Simple Network Discovery Protocol (SNDP)

Ver. 1.02

May 28, 2011

Moe Wheatley, AE4JY

This specification describes a simple protocol that can be used to discover custom network connected devices using a simple broadcast UDP scheme.

Table of Contents

1. Simple Network Discovery Protocol Overview

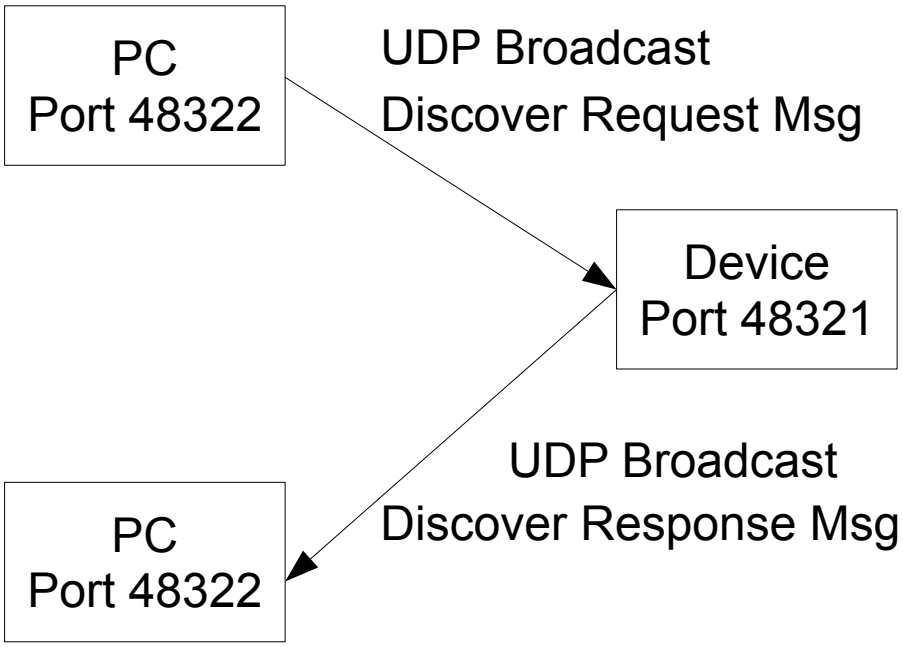
1.1. Concept

A fundamental requirement for connecting to a hardware device on a network is to know the devices network parameters such as IP address, port number, etc. Many devices do not have any user interface to allow setting or changing the network parameters so some other means is required.

This protocol is a simple method to discover networked devices using broadcast UDP messages. The scheme uses the same basic UDP broadcast message scheme as DHCP except the messages are customized to the specific devices supported by the protocol.

The key is that the scheme uses UDP broadcast messages(IP address 255.255.255.255) on a specific port. This allows communication with any device physically connected to a network but whose IP and port settings are unknown as well as changing these settings remotely. These messages will only work on local networks.

1.2. Discovery Procedure

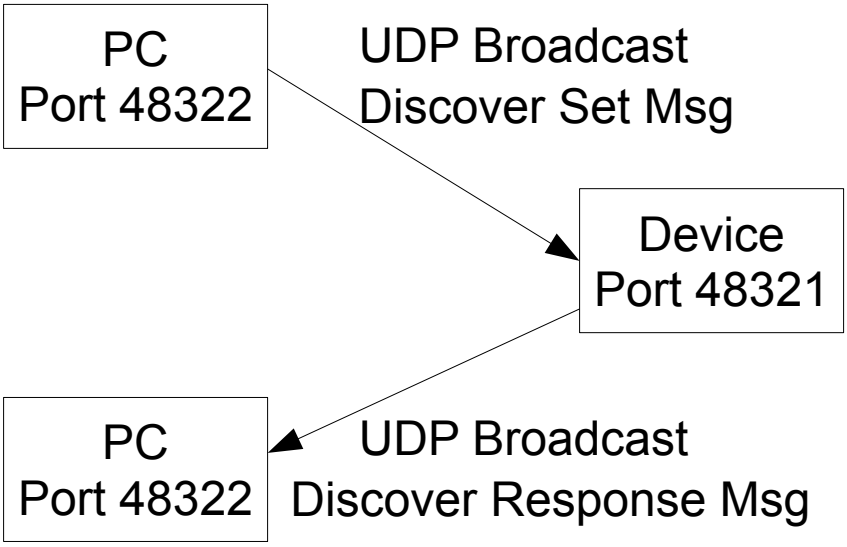


The sequence of events are as follows for obtaining a devices parameters:

1. PC sends a UDP broadcast Request Message to a device that is listening on Port 48321.
2. The device then sends a UDP broadcast Response Message to the PC which is listening on Port 48322. The response message contains its pertinent network parameters.

1.3. Modification Procedure

A similar means can be used to set a devices network parameters using a broadcast UDP message:



The PC simply sends a UDP broadcast Set message that when the device receives it, reconfigures its network to the parameters specified in the Set message. The device then sends a UDP broadcast Response Message back to the PC containing the changed settings. This response can be used as an acknowledge as well as a means to indicate any fields that could not be set.

This scheme can be expanded to allow discovering and modifying other custom parameters depending on what the device is.

2. Network Discovery Message Format

The discovery message packets are variable length packets containing various byte fields that contain message type information as well as the parameters of the specific device.

The message is divided into two sections. The first section is required and its fields are fixed in length and its definitions are common to all discovery messages and all devices supporting the protocol. The second optional section is variable length and the byte fields are custom to the particular device.

***Note**: the term "little endian byte order" refers to the order that multi byte data items are stored in memory and is least significant byte first order. A 16 bit word would be stored as parameter[0]==bits 7-0, parameter[1]==bits 15-8.*

2.1. Overall Packet Format

The message packet contains a fixed 56 byte field followed by a variable length custom field that is specific to a particular device.

```
 struct DISCOVER_MSG
{               //fixed common 56 byte fields
    unsigned char length[2];        //length of total message in bytes (little endian byte order)
    unsigned char key[2];           //fixed key key[0]==0x5A  key[1]==0xA5
    unsigned char op;               //0==Request(to device)  1==Response(from device) 2 ==Set(to device)
    char name[16];                  //Device name string null terminated
    char sn[16];                    //Serial number string null terminated
    unsigned char ipaddr[16];       //device IP address (little endian byte order)
    unsigned char port[2];          //device Port number (little endian byte order)
    unsigned char customfield;      //Specifies a custom data field for a particular device
..
                //start of optional variable custom byte fields
    unsigned char Custom[N];
}
```

2.1.1 Fixed Common Field Definitions

The first 2 byte 16 bit field 'length' is the total message length in little endian byte order.

The second 2 byte 16 bit field 'key' is a fixed value used to identify this packet as a discovery packet in the rare case there are other broadcast packets on this port.

The third 1 byte field 'op' identifies the type of discovery message. A value of 0 is defined as a Request message from the PC client to the device. A value of 1 is a Response message from the device to the PC, and a value of 2 is a Set message from the PC to the device. All other values are reserved.

The fourth fixed 16 byte 'name' field is a variable zero terminated character string identifying the external device. Any unused bytes in the 16 byte field should be set to zero. This string is the primary identifier of the device on the network. A device must always fill in this field when sending a response message back to the PC.

The PC can leave this field all zeros when sending a request message and all devices regardless of their name to discover all devices with any name on the network.

The fifth fixed 16 byte 'sn' field is a variable zero terminated character string identifying the serial number of the external device. Any unused bytes in the 16 byte field should be set to zero. This string can be used to distinguish multiple devices with the same name on the network. A device must always fill in this field when sending a response message back to the PC.

The PC can leave this field all zeros when sending a request message to discover all devices with the same name on the network.

The sixth fixed 16 byte 'ipaddr' field is the IP address of the device in little endian byte order. If this is a request message, the field is ignored by the device. If it is a set message then this is the IP address that the device should change to. If it is a response message from the device, this is the current IP address of the device.

The seventh fixed 2 byte 'port' field is the port number of the device in little endian byte order. If this is a request message, the field is ignored by the device. If it is a set message then this is the port number that the device should change to. If it is a response message from the device, this is the current port number of the device.

The last byte of the fixed common field can be used to specify different custom field formats for the same device.

2.1.2 Examples of Filling fixed common fields.

```
DISCOVER_MSG msg;
```

Example PC request message for any devices on network:

```
memset((void*)&msg, 0, sizeof(DISCOVER_MSG));       //zero out all fields
msg.key[0] = 0x5A;                                  //fill in key values
msg.key[1] = 0xA5;
msg.op = 0;                                         //request message
msg.length[0] = 56;                                 //set message length of just the fixed section
```

Example PC request message for any device with the name “MyDevice” on the network:

```
memset((void*)&msg, 0, sizeof(DISCOVER_MSG));       //zero out all fields
msg.key[0] = 0x5A;                                  //fill in key values
msg.key[1] = 0xA5;
msg.op = 0;                                         //request message
strcpy( msg.name, “MyDevice” );                     //fill in name string field
msg.length[0] = 56;                                 //set message length of just the fixed section
```

Example simple Device response message with the name “MyDevice” on the network:

```
memset((void*)&msg, 0, sizeof(DISCOVER_MSG));       //zero out all fields
msg.key[0] = 0x5A;                                  //fill in key values
msg.key[1] = 0xA5;
msg.op = 1;                                         //response message
strcpy( msg.name, “MyDevice” );                     //fill in name string field
        //assume device IP address is 192.169.1.100 on port 12345
msg.ipaddr[0] = 100;                                //device IP address( little endian byte order)
msg.ipaddr[1] = 1;
msg.ipaddr[2] = 168;
msg.ipaddr[3] = 192;
msg.port[0] = 12345&0x00FF;                         //device Port Number( little endian byte order)
msg.port[1] = 12345>>8;
msg.length[0] = 56;                                 //set message length of just the fixed section
```

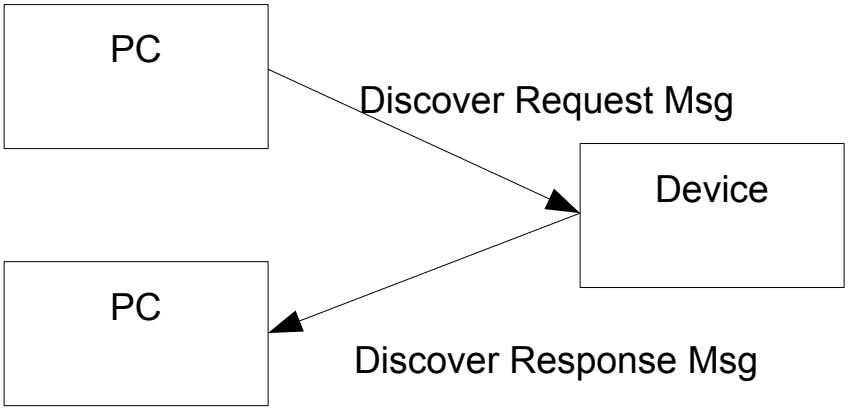
Example PC set message for device with the name “MyDevice” on the network:

```
memset((void*)&msg, 0, sizeof(DISCOVER_MSG));       //zero out all fields
msg.key[0] = 0x5A;                                  //fill in key values
msg.key[1] = 0xA5;
msg.op = 2;                                         //set message
strcpy( msg.name, “MyDevice” );                     //fill in name string field
        //assume want to set device IP address to 192.169.5.32 on port 54321
msg.ipaddr[0] = 32;                                 //device IP address( little endian byte order)
msg.ipaddr[1] = 5;
msg.ipaddr[2] = 168;
msg.ipaddr[3] = 192;
msg.port[0] = 54321&0x00FF;                         //device Port number( little endian byte order)
```

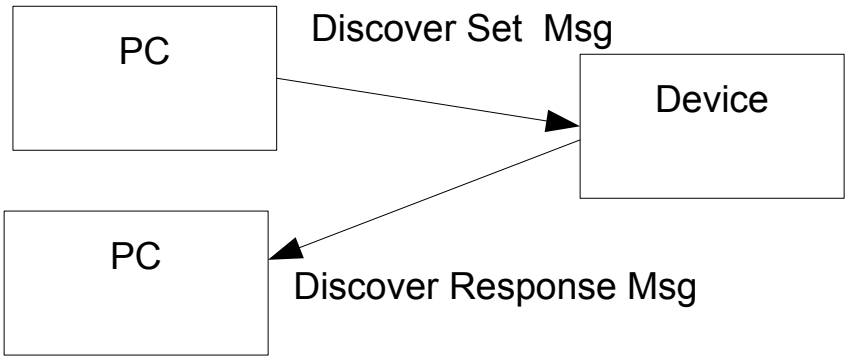
```
msg.port[1] = 54321>>8;
msg.length[0] = 56;                    //set message length of just the fixed section
```

2.1.3 Example of Modifying a Devices Settings using Read Modify Write.

One can save work by first reading all the devices settings, modify the fields that need to change, then send the message back out as a Set message to the device.



Got filled in fields from device so only update fields that are to be changed then send Set message with modified fields.



2.2. Custom Field Packet Formats

The custom fields are defined for specific devices as they contain information unique to that device. These fields are appended after the fixed common fields of the discover message with the total length of the message adjusted to include the extra bytes of the custom fields. The following sections describe currently supported devices.

2.2.1 SDR-IP and NetSdr Custom Field Packet Definitions

```
 struct _DISCOVER_MSG_NETSDR
{               //fixed common 56 byte fields
    unsigned char length[2];        //length of total message in bytes (little endian byte order)
    unsigned char key[2];           //fixed key key[0]==0x5A  key[1]==0xA5
    unsigned char op;               //0==Request(to device)  1==Response(from device) 2 ==Set(to device)
    char name[16];                  //Device name string null terminated
    char sn[16];                    //Serial number string null terminated
    unsigned char ipaddr[16];       //device IP address (little endian byte order)
    unsigned char port[2];          //device Port number (little endian byte order)
    unsigned char customfield;      //Specify a custom data field for a particular device
            //start of optional variable custom byte fields
    unsigned char macaddr[6];       //HW mac address (little endian byte order) (read only)
    unsigned char hwver[2];         //Hardware version*100  (little endian byte order) (read only)
    unsigned char fwver[2];         //Firmware version*100 (little endian byte order)(read only)
    unsigned char btver[2];         //Boot version*100 (little endian byte order) (read only)
    unsigned char fpgaid;           //FPGA ID (read only)
    unsigned char fpgarev;          //FPGA revision (read only)
    unsigned char opts;             //Options (read only)
    unsigned char mode;             //0 == Use DHCP 1==manual  2==manual Alternate data address
    unsigned char subnet[4];        //IP subnet mask (little endian byte order)
    unsigned char gwaddr[4];        //gateway address (little endian byte order)
    unsigned char dataipaddr[4];    // Alternate data IP address for UDP data  (little endian byte order)
    unsigned char dataport[2];      // Alternate data Port address for UDP (little endian byte order)
    unsigned char fpga;             //0 == default cfg   1==custom1    2==custom2
    unsigned char status;           //Bit 0 == TCP Connected, Bit 1 == Running, Bits 2-7 not defined (read only)
    unsigned char future[15];       //future use
}
```

2.2.2 SDR-IQ and SDR-14 Using Server Custom Field Packet Definitions

```
 struct _DISCOVER_MSG_SDRXX
{               //fixed common 56 byte fields
    unsigned char length[2];        //length of total message in bytes (little endian byte order)
    unsigned char key[2];           //fixed key key[0]==0x5A  key[1]==0xA5
    unsigned char op;               //0==Request(to device)  1==Response(from device) 2 ==Set(to device)
    char name[16];                  //Device name string null terminated
    char sn[16];                    //Serial number string null terminated
    unsigned char ipaddr[16];       //device IP address (little endian byte order)
    unsigned char port[2];          //device Port number (little endian byte order)
    unsigned char customfield;      //Specify a custom data field for a particular device
            //start of optional variable custom byte fields
    unsigned char fwver[2];         //Firmware version*100 (little endian byte order)(read only)
    unsigned char btver[2];         //Boot version*100 (little endian byte order) (read only)
    unsigned char subnet[4];        //IP subnet mask (little endian byte order)
    unsigned char gwaddr[4];        //gateway address (little endian byte order)
    char connection[32];            //interface connection string null terminated(ex: COM3, DEVTTY5, etc)
    unsigned char status;           //Bit 0 == TCP Connected, Bits 1-7 not defined (read only)
    unsigned char future[15];       //future use
}
```