

abSerial User Guide

25 July 2011

Contents

1	Preface	3
1.1	Purpose	3
1.2	Definitions and Acronyms	3
2	Overview.....	4
2.1	Modes of Operation	4
2.2	System Configuration.....	4
2.2.1	Hardware	4
2.2.2	Software.....	5
2.3	System Data Flow.....	5
2.4	Commands and Responses.....	6
3	Startup.....	7
4	Configuration	8
4.1	Device configuration	8
4.2	abSerial Interface-to-Host Baud Rate	9
5	Connect with Remote Device	10
5.1	Successful Connection	10
5.2	Unsuccessful Connection	11
6	Disconnect with Remote Device	12
7	Escape from Bypass Mode	14
7.1	Connection Still Active	14
8	Device Discovery	15
9	Bonding and Security.....	17
10	Smart Cable.....	19
10.1	Configuration Parameters	19
10.2	Usage	19
11	Remote Escape Sequence, “@#@\$@%”	20
11.1	Usage	20
12	Power Saving Mode.....	21
12.1	Deep Sleep Mode	21
12.1.1	UART Usage with Deep Sleep	21
12.1.2	Deep Sleep Wakeup	21
12.1.3	Flow Control with Deep Sleep	21
12.2	Sniff	21
12.3	Auto Sniff Mode	22
13	GPIO Usage	23
13.1	abSerial Application GPIO PIN Assignments	23
13.2	GPIO AT Commands.....	23
14	Data Throughput.....	24

1 Preface

The document describes an embedded application that provides a wireless serial cable replacement service using the Bluetooth Serial Port Profile, abSerial.

1.1 Purpose

This document provides guidelines for users of host applications that use the abSerial interface. Such host applications may execute on a personal computer or other device that will primarily pass a serial data stream to the abSerial interface, which will then be transferred using the Bluetooth Serial Port Profile (SPP).

The abSerial interface supports AT-like attention commands for configuration and control. The *abSerial User Guide* explains the commands and sequences needed to use the abSerial interface as a serial port. For a more detailed discussion on each command, please refer to the *abSerial Reference Guide*.

1.2 Definitions and Acronyms

The following acronyms are used in this document.

Table 1. Definitions and Acronyms

Term	Description/Meaning
AT	Text based command standard commonly used for modems
BD	Bluetooth Device
bps	Bits Per Second
CTS	Clear to Send line (hardware flow control input on UART that allows data transmission)
RTS	Ready to Send line (hardware flow control output on UART that stops receiving data)
RxD	Receive Data line (on UART)
SPP	Serial Port Profile
TxD	Transmit Data line (on UART)
UART	Universal Asynchronous Receiver-Transmitter

2 Overview

2.1 Modes of Operation

The software behavior of the abSerial interface is similar to a Hayes-compatible modem. The application has two modes, a command mode and a bypass mode. In the command mode, the host can issue specially formatted text strings called commands. These command strings can be used for configuration or to manage a connection with a remote device. Note that the abSerial interface does not support the standard Hayes AT command set. Instead, it has commands that leverage off the vendor-specific command form.

Once a connection is established, the application transitions to the bypass mode. In the bypass mode, bytes sent from the host will be sent over the Bluetooth link to the remote device. Any data received from the remote device will be delivered to the host.

All bytes received on the UART by the host are transferred to the remote device with the exception of the Escape sequence. While in the bypass mode, the abSerial interface will search for this Escape sequence from the host. If this sequence is found, the application will go back to command mode. This allows commands to be issued again from the host, but the connection to the remote device will remain. Any data received on the Bluetooth link will be discarded while in command mode.

While in the command mode, the abSerial interface will send responses back to the host for commands received. Responses from the abSerial interface will also be sent on system reset. These responses are also referred to as Event strings in this document. Event strings are not sent to the host during bypass mode. However, it is possible to configure a disconnect notification to be sent during bypass mode. Please refer to the [abSerial Reference Guide](#) for more details.

2.2 System Configuration

2.2.1 Hardware

To connect the host to the abSerial interface, an RS-232 port is used. This cable will provide the connections illustrated below. The default port speed of the application is 115200 bps, with 8 data bits, no parity, and 1 stop bit. Serial port baud rates up to 921600 bps are supported. Hardware flow control (RTS/CTS) may be used by the abSerial interface. If CTS is de-asserted, the application will stop its transmission within one byte. When the abSerial interface de-asserts RTS, by default it can accept up to 22 more bytes (i.e., the host is expected to stop transmitting within 22 bytes).

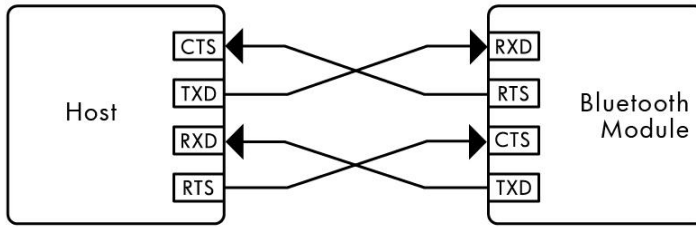


Figure 1. Connection to Host Device

2.2.2 Software

The abSerial interface executes on the device connected physically to the local host. The remote device must at minimum be a device that supports the Serial Port Profile (SPP).

If deep sleep is enabled, RTS will be de-asserted (CTS on the local host side) while the Bluetooth radio is in deep sleep.

2.3 System Data Flow

The following diagram shows how the data stream into and out of the abSerial interface is a part of an overall system that uses the Bluetooth SPP.

The bytes into and out of the application are of two types. The first is for commands and responses. Commands and responses are handled only while the application is in command mode. When in bypass mode, the second type of data stream is transferred directly to/from the UART and the Bluetooth SPP.

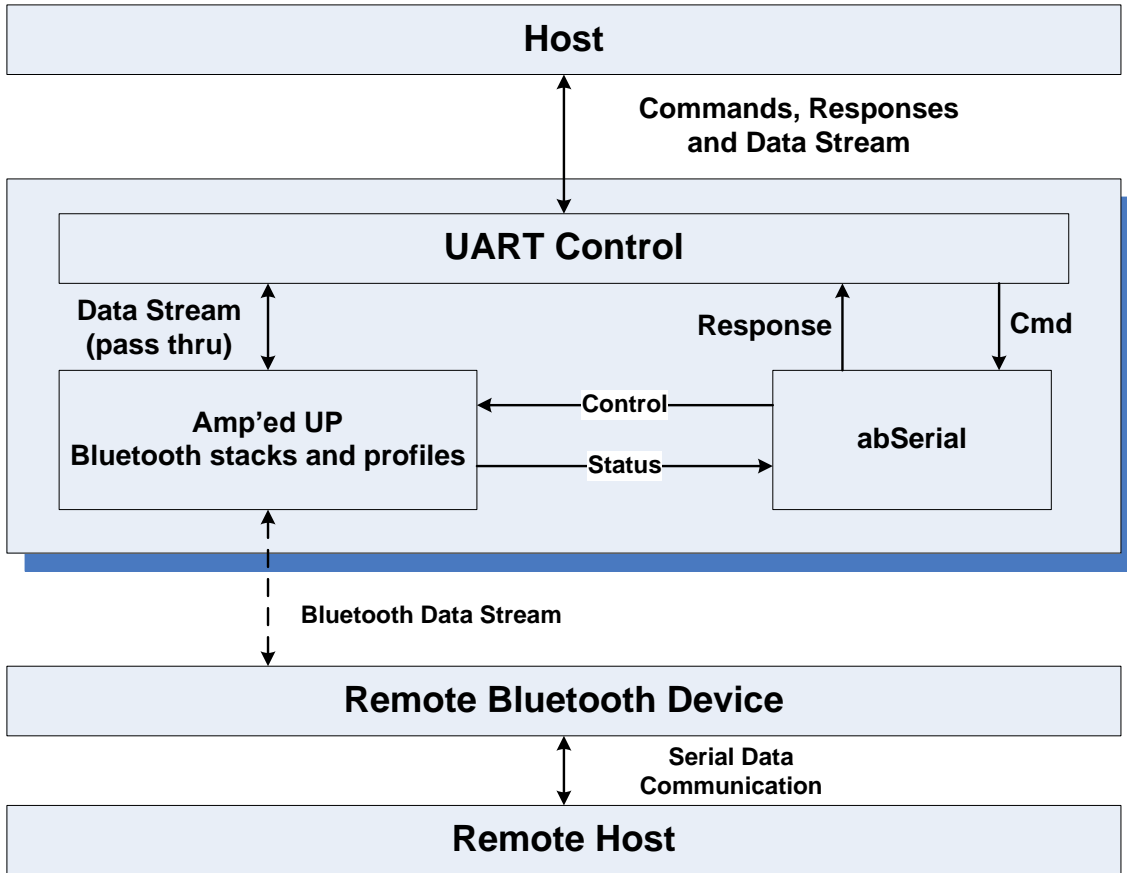


Figure 2. Data Flow between abSerial, Host, and Remote Bluetooth Device

2.4 Commands and Responses

A summary of all commands and responses can be found in the abSerial Reference Guide.

3 Startup

Upon initialization (due to power up or system reset), the abSerial interface starts in command mode and the serial port speed is set to its default baud rate. The application will then send two event strings (if host event strings are enabled). One notifies the host that the abSerial interface is in command mode and the other lists the BD address of the local device.

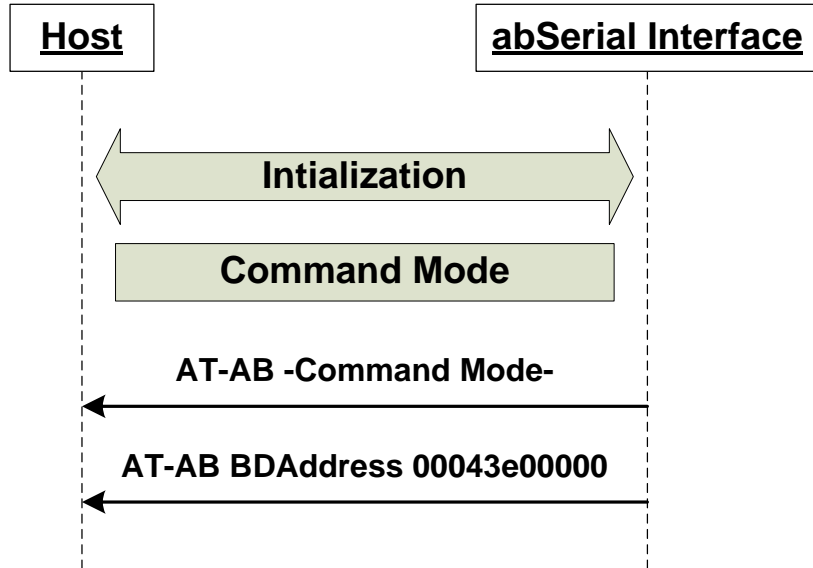


Figure 3. Event Strings Sent on Startup

Then, the application listens for a connection on the SPP profile and remains in command mode until a connection is established.

4 Configuration

The configuration of the abSerial interface allows the user to tailor the interface for a specific environment. This chapter addresses configuration through the use of AT commands; please refer to the abSerial Reference Guide for a description of these commands.

Configurations may be applied following system startup of the abSerial interface. The following subsections will describe common ways of using the configuration in different situations.

4.1 Device configuration

A common startup configuration may consist of setting the local name and/or the security mode.

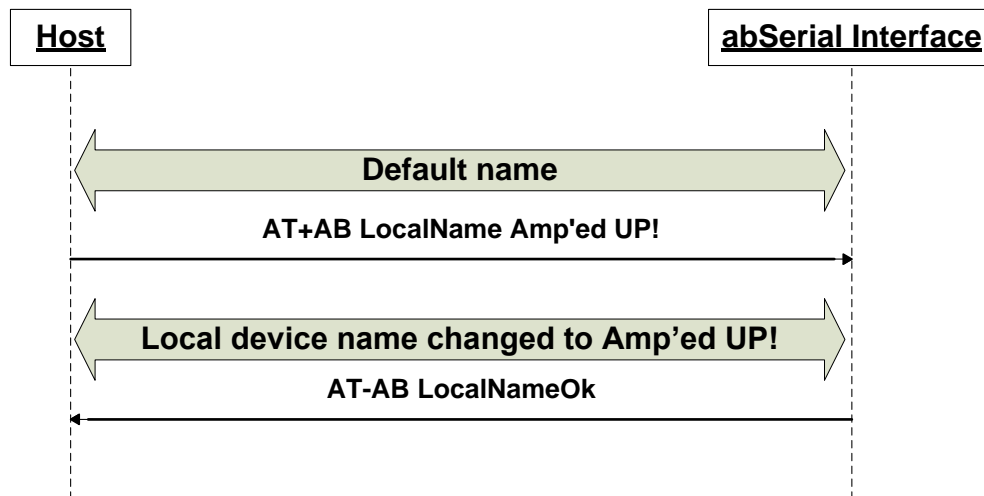


Figure 5. Command and Event Strings for Slave Device Configuration

4.2 abSerial Interface-to-Host Baud Rate

The host and abSerial interface can communicate at a number of baud rates. It may be necessary for the host to temporarily change the baud rate from the baud rate used at power up (the default speed is 115200 bps). The host can change the baud rate by changing the serial port speed while in command mode. The new port speed becomes effective once the response string has been transmitted to the host. This baud rate change remains in effect until the next system reset or power down. Alternatively, a new default baud rate may be configured, which will take effect after the next system reset. The configured baud rate does not directly affect the transfer of data over the Bluetooth connection.

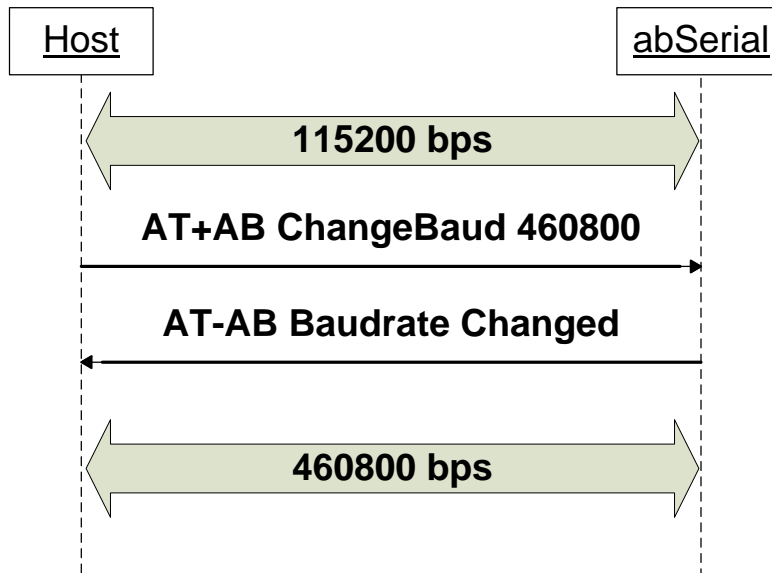


Figure 4. Command and Event Strings for Baud Rate Configuration

5 Connect with Remote Device

In order to create a connection with a remote device, the host issues a command string to the abSerial interface. This connection command may only be sent while in the command mode and when there is no active connection. The BD address of the remote device must be known at the time the connection is requested. Once the connection is established, the application goes into bypass mode.

5.1 Successful Connection

If the connection request is successful, the application will go to the bypass mode. The response can take a few seconds.

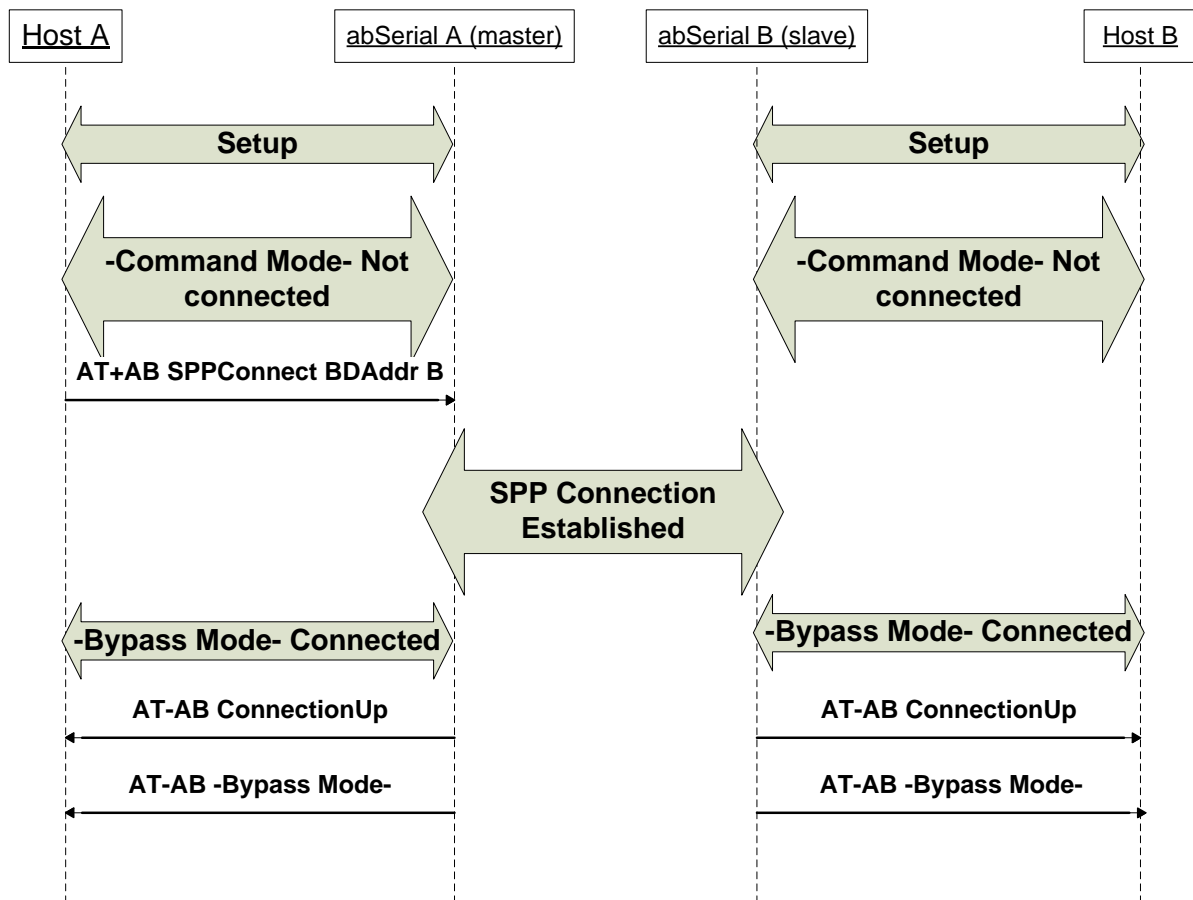


Figure 5. Command and Event Strings for Successful Connection

5.2 Unsuccessful Connection

There are numerous reasons a connection may not be established including remote device rejection due to security or poor RF quality. If the connection request is not successful, the application will remain in command mode. A response can take a few seconds.

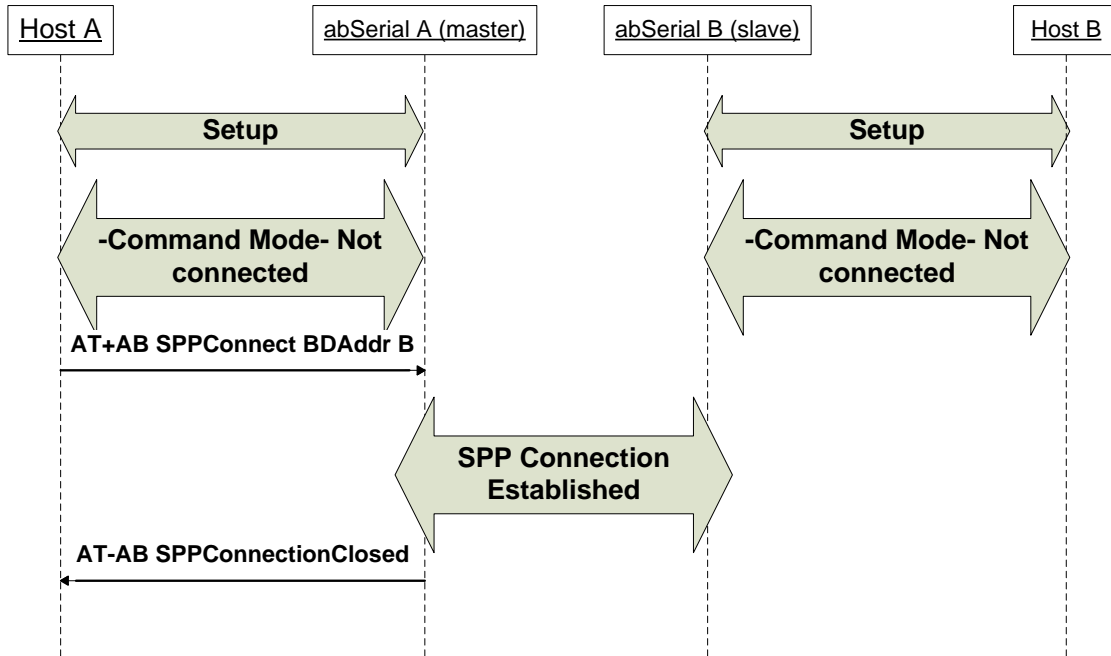


Figure 6. Command and Event Strings for Unsuccessful Connection

6 Disconnect with Remote Device

Once a connection is made, either device may request a disconnect. Also, a disconnect may occur unexpectedly due to changing conditions such as moving a device beyond the reception range. This section will illustrate disconnects under different situations.

In order to terminate an existing connection with a remote device, the host issues a disconnect command string. However, once a connection to a remote device has been established, the application is in bypass mode. Therefore, the host must first put the application in command mode before it can close the connection. The Escape sequence is sent to begin this process. The Escape sequence is discussed in greater detail in Section 7.

Once the abSerial interface is back in command mode, the host sends the Disconnect command string. The application notifies the host when the connection is broken and returns to command mode.

For disconnects initiated due to changing RF conditions, both hosts would receive the same notification as the non-initiating host in the following examples.

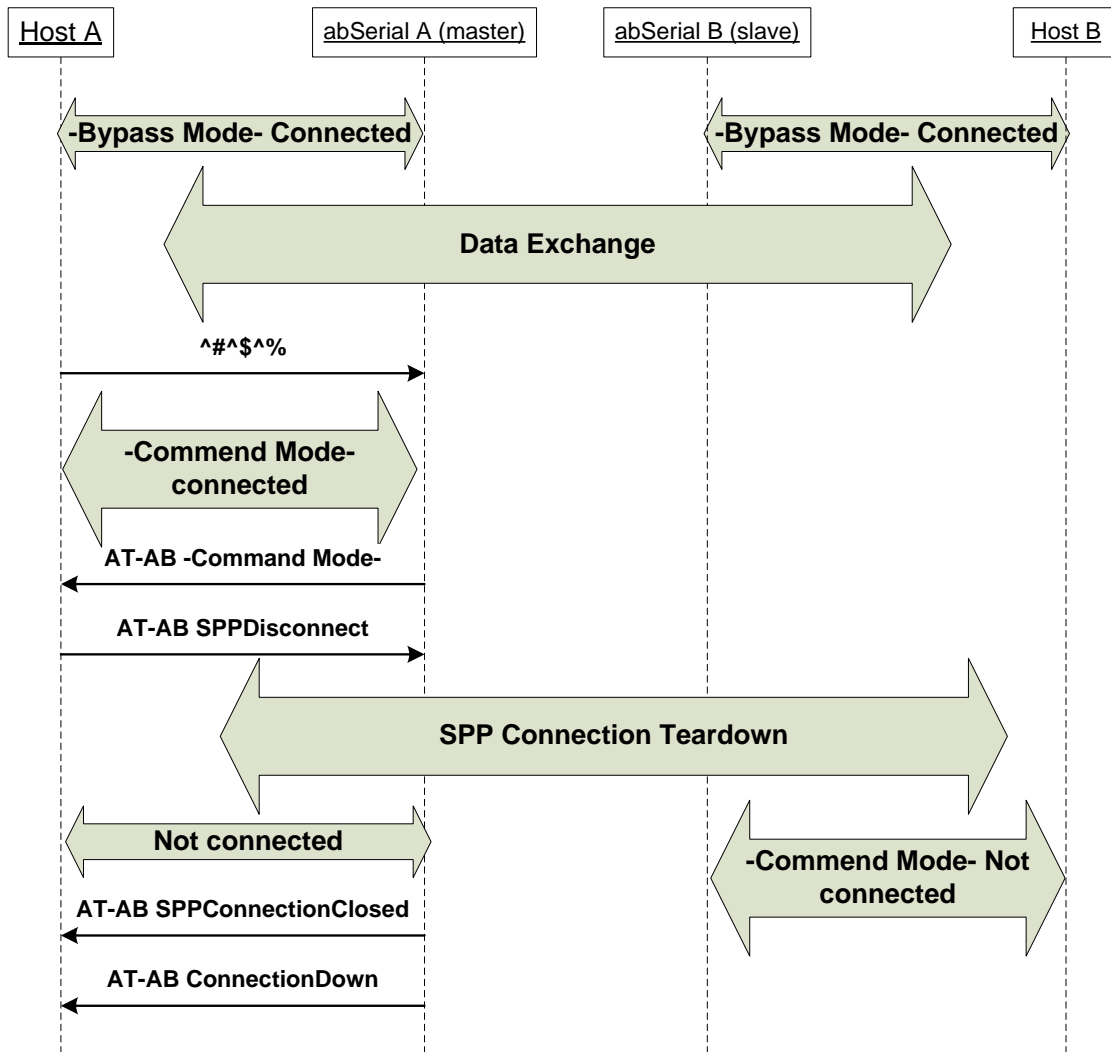


Figure 7. Escape, Command, and Event Strings for Notification Disabled

7 Escape from Bypass Mode

Once a connection has been established between host and remote device, the host can put the abSerial interface back into command mode. Once back in command mode, new commands (including the termination of a connection) can be issued. To change the application out of bypass mode and into the command mode, the Escape sequence is used.

The Escape sequence is an escape string followed by 1000 ms of no data. The Bluetooth connection to a remote device is not affected.



Note: any data received from the remote device while in command mode will be discarded by the local abSerial interface and not passed to the local host.

7.1 Connection Still Active

If the abSerial interface is in bypass mode when the escape string is sent (i.e., the connection is still active), the host must wait 2 seconds before the application will respond in the command mode.

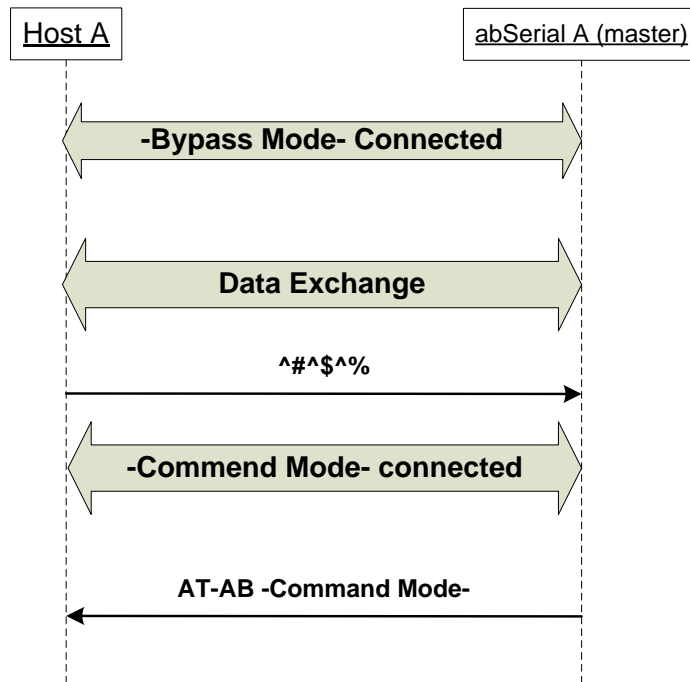


Figure 8. Escape Sequence and Event String when Connection Is Active



Note: the escape string must NOT be followed by a line-feed or carriage return.

8 Device Discovery

A abSerial device can use its discovery command to search for nearby Bluetooth devices. The information parameters returned by the feature to the host are the BD address and remote device name. The abSerial interface can also filter responses to show only particular classes of devices or service profiles. For more information on how to use the discovery command, please refer to the [abSerial Reference Guide](#).

On issuing the discovery command, the local device first does inquiry and displays the number of remote devices which responded to the inquiry procedure. The local device then performs a name request procedure on all of the remote devices found in during inquiry, in the order in which they responded. The name request procedure consists of establishing a connection, performing the name request, and then disconnecting.

Once name discovery and service discovery are performed on one device, the same procedure can be repeated for all devices that responded to the global inquiry.

The following example shows a general device discovery that returns two devices.

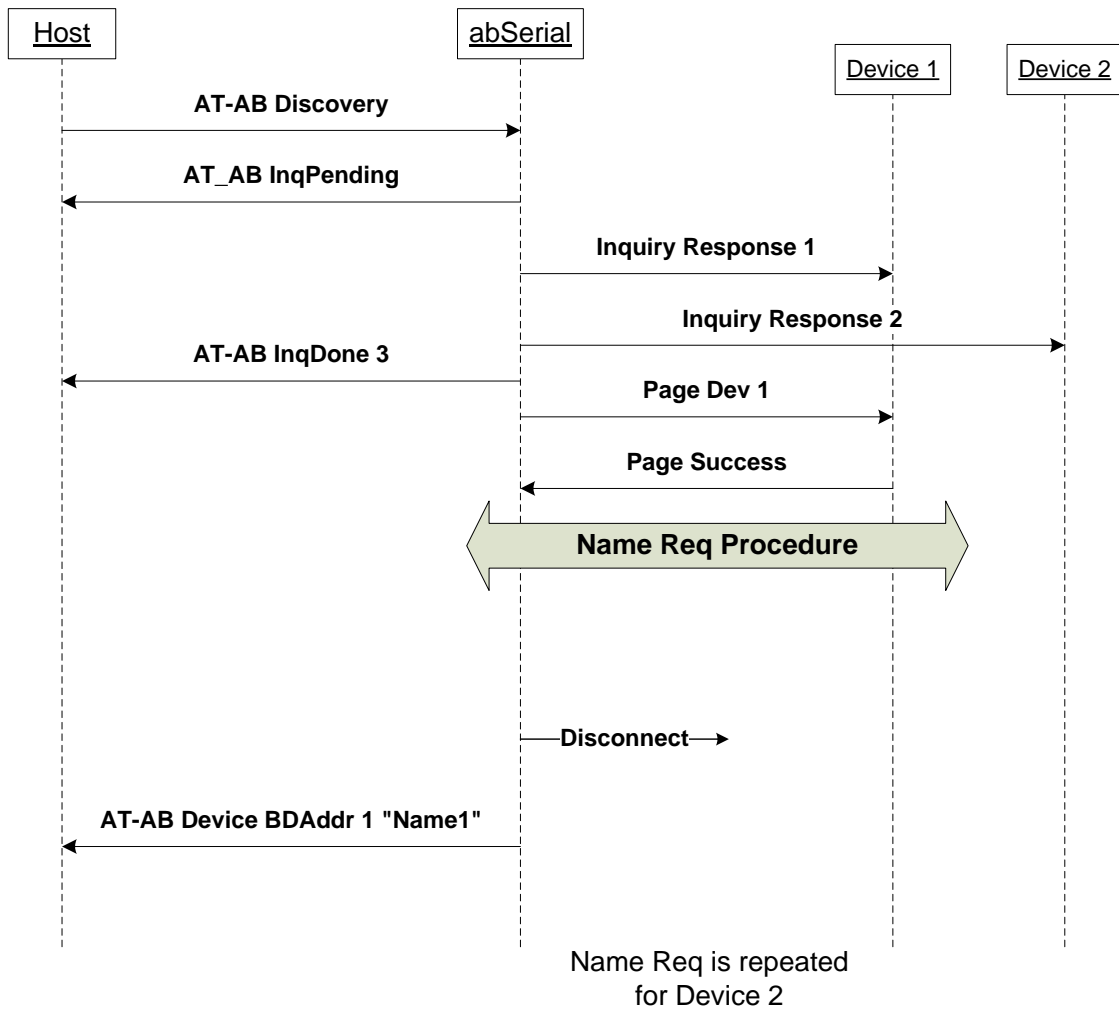


Figure 9. Commands and Events for Device Discovery

9 Bonding and Security

Bonding is used when an application needs to pair with another remote device. When the application issues “EnableBond”, it allows the local device to accept bonding requests. When the application issues “Bond”, it causes the local device to initiate bonding with the device to which it connects. Of course, if the application issues “Disable Bond” it forbids the local device from accepting any bonding requests, thus not allowing connections from devices that require bonding.

If the application issues “EnableBond” and is accepting a connection from a particular device (DeviceA) for the first time, it will:

Initiate authentication with the remote device (also known as Pairing)

Both devices will ask their respective hosts for a PIN which they then send to the other device for verification

If both devices confirm their PIN or PassKey codes, bonding succeeds and an encryption LinkKey is generated.

Store the new LinkKey for Device A.

If one of the devices does not verify the PIN, bonding fails and the connection is terminated.

Upon successive connections to DeviceA, the abSerial device will automatically use the stored LinkKey to authenticate the remote device and initiate encryption without notifying the abSerial host.

The following example shows how the abSerial interface can be used for bonding.

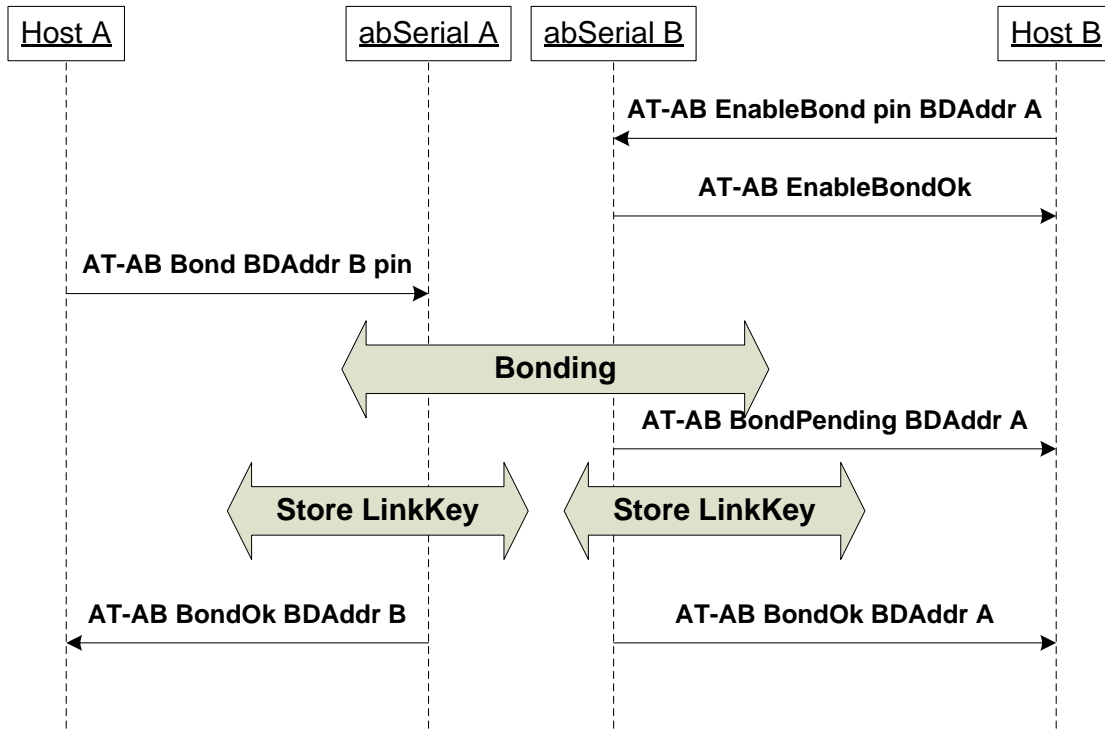


Figure 10. Command and Event Strings for Successful Bonding

If bonding is not successful, then `AT-AB BondFail` is sent to both hosts instead of `AT-AB BondOk`.

For further details on “Bond”, “EnableBond”, please refer to the [abSerial Reference Guide](#).

10 Smart Cable

This feature provides a simple, user-friendly, cable replacement application, the Smart Cable. An initial configuration from either the dynamic configuration file or AT command interface is used to set up the Smart Cable parameters. Once this is done, no further user intervention is necessary during normal usage. This section details this feature's initial setup and usage.

10.1 Configuration Parameters

The following parameters are supported:

- The BD Address of the remote auto-connect device.
- Reconnect attempt interval - 100ms to 100 seconds in 100ms increments.
- Reconnect attempts – 0 to 999. A value of 1,000 signifies unlimited attempts.

10.2 Usage

- Automatically establishes a SPP link to its designated remote device.
- The designated device is paged and retried up to the retry attempt limit setting, if it is unable to connect initially.
- If a link is disconnected, the Smart Cable feature will automatically re-connect the link without user interaction.
- A wait interval is inserted between automatic page attempts.

The Smart Cable setup AT command automatically updates its non-volatile memory parameters. These new settings are loaded after the next reset. The Delete Smart Cable AT command deletes these parameter settings in NVM and deactivates the Smart Cable feature for the remainder of the session.

11 Remote Escape Sequence, “@#@\$\$@%”

The purpose of this feature is to allow a remote abSerial device be controlled and configured by a Bluetooth link using a local host and Bluetooth device.

NOTE: This feature is disabled by default.

11.1 Usage

The Remote Escape Sequence feature utilizes an existing SPP profile link. To enable remote control, use the follow configuration command:

```
AT+AB Config RmtEscapeSequence = true
```

```
AT+AB Reset
```

A SPP connection must be established first. Once connected, user must send the Remote Escape Sequence “@#@\$\$@%” to the remote device, which will respond locally with “AT-AB RemoteMode. Then, it will accept and respond (both locally and over the link) to all standard AT commands send remotely.

- The Remote Escape Sequence “@#@\$\$@%” must be send in a single complete packet over the BT link.
- There should NOT be a CR or LF in this sequence.

12 Power Saving Mode

abSerial devices support various features, which allow low power operation over a range of scenarios. This section will discuss the Deep Sleep Mode, Sniff, and Auto Sniff features and how they may be effectively used.

NOTE: This feature is disabled by default.

12.1 Deep Sleep Mode

In abSerial, the basis for low power operation is Deep Sleep Mode, DSM. This feature temporarily halt's the chip's operation by stopping the main crystal and switching to the low power 32KHz oscillator instead. When enabled, DSM automatically enters this halt state whenever possible. Scheduled CPU activity, timers, remote link activity, and GPIO wakeup will automatically resume active mode operation.

12.1.1 UART Usage with Deep Sleep

When a UART is connected, the CTS line on the device's UART connector must not be asserted in order to allow DSM. The host device design must consider this when DSM is desired.

12.1.2 Deep Sleep Wakeup

abSerial supports a Deep Sleep Wakeup feature using a GPIO. When enabled, an active signal on the GPIO will temporarily prevent or block DSM. Normal DSM operation will resume when this signal is no longer asserted.

12.1.3 Flow Control with Deep Sleep

RTS/CTS flow control should be enabled with DSM. When no hardware flow control is supported, the Rx UART on the abSerial device cannot receive data or halt its transmission while in deep sleep. This scenario will lose all of the data sent to the abSerial device when DSM is active.

12.2 Sniff

abSerial supports sniff mode using an AT command, see the [abSerial Reference Guide](#) for details. When a connection is placed into sniff mode on a deep sleep enabled device, it will enter deep sleep during the inactive intervals between sniff polls. Since UART data cannot be received or transmitted when in deep sleep, communication will be blocked during the sniff intervals. This may not be acceptable for many applications, so an application controlled Auto Sniff mode is supported.

12.3 Auto Sniff Mode

This feature dynamically enables and disables sniff mode depending on a link's communication needs. Two configuration parameters control this feature: `Sniff Interval` and `Inactivity Timeout`. The `Sniff Interval` is the number of sniff poll interval slots that the sniff mode uses. The `Inactivity Timeout` is the number of seconds that the link will stay active after data is received or transmitted.

13 GPIO Usage

Up to 16 GPIO pins are supported on Amp'ed RF Bluetooth devices. Some abSerial features require these pins when they are enabled; see table below. Also, AT commands can be used to control these pins.

NOTE: For BT11 and BT31 modules. Other products may have fewer GPIO pins.

13.1 abSerial Application GPIO PIN Assignments

This table gives a summary of the abSerial interface's assignment of certain GPIO pins. Only the GPIO pins directly used by the abSerial interface are considered in this table; for complete GPIO assignments see the applicable device Data Sheet.

GPIO	abSerial Usage
1	CPU/Deep Sleep activity, output
4	Link Connection, output

13.2 GPIO AT Commands

abSerial AT commands can configure, read, and write to any of the 16 GPIO pins. If a pin is already in use by one of the above features, using an AT command to modify the pin will cause a conflict; this is not recommended. Refer to the [abSerial Reference Guide](#) for GPIO AT command details.

NOTE: For BT11 and BT31 modules. Other products may have fewer GPIO pins.

14 Data Throughput

Bluetooth sends 3 basic types of packets: 5, 3, and 1 slots. A 1 slot packet may be sent up to 800 per sec, or 1.25ms. A five slot packet (up to 322 bytes), may be sent at 266 per second.

There is a packet timeout (abSerial specific, not Bluetooth) of 44 bits, based on the baud rate. So, 115K baud, gives a timeout of about 400us. If no more data comes into the UART, the packet is sent even if it is not full.

If the maximum packet size, 322 bytes with Basic rate and up to 1012 bytes with EDR, is reached, a packet is sent without further delay.

The abSerial protocol does not extra bytes, but Bluetooth requires 10 bytes per packet for overhead.

Therefore, sending only 1 byte per packet is extremely inefficient. High baud rates can be supported if the bytes are sent without gaps. Then, they will be sent in the same packet, using the available bandwidth efficiently.