



RoboticsConnection.com

## **Shared Serial Device Protocol (SSDP)**

2/6/2010

V1.0

Developed by: Jason Summerour and Amal Graafstra

## Introduction

The Shared Serial Device Protocol (SSDP) is a simple protocol that was developed to allow independent, non similar devices that leverage XBee modules, to communicate with each other using the XBee Broadcast Personal Area Network (BPAN) functionality.

We have developed several devices that use XBee modules, and felt it would be powerful to develop such a protocol to allow them to communicate with each other.

While it's useful for a single host application to communicate wirelessly with a single device (such as a RedBee™ RFID Reader) using XBee technology, it would be even more useful and powerful if you easily communicate with multiple devices simultaneously. This might include a single host talking to multiple RFID readers, or multiple hosts talking to one RFID reader. It might even be a mix of hosts, RFID readers, and other XBee enabled devices yet to be developed.

We like to think of this protocol as a 'wrapper' around our more specific protocols used in our RedBee™ RFIDReader and Serializer devices.

The protocol is simple and open, and we welcome improvements.

## Device Ids

All devices that want to leverage SSDP must have a Device Id, or *did*. The '*did*' identifies a unique module on the network, and basically serves as its network address. There can be up to 255 unique dids operating simultaneously on the BPAN. There cannot be more than one device using the same did at a time connected to the BPAN. Undefined results will occur if multiple devices using the same did are communicating across the BPAN. Additionally, devices must provide an API for setting the did, so that it can be changed per device.

## Boot Prompt

When devices that use the XBPP boot up, they must send out a message identifying the following attributes about the device:

- Device Type (dname)
- Device Id (did)
- Device Firmware Version
- Device Info (free form string, less than 100 bytes)

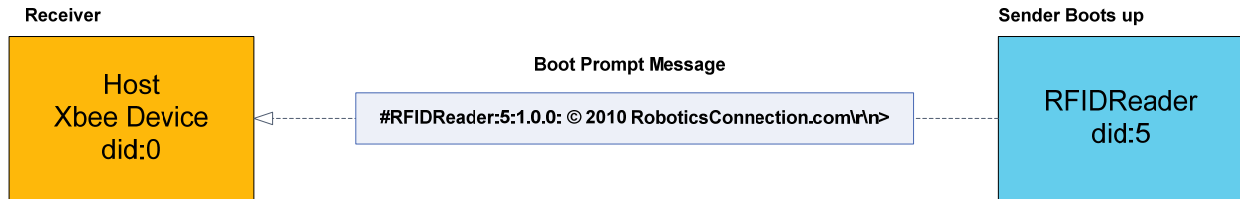
The format of the boot message is as follows:

```
#<dname>:<did>:<fw version>:<device info string>\r\n
```

Here's an example of the RedBee™ RFID Reader boot prompt, configured with a Device Id of 5, firmware version of 1.0.0, and a free form string of "© 2010 RoboticsConnection.com".

```
#RFIDReader:5:1.0.0:(c)2010 RoboticsConnection.com\r\n>
```

### **Example:**



Keep in mind that device buffers could be small, so you might want to keep the free form device info string short (something less than 100 characters).

Other devices listening on the XBee BPAN can notice when new devices connect to the network thanks to the boot message, and use that information to communicate with them. As an example, there could be a host application that dynamically adds RFID readers as they connect to the BPAN.

## **Commands/Responses**

All commands begin with a '@' character, and terminate with a '\r' character sequence. The '\r' is a Carriage Return character (0xD).

Devices need to know where commands come from, so that they can respond back to the appropriate calling device. Therefore, the '@' character is followed by a "fromDid/toDid" addressing sequence. The "fromDid" identifies the senders Device Id, and the "toDid" identifies the destination target Device Id, and are always separated by a forward slash character '/'. Finally, the "fromDid/toDid" addressing sequence ends with a ':' character, to separate it from the command payload. The entire command format is shown below:

### **Command Format:**

```
@<fromDid>/<toDid>:<command payload>\r
```

The receiver responds wraps the response with almost the same format as a command, except that it terminates with a '\r\n>' character sequence. The '\r' and '\n' characters are invisible, and were added to allow users to use terminal apps to develop and debug their devices using the XBPP. The additional '\r\n>' character sequence allows responses to print pretty in terminal applications, hence the reason we added it. Other devices on the BPAN using SSDP, will see the '\r' in the '\r\n>' sequence, but should ignore the '\n' and '>' characters. Once those devices see the '\r', they will process the message, and toss it aside, since it won't be addressed to them.

**Response Format:**

@<fromDid>/<toDid>:<command payload>\r\r>

It's important to note that the response will now have the original fromDid/toDid values swapped, since the original recipient is sending a response back to the original caller. Let's look at an example to clarify the command/response format .

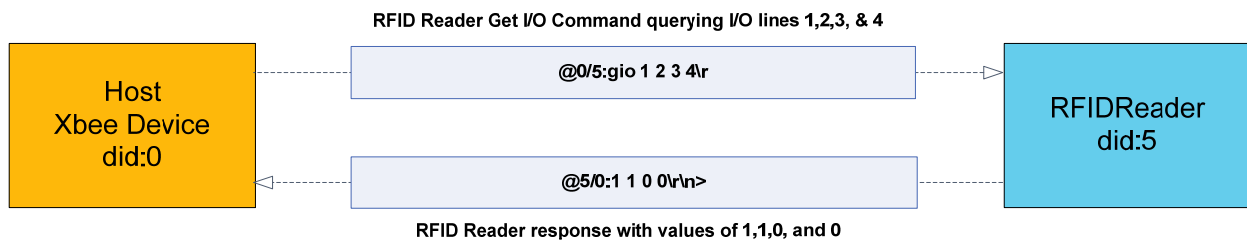
**Device 5 sends a command to device 3:**

@5/3:<command>\r

**Device 3 responds back to device 5:**

@3/5:<response>\r\n>

**Example:**



## Device Ping

Once a device comes onto the BPAN, there needs to be a way to ensure that it's still there as time passes by. If a device loses power, there's no way to inform other devices on the BPAN that it's powering down. Therefore the XBPP supports the notion of a device 'Ping' command. The ping command format is as follows, where the 'fromDid' is that of the device sending the ping command.

@<fromDid>:ping\r

Devices on the BPAN should respond back using the following response format:

@<fromDid>/<toDid>:<dname>:<fw version>\r\n>

This is enough information for the requesting device to understand that another device is still connected to the BPAN. Here's an example to clarify the Ping command operation:

**Device 8 pings all devices on a BPAN:**

@8:ping\r

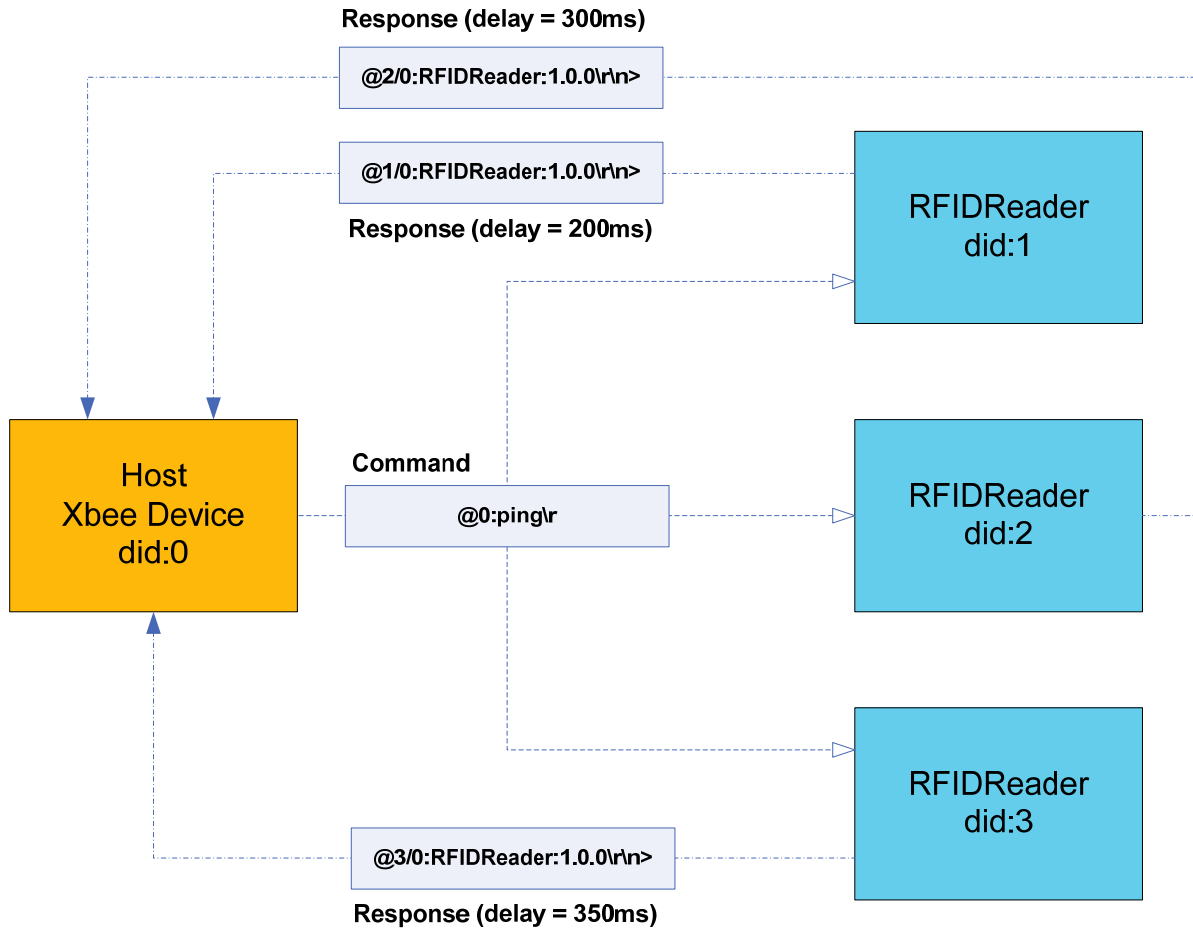
**Devices 2 and 3 respond:**

@2/8:RFIDReader:1.0.0\r\n>

```
@3/8:RFIDReader:1.1.0\r\n>
```

Notice how devices 2 and 3 have differing firmware versions. This information can be useful in knowing device capabilities.

**Example:**



**Ping Response Delay Strategy**

If every device on the BPAN were to respond at exactly the same time, message collisions would occur. Therefore, our devices that leverage XBee modules, and the XBPP implement a ping response delay strategy, where it waits a calculated amount of time before replying to the ping. The calculation used is as follows:

$$\text{Delay time (in msec)} = 200 + (\text{Device Id} * 50 \text{ msec});$$

This would take 255 modules on a BPAN a little under 13 seconds for all modules to respond.

## Broadcast Events

The XBPP supports the notion of events, so that a device can notify other devices of asynchronous events that might be of interest. An event just contains a “fromDid”, and doesn’t include a “toDid”, since its being broadcasted out to ALL devices. Devices not interested in the event can ignore it. Events work like a response, in that it’s terminated with a ‘\r\n>’ character sequence. The Event format is shown below:

```
@<fromDid>:<event>\r\n>
```

Let’s look at an example:

### Device 5 sends out an event:

```
@5:<event>\r\n>
```

Other devices can easily differentiate between a typical response and an event by noticing that an event is missing a “/<toDid>” sequence between the “@<fromDid>” and the “:” character.

### Example:

