Using Bluetooth

By Ben DuPont, January 31, 2012

8 Comments

Trying to communicate with a remote device with no other familiar protocol? Bluetooth provides an easy answer with well-documented specs and straightforward programming APIs.

Register with the Service Directory Protocol (SDP) Server

The SDP server is an integral part of a Bluetooth system. Services register with the local SDP; remote devices query the SDP to find out how to connect to particular services. In our example, we're going to register the HSP service.

BlueZ provides a tool for communicating with local and remote SDP servers: sdptool. To view the list of services offered by your machine, run this command: `dptool browse local`. We can use this command to determine whether our program has correctly registered with the SDP server. Here's output from my SDP server:

```
Browsing FF:FF:FF:00:00:00 ...
Service Name: Headset Audio Gateway
Service RecHandle: 0x10000
Service Class ID List:
  "Headset Audio Gateway" (0x1112)
  "Generic Audio" (0x1203)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 12
Profile Descriptor List:
  "Headset" (0x1108)
    Version: 0x0102

Service Name: Hands-Free Audio Gateway
Service RecHandle: 0x10001
Service Class ID List:
  "Handsfree Audio Gateway" (0x111f)
  "Generic Audio" (0x1203)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 13
Profile Descriptor List:
  "Handsfree" (0x111e)
    Version: 0x0105
```

Notice that the first two entries reference headset and hands-free; those services need to be disabled so they don't interfere with our program. On my machine, I had to modify `/etc/Bluetooth/audio.conf`. Under the `[General]` section, add this line:

```
Disable=Headset,Gateway
```

After modifying the config file, restart Bluetooth by running `sudo service Bluetooth restart`, and check that the SDP server no longer has references to the headset or hands-free profiles.

The next piece of code, Listing Two, shows how to register the the headset (HS) side of the headset profile. The code is largely borrowed from two sources I found on the Web.

Listing Two

```
#include "btinclude.h"

// source adapted from:     // http://people.csail.mit.edu/albert/bluez-intro/x604.html and
// http://nohands.sourceforge.net/source.html (libhfp/hfp.cpp: SdpRegister)                uint8_t channel = 3;

int main()
{
    const char *service_name = "HSP service";
    const char *service_dsc = "HSP";
    const char *service_prov = "nebland software, LLC";

    uuid_t hs_uuid, ga_uuid;

```

```
    sdp_profile_desc_t desc;

    uuid_t root_uuid, l2cap_uuid, rfcomm_uuid;
    sdp_list_t *l2cap_list = 0,
               *rfcomm_list = 0,
               *root_list = 0,
               *proto_list = 0,
               *access_proto_list = 0;

    sdp_data_t *channel_d = 0;

    int err = 0;
    sdp_session_t *session = 0;

    sdp_record_t *record = sdp_record_alloc();

// set the name, provider, and description
    sdp_set_info_attr(record, service_name, service_prov, service_dsc);

// service class ID (HEADSET)
    sdp_uuid16_create(&hs_uuid, HEADSET_SVCLASS_ID);

    if (!(root_list = sdp_list_append(0, &hs_uuid)))
        return -1;

    sdp_uuid16_create(&ga_uuid, GENERIC_AUDIO_SVCLASS_ID);

    if (!(root_list = sdp_list_append(root_list, &ga_uuid)))
        return -1;

    if (sdp_set_service_classes(record, root_list) < 0)
        return -1;

    sdp_list_free(root_list, 0);
    root_list = 0;

// make the service record publicly browsable
    sdp_uuid16_create(&root_uuid, PUBLIC_BROWSE_GROUP);

    root_list = sdp_list_append(0, &root_uuid);
    sdp_set_browse_groups( record, root_list );

// set l2cap information
    sdp_uuid16_create(&l2cap_uuid, L2CAP_UUID);
    l2cap_list = sdp_list_append( 0, &l2cap_uuid );
    proto_list = sdp_list_append( 0, l2cap_list );

// set rfcomm information
    sdp_uuid16_create(&rfcomm_uuid, RFCOMM_UUID);
    channel_d = sdp_data_alloc(SDP_UINT8, &channel);
    rfcomm_list = sdp_list_append( 0, &rfcomm_uuid );

    sdp_list_append( rfcomm_list, channel_d );
    sdp_list_append( proto_list, rfcomm_list );

// attach protocol information to service record
    access_proto_list = sdp_list_append( 0, proto_list );
    sdp_set_access_protos( record, access_proto_list );

    sdp_uuid16_create(&desc.uuid, HEADSET_PROFILE_ID);

// set the version to 1.0
    desc.version = 0x0100;

    if (!(root_list = sdp_list_append(NULL, &desc)))
        return -1;

    if (sdp_set_profile_descs(record, root_list) < 0)
        return -1;

// connect to the local SDP server and register the service record
    session = sdp_connect( BDADDR_ANY, BDADDR_LOCAL, SDP_RETRY_IF_BUSY );
    err = sdp_record_register(session, record, 0);

// cleanup
    sdp_data_free( channel_d );
    sdp_list_free( l2cap_list, 0 );
    sdp_list_free( rfcomm_list, 0 );
    sdp_list_free( root_list, 0 );
    sdp_list_free( access_proto_list, 0 );

    while (1)
        sleep(5000);

    return err;
}
```

The code essentially builds lists of values that are registered with the SDP server. The constants used in the code, like HEADSET_SVCCLASS_ID and GENERIC_AUDIO_SVC_CLASS_ID, are defined specifically from the HSP profile. The values for the headset side of the profile are listed on page 220 of the HSP profile document. Page 221 lists the values for the audio gateway side of the profile.

When a Bluetooth program exits, any services registered with the SDP server are removed. The test program waits in an infinite loop so you can run sdptool and verify that the service is indeed registered. Here's what the service should look like when you run the sdptool command:

```
Service Name: HSP service
Service Description: HSP
Service Provider: nebland software, LLC
Service RecHandle: 0x10003
Service Class ID List:
  "Headset" (0x1108)
  "Generic Audio" (0x1203)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 3
Profile Descriptor List:
  "Headset" (0x1108)
    Version: 0x0100
```

The SDP code registers a specific profile with the SDP server. Besides describing the profile that our service provides (HSP in this example), the code also tells the service what RFCOMM channel we will accept connections on. Remote devices that want to use our HSP service will query the SDP server and extract the channel that we registered. When a connection to our service is initiated, the remote device will connect to channel 3.

You might notice that the SDP record is missing the optional parameter that specifies whether remote volume control is supported. It seems that sdp_attr_add or sdp_attr_add_new could be used to add a value for the constant SDP_ATTR_REMOTE_AUDIO_VOLUME_CONTROL to the SDP record, but it didn't seem to work. At least, when I added the parameter, it did not show up in the listing produced by sdptool.

Listening for RFCOMM Connections

When a device initiates an HSP connection, the first connection created between the devices is through RFCOMM. As long as this connection is active, the headset can assume that the audio gateway will, at any moment, initiate a SCO connection. SCO connections are covered shortly.

The code in Listing Three creates a RFCOMM socket and listens for connections on channel 3.

Listing Three

```
#include "btinclude.h"

uint8_t channel = 3;

int main()
{

// socket descriptor for local listener          int sock;
// socket descriptor for remote client           int client;
     unsigned int len = sizeof(struct sockaddr_rc);

// local rfcomm socket address             struct sockaddr_rc remote;
// remote rfcomm socket address             struct sockaddr_rc local;
     char pszremote[18];

// initialize a bluetooth socket
     sock = socket(AF_BLUETOOTH, SOCK_STREAM, BTPROTO_RFCOMM);

     local.rc_family = AF_BLUETOOTH;

// TODO: change this to a local address if you know what
// address to use
     local.rc_bdaddr = *BDADDR_ANY;
     local.rc_channel = channel;

// bind the socket to a bluetooth device
     if (bind(sock, (struct sockaddr *)&local,
```

```
28  sizeof(struct sockaddr_rc)) < 0)
29              return -1;
30
31  // set the listening queue length
32      if (listen(sock, 1) < 0)
33          return -1;
34
35      printf("accepting connections on channel: %d\n", channel);
36
37  // accept incoming connections; this is a blocking call
38      client = accept(sock, (struct sockaddr *)&remote, &len);
39
40      ba2str(&remote.rc_bdaddr, pszremote);
41
42      printf("received connection from: %s\n", pszremote);
43
44      return 0;
```

If you're familiar with BSD-style socket programming, this example should look very familiar. If not, here are the basic steps:

1. Create a socket: line 17
2. Set the Bluetooth address and channel number: lines 23, and 24. If you want to use a specific Bluetooth device, use the `str2ba(...)` function to convert the string to a `bdaddr_t`. For exmaple: `str2ba("00:11:22:33:44:55:66", &local.rc_bdaddr);`
3. Bind the socket to the address and channel: line 27
4. Set the number of clients to queue up to wait for a connection handshake (additional clients will be refused): 35
5. Wait for/accept client connections: line 42

A successful call to accept will return a full duplex socket descriptor. The functions `send` and `recv`should be used to send data to, and receive data from the remote device. This connection is used to exchange AT commands.

The AT commands and formats for HSP are defined starting on page 215 of the document. Various actions initiated on the handset, like adjusting the volume, will generate an AT command, telling our program of the event. Our program responds to the event by printing out the command and returning the AT OK command. In a real implementation, the service should adjust the hardware (turn the volume down, for example) as requested by the remote client.