluetooth.RFCOMM )
sock.connect((bd_addr, port))
```

```
sock.send("hello!!")

sock.close()
```

In the socket programming model, a socket represents an endpoint of a communication channel. Sockets are not connected when they are first created, and are useless until a call to either `connect` (client application) or `accept` (server application) completes successfully. Once a socket is connected, it can be used to send and receive data until the connection fails due to link error or user termination.

PyBluez currently supports two types of `BluetoothSocket` objects: RFCOMM and L2CAP. The RFCOMM socket, shown above, is created by passing `RFCOMM` as an argument to the `BluetoothSocket` constructor. As the name suggests, it allocates resources for an RFCOMM based communication channel. The second type of `BluetoothSocket`, the L2CAP socket, is described in the next section.

An RFCOMM `BluetoothSocket` used to accept incoming connections must be attached to operating system resources with the `bind` method. `bind` takes in a tuple specifying the address of the local Bluetooth adapter to use and a port number to listen on. Usually, there is only one local Bluetooth adapter or it doesn't matter which one to use, so the empty string indicates that any local Bluetooth adapter is acceptable. Once a socket is bound, a call to `listen` puts the socket into listening mode and it is then ready to accept incoming connections.

The RFCOMM `BluetoothSocket` used to establish an outgoing connection connects to its target with the `connect` method, which also takes a tuple specifying an address and port number. In Example 3-3 , the client tries to connect to the Bluetooth device with address ``01:23:45:67:89:AB" on port 1. This example, and Example 3-2 , assumes that all communication happens on RFCOMM port 1. Section 3.4 shows how to dynamically choose ports and use SDP to search for which port a server is operating on.

Error handling code has been omitted for clarity in the examples, but is fairly straightforward. If any of the Bluetooth operations fail for some reason (e.g. connection timeout, no local bluetooth resources are available, etc.) then a `BluetoothError` is raised with an error message indicating the reason for failure.