Specification Volume 4

Specification of the Bluetooth System

Wireless connections made easy

Host Controller Interface

[Transport Layer]

Volume 04
Revision 1.2 or later
Issued: 01 January 2006

Including:

A:	UART	v1.1
B:	SD	v1.1
C:	USB	v1.0
D:	3-wire	v0.95

Revision History

Rev	Date	Comments
D05R00	May 13 2004	First revision of this volume with informational content.
D10R00	28 July 2005	Second draft ready for review.
D10R02	11 October 2005	Second draft ready for review.
V10r00	01 January 2006	Approved, Board of Directors

Contributors

Toru Aihara	IBM Corporation
Edgar Kerstan	IBM Corporation
Nathan Lee	IBM Corporation
Kris Fleming	Intel Corporation
Robert Hunter	Intel Corporation
Patrick Kane	Motorola, Inc.
Uwe Gondrum	Nokia Corporation
Thomas Müller	Nokia Corporation
Christian Zechlin	Nokia Corporation
Johannes Elg	Telefonaktiebolaget LM Ericsson
Sven Jerlhagen	Telefonaktiebolaget LM Ericsson
Christian Johansson	Telefonaktiebolaget LM Ericsson
Patrik Lundin	Telefonaktiebolaget LM Ericsson
Lars Novak	Telefonaktiebolaget LM Ericsson
Masahiro Tada	Toshiba Corporation
Steve Ross	Digianswer A/S
Chatschik Bisdikian	IBM Corporation
Les Cline	Intel Corporation
Brad Hosler	Intel Corporation
John Howard	Intel Corporation
Srikanth Kambhatla	Intel Corporation
Kosta Koeman	Intel Corporation
John McGrath	Intel Corporation
Patrik Lundin	Telefonaktiebolaget LM Ericsson
Leonard Ott	Socket Communications
Rebecca O’Dell	Signia Technologies
Tsuyoshi Okada	Matsushita Electric
Robin Heydon	CSR

Web Site

This specification can also be found on the official Bluetooth website: http://www.bluetooth.org

DISCLAIMER AND COPYRIGHT NOTICE

Use of the Specification by anyone who is not a member of Bluetooth SIG or a party to an Early Adopters Agreement (each such person or party, a "Member"), is prohibited. The legal rights and obligations of each Member are governed by their applicable Membership Agreement, Early Adopters Agreement or Promoters Agreement. No license, express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

Any use of the Specification not in compliance with the terms of the applicable Membership Agreement, Early Adopters Agreement or Promoters Agreement is prohibited and any such prohibited use may result in termination of the applicable Membership Agreement or Early Adopters Agreement and other liability permitted by the applicable agreement or by applicable law to Bluetooth SIG or any of its members for patent, copyright and/or trademark infringement.

THE SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, SATISFACTORY QUALITY, OR REASONABLE SKILL OR CARE, OR ANY WARRANTY ARISING OUT OF ANY COURSE OF DEALING, USAGE, TRADE PRACTICE, PROPOSAL, SPECIFICATION OR SAMPLE.

Each Member hereby acknowledges that products equipped with the Bluetooth technology ("Bluetooth products") may be subject to various regulatory controls under the laws and regulations of various governments worldwide. Such laws and regulatory controls may govern, among other things, the combination, operation, use, implementation and distribution of Bluetooth products. Examples of such laws and regulatory controls include, but are not limited to, airline regulatory controls, telecommunications regulations, technology transfer controls and health and safety regulations. Each Member is solely responsible for the compliance by their Bluetooth Products with any such laws and regulations and for obtaining any and all required authorizations, permits, or licenses for their Bluetooth products related to such regulations within the applicable jurisdictions. Each Member acknowledges that nothing in the Specification provides any information or assistance in connection with securing such compliance, authorizations or licenses. NOTHING IN THE SPECIFICATION CREATES ANY WARRANTIES, EITHER EXPRESS OR IMPLIED, REGARDING SUCH LAWS OR REGULATIONS.

ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHTS OR FOR NONCOMPLIANCE WITH LAWS, RELATING TO USE OF THE SPECIFICATION IS EXPRESSLY DISCLAIMED. BY USE OF THE SPECIFICATION, EACH MEMBER EXPRESSLY WAIVES ANY CLAIM AGAINST BLUETOOTH SIG AND ITS PROMOTER MEMBERS RELATED TO USE OF THE SPECIFICATION.

Bluetooth SIG reserve the right to adopt any changes or alterations to the Specification as it deems necessary or appropriate.

TABLE OF CONTENTS

Part A
UART TRANSPORT LAYER

Part B
SECURE DIGITAL (SD) TRANSPORT LAYER

Part C
USB TRANSPORT LAYER

Part D

THREE-WIRE UART TRANSPORT LAYER

Host Controller Interface [Transport Layer]
Part A

UART TRANSPORT LAYER

This document describes the UART transport layer (between the Host and the Host Controller). HCI command, event, and data packets flow through this layer, but the layer does not decode them.

Revision 1.1

CONTENTS

1 GENERAL

The objective of this HCI UART Transport Layer is to make it possible to use the Bluetooth HCI over a serial interface between two UARTs on the same PCB. The HCI UART Transport Layer assumes that the UART communication is free from line errors.

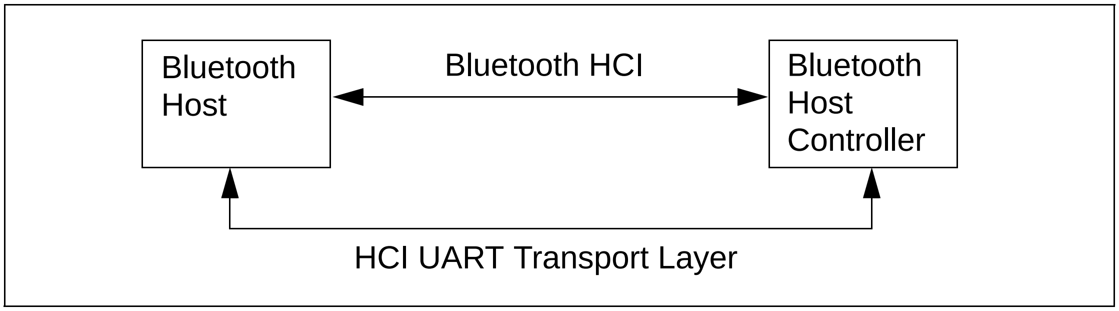

Figure 1.1: HCI UART Transport Layer

2 PROTOCOL

There are four kinds of HCI packets that can be sent via the UART Transport Layer; i.e. HCI Command Packet, HCI Event Packet, HCI ACL Data Packet and HCI SCO Data Packet (see "Host Controller Interface Functional Specification" in Volume 2, Part E). HCI Command Packets can only be sent to the Bluetooth Host Controller, HCI Event Packets can only be sent from the Bluetooth Host Controller, and HCI ACL/SCO Data Packets can be sent both to and from the Bluetooth Host Controller.

HCI does not provide the ability to differentiate the four HCI packet types. Therefore, if the HCI packets are sent via a common physical interface, a HCI packet indicator has to be added according to Table 2.1 below.

HCI packet type	HCI packet indicator
HCI Command Packet	0x01
HCI ACL Data Packet	0x02
HCI SCO Data Packet	0x03
HCI Event Packet	0x04

Table 2.1: HCI packet indicators

The HCI packet indicator shall be sent immediately before the HCI packet. All four kinds of HCI packets have a length field, which is used to determine how many bytes are expected for the HCI packet. When an entire HCI packet has been received, the next HCI packet indicator is expected for the next HCI packet. Over the UART Transport Layer, only HCI packet indicators followed by HCI packets are allowed.

3 RS232 SETTINGS

The HCI UART Transport Layer uses the following settings for RS232:

Baud rate:	**manufacturer-specific**
Number of data bits:	8
Parity bit:	no parity
Stop bit:	1 stop bit
Flow control:	RTS/CTS
Flow-off response time:	manufacturer specific

Table 3.1:

Flow control with RTS/CTS is used to prevent temporary UART buffer overrun. It should not be used for flow control of HCI, since HCI has its own flow control mechanisms for HCI commands, HCI events and HCI data.

If CTS is 1, then the Host/Host Controller is allowed to send.
If CTS is 0, then the Host/Host Controller is not allowed to send.

The flow-off response time defines the maximum time from setting RTS to 0 until the byte flow actually stops.

The RS232 signals should be connected in a null-modem fashion; i.e. the local TXD should be connected to the remote RXD and the local RTS should be connected to the remote CTS and vice versa.

4 ERROR RECOVERY

If the Host or the Host Controller lose synchronization in the communication over RS232, then a reset is needed. A loss of synchronization means that an incorrect HCI packet indicator has been detected, or that the length field in an HCI packet is out of range.

If the UART synchronization is lost in the communication from Host to Host Controller, then the Host Controller shall send a Hardware Error Event to tell the Host about the synchronization error. The Host Controller will then expect to receive an HCI_Reset command from the Host in order to perform a reset. The Host Controller will also use the HCI_Reset command in the byte stream from Host to Host Controller to re-synchronize.

If the UART synchronization is lost in the communication from Host Controller to Host, then the Host shall send the HCI_Reset command in order to reset the Host Controller. The Host shall then re-synchronize by looking for the HCI Command Complete event for the HCI_Reset command in the byte stream from Host Controller to Host.

See “Host Controller Interface Functional Specification” for HCI commands and HCI events in the Bluetooth Specification v1.2 or later.

Host Controller Interface [Transport Layer]
Part B

SECURE DIGITAL (SD) TRANSPORT LAYER

This document describes the SD transport layer (between the Host and Controller). HCI command, event and data packets flow through this layer, but the layer does not decode them. The Bluetooth SD Transport layer is defined in a document owned and maintained by the Secure Digital Association. Information regarding that document is described herein.

Revision 1.0

CONTENTS

1 INTRODUCTION

This document discusses the requirements of the Secure Digital (SD) interface for Bluetooth hardware. Readers should be familiar with SD, SD design issues, and the overall Bluetooth architecture. The reader should also be familiar with the Bluetooth Host Controller Interface.

The SD Bluetooth Protocol is documented in the SDIO Card Type-A Specification for Bluetooth, which is owned and maintained by the Secure Digital Association (SDA). The full specification is available to members of the SDA that have signed all appropriate SD NDA and license requirements. The SDA also makes a Non-NDA version available, the Simplified Version of: SDIO Card Type-A Specification for Bluetooth. There are no changes to the SDA document to comply with the requirements of the Bluetooth SIG.

2 GOALS

2.1 HARDWARE GOALS

The Bluetooth SD transport interface specification is designed to take advantage of both the SD Physical Transport bus and the packet orientation of the Bluetooth HCI protocol. Thus, all data is transferred in blocks as packets. Since the block size used on the SD bus may be smaller than the HCI packet, a segmentation and recombination protocol is defined.

2.2 SOFTWARE GOALS

The Bluetooth SD transport interface specification is designed for non-embedded solutions. It is assumed that the host software does not necessarily have a priori knowledge of the SD Bluetooth device.

The interface is not designed for embedded applications where much of the information passed via the interface is known in advance.

The SDA also defines a Bluetooth interface for embedded applications where the Controller contains protocol layers above HCI (RFComm, SDP etc.). This specification is called SDIO Card Type-B Specification for Bluetooth. Information about this specification can be obtained from the SDA (http://www.sdcard.org).

3 PHYSICAL INTERFACE DOCUMENTS

This specification references the SD SDIO Card Type-A Specification for Bluetooth. This SDA document defines the Bluetooth HCI for all SD devices that support an HCI level interface. Any SD Bluetooth device claiming compliance with the SD Bluetooth Transport must support this interface and additionally adhere to its device type specification, which is set by the Secure Digital Association. The SDIO Card Type-A Specification for Bluetooth document is based on the SDIO Card Specification, which in turn is based on the SD Memory Card Specification: Part 1 Physical Layer Specification. All of these documents are copyrighted by the SDA and are available ONLY to SDA member companies that have signed the appropriate NDA documents with the SDA. As an introduction to the SD Bluetooth Type A specification, the SDA has created 'Simplified' versions of each of these documents. The simplified versions do not contain enough information to fully implement a device, however they do contain enough information to convey the structure and intent of the specifications.

Applicable SDA Documents available to members of the SDA:

SD Memory Card Specification: Part 1 Physical Layer Specification

SDIO Card Specification

SDIO Card Type-A Specification for Bluetooth.

Applicable Simplified SDA Documents available to non-members and members of the SDA:

Simplified Version of: SD Memory Card Specification: Part 1 Physical Layer Specification

Simplified Version of: SDIO Card Specification:

Simplified Version of: SDIO Card Type-A Specification for Bluetooth

More information on the Secure Digital Association and the SD specifications can be found at the SDA website at. http://www.sdcard.org.

4 COMMUNICATION

4.1 OVERVIEW

Figure 4.1 below is a diagram of the communication interface between a Bluetooth SD device and the Bluetooth host protocol stack. Modifications to this diagram might be needed for operating systems that do not support a miniport model:

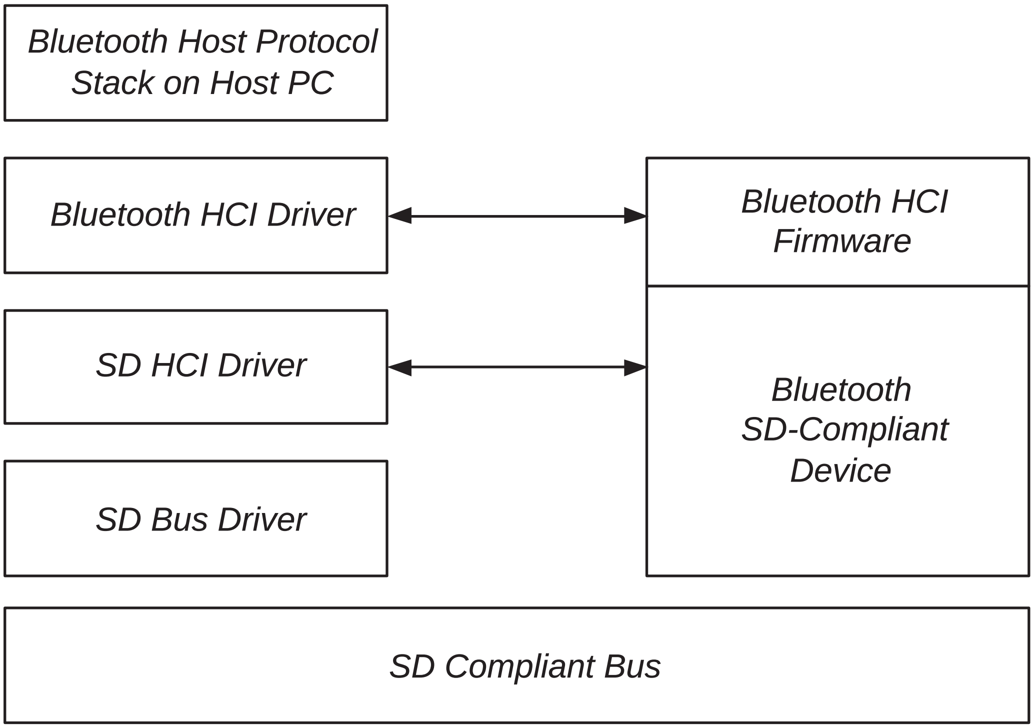

Figure 4.1: SD Communication Diagram

5 APPENDIX A - ACRONYMS AND ABBREVIATIONS

Acronym	Description
HCI	Host Controller Interface
NDA	Non-Disclosure Agreement
OS	Operating System
SD	Secure Digital
SDA	Secure Digital Association
SDIO	Secure Digital Input/Output
SDP	Service Discovery Protocol
SIG	Special Interest Group

Table 5.1: Acronyms and Abbreviations

6 APPENDIX B - RELATED DOCUMENTS

A) Bluetooth Core Specification v1.2 or later.

B) Applicable SDA Documents available to members of the SDA:

B.1) SD Memory Card Specification: Part 1 Physical Layer Specification

B.2) SDIO Card Specification

B.3) SDIO Card Type-A Specification for Bluetooth.

B.4) SDIO Card Type-B Specification for Bluetooth.

B.5) SDIO Card Physical Test Specification

B.5) SDIO Host Physical Test Specification

B.6) SD Bluetooth Type A Test Specification

These documents are available to members of the SDA in the "Members Only" section of the SDA web site (http://www.sdcard.org/access.htm). See http://www.sdcard.org/join.htm for information on joining the SDA.

C) Applicable Simplified SDA Documents available to non-members and members of the SDA:

C.1) Simplified Version of: SD Memory Card Specification: Part 1 Physical Layer Specification
http://www.sdcard.org/sdphysical_simplifyed_Ver101.pdf

C.2) Simplified Version of: SDIO Card Specification http://www.sdcard.org/SDIO-SimpleSpec-1.00_A.pdf

C.3) Simplified Version of: SDIO Card Type-A Specification for Bluetooth - http://www.sdcard.org/Simple_of_SDIO_Card_Type A_Spec_v1.0_Draft_C_20020524.pdf

7 APPENDIX C - TESTS

The SDA has defined formal test procedures for SDIO Type A Bluetooth cards (Controller) and Hosts. It is expected that both Controllers and Hosts will comply with all test requirements set forth by the SDA in accordance with the rules of the SDA. The Bluetooth SIG does not require any formal testing to comply with SIG requirements. The test document names are listed in Appendix B.

7.1 TEST SUITE STRUCTURE

There are two types of tests defined for the HCI SD Transport Layer:

1. Functional Tests
2. Protocol Tests

Tests of both types are defined for both the Host and Controller.

The purpose of the functional tests is to verify that the SD Bluetooth Type A Specification, SDIO Standard and SD Physical Standard have been implemented according to the specifications. These tests and the test environment for these tests are defined in documents provided by the SDA.

The purpose of the protocol tests are to verify that the Bluetooth Controller SD implementation or the Host implementation are according to the SD Bluetooth Type A specification.

The test environment for the protocol tests consists of the tester and the Device Under Test (DUT) as illustrated in Figure 7.1 below.

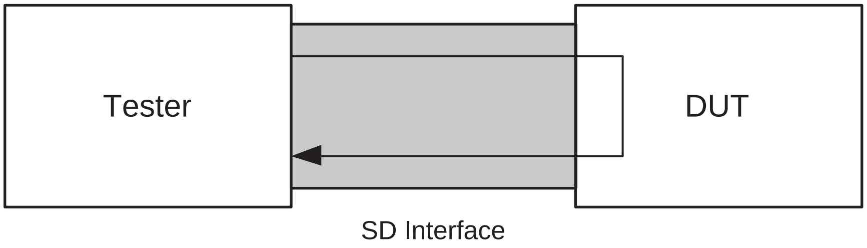

Figure 7.1: Protocol Test Environment

The tester is typically a PC with an SD interface. The DUT is placed into local loopback mode and standard HCI commands are used to drive the tests. The test results are verified in the tester.

Host Controller Interface [Transport Layer]
Part C

USB TRANSPORT LAYER

This document describes the USB transport layer (between a host and the host controller). HCI commands flow through this layer, but the layer does not decode the commands.

Revision 1.1

CONTENTS

1 OVERVIEW

This document discusses the requirements of the Universal Serial Bus (USB) interface for Bluetooth hardware. Readers should be familiar with USB, USB design issues, Advanced Configuration Power Interface (ACPI), the overall Bluetooth architecture, and the basics of the radio interface.

The reader should also be familiar with the Bluetooth Host Controller Interface.

Referring to Figure 1.1 below, notice that this document discusses the implementation details of the two-way arrow labelled ’USB Function’.

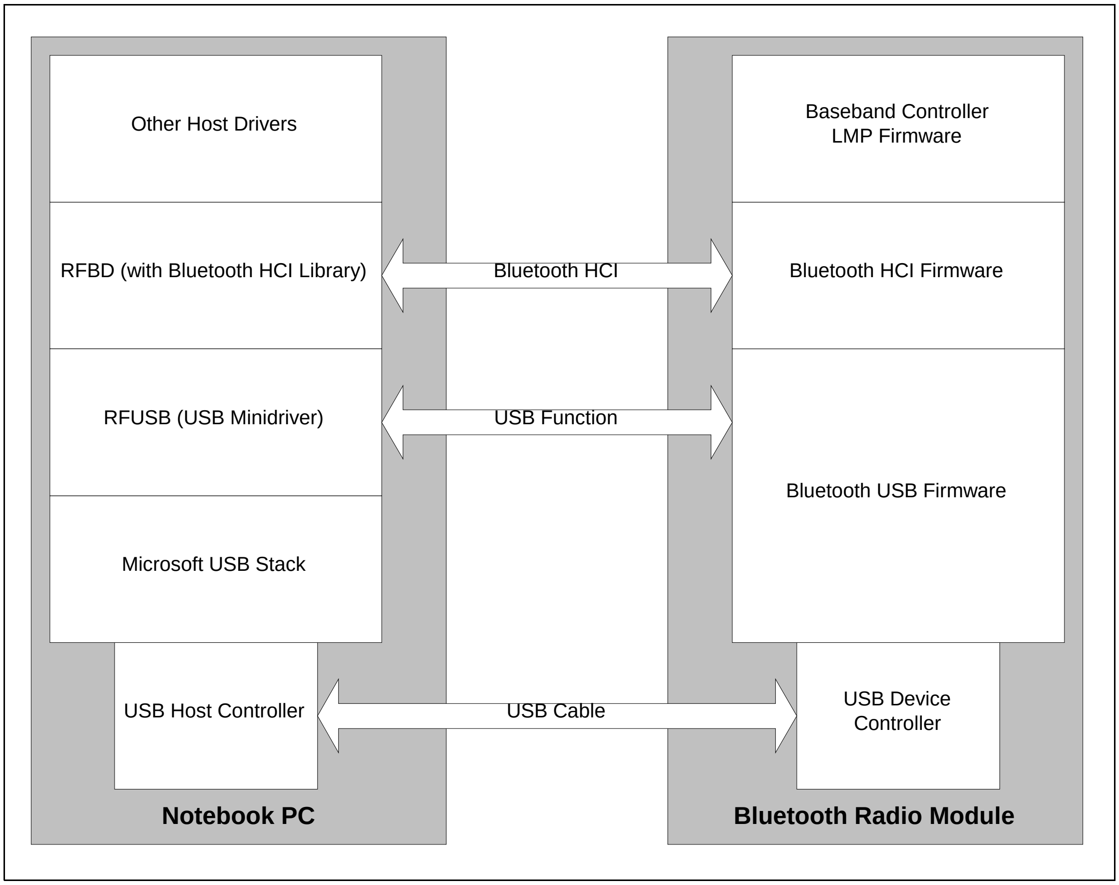

Figure 1.1: Relationship between the host and the Bluetooth Radio Module

The USB hardware can be embodied in one of two ways:

1. As a USB dongle, and
2. Integrated onto the motherboard of a notebook PC.

Finally, for an overview of the connection that is established between two Bluetooth devices, reference Figure 1.2, below.

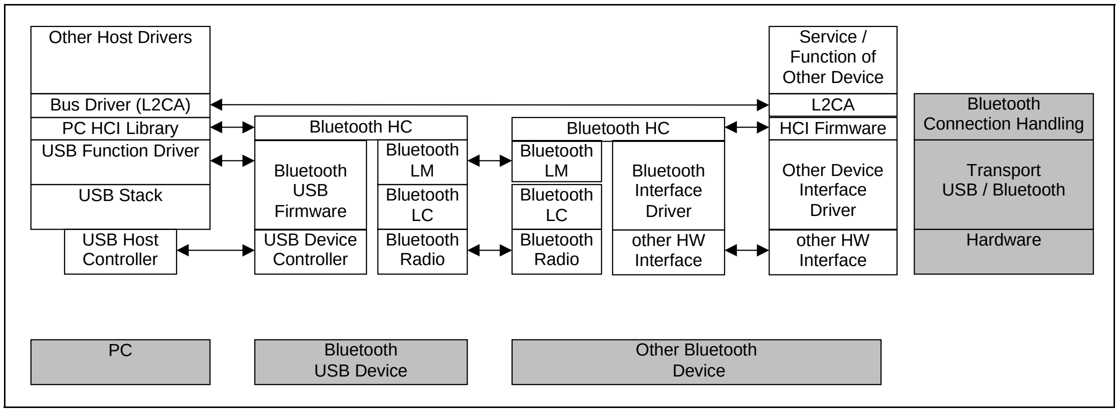

Figure 1.2: Flow of data from one Bluetooth device to another

2 USB ENDPOINT EXPECTATIONS

This section outlines specific USB endpoints that are required in order to function properly with the host. This section assumes a basic familiarity with USB. The endpoint numbers (labelled ’Suggested Endpoint Address’ below) may be dynamically recognized upon driver initialization – this depends on the implementation.

2.1 DESCRIPTOR OVERVIEW

The USB device is intended for high speed. The firmware configuration consists of two interfaces. The first interface (interface zero) has no alternate settings and contains the bulk and interrupt endpoints. The second interface (interface one) provides scalable isochronous bandwidth consumption. The second interface has four alternate settings that provide different consumption based on the required isochronous bandwidth. The default interface is empty so that the device is capable of scaling down to no isochronous bandwidth.

An HCI frame, consisting of an HCI header and HCI data, should be contained in one USB transaction. A USB transaction is defined as one or more USB frames that contain the data from one IO request. For example, an ACL data packet containing 256 bytes (both HCI header and HCI data) would be sent over the bulk endpoint in one IO request. That IO request will require four 64-byte USB frames, and forms a transaction.

The endpoints are spread across two interfaces so that when adjusting isochronous bandwidth consumption (via select interface calls), any pending bulk and/ or interrupt transactions do not have to be terminated and resubmitted.

The following table outlines the required configuration.

Interface Number	Alternate Setting	Suggested Endpoint Address	Endpoint Type	Suggested Max Packet Size
HCI Commands				
0	0	0x00	Control	8/16/32/64
HCI Events				
0	0	0x81	Interrupt (IN)	16
ACL Data				
0	0	0x82	Bulk (IN)	32/64
0	0	0x02	Bulk (OUT)	32/64
No active voice channels (for USB compliance)				
1	0	0x83	Isoch (IN)	0
1	0	0x03	Isoch (OUT)	0
One voice channel with 8-bit encoding				
1	1	0x83	Isoch (IN)	9
1	1	0x03	Isoch (OUT)	9
Two voice channels with 8-bit encoding & One voice channel with 16-bit encoding				
1	2	0x83	Isoch (IN)	17
1	2	0x03	Isoch (OUT)	17
Three voice channels with 8-bit encoding				
1	3	0x83	Isoch (IN)	25
1	3	0x03	Isoch (OUT)	25
Two voice channels with 16-bit encoding				
1	4	0x83	Isoch (IN)	33
1	4	0x03	Isoch (OUT)	33
Three voice channels with 16-bit encoding				
1	5	0x83	Isoch (IN)	49
1	5	0x03	Isoch (OUT)	49

Table 2.1: USB firmware interface and endpoint settings

The following two examples are used to demonstrate the flow of data given the describe endpoints.

Number of voice channels	Duration of voice data	Encoding
One	3 ms per IO Request	8-bit

Time (ms	USB data (header refers to HCI header) (Receive & Send from the host)	Queued data (read / write)	Time (ms)	Air data	Amount Received/ Sent (ms)
0	Receive 0 bytes Send 9 bytes (3 header, 6 data)	0 / 6	0	Send 0	0 / 0
		10 / 6	0.625	Receive 10	1.25 / 0
1	Receive 0 bytes Send 9 bytes (9 bytes HCI data)	10 / 15	1.25	Send 0	1.25 / 0
		20 / 15	1.875	Receive 10	2.50 / 0
2	Receive 0 bytes Send 9 bytes (9 bytes HCI data)	20 / 24	2.50	Send 0	2.50 / 0
		30 / 24	3.125	Receive 10	3.75 / 0
3	Receive 9 bytes (3 header, 6 data) Send 9 bytes (3 header, 6 data)	24 / 20	3.75	Send 10	3.75 / 1.25
4	Receive 9 bytes (9 bytes data) Send 9 bytes (9 bytes HCI data)	25 / 29	4.375	Receive 10	5.0 / 1.25
5	Receive 9 bytes (9 bytes data) Send 9 bytes (9 bytes HCI data)	16 / 28	5.0	Send 10	5.0 / 2.50
		26 / 28	5.625	Receive 10	6.25 / 2.50
6	Receive 9 bytes (3 header, 6 data) Send 9 bytes (3 header, 6 data)	20 / 24	6.25	Send 10	6.25 / 3.75
		30 / 24	6.875	Receive 10	7.5 / 3.75
7	Receive 9 bytes (9 bytes data) Send 9 bytes (9 bytes HCI data)	21 / 23	7.5	Send 10	7.5 / 5.0

Table 2.2: USB transactions

Time (ms	USB data (header refers to HCI header) (Receive & Send from the host)	Queued data (read / write)	Time (ms)	Air data	Amount Received/ Sent (ms)
8	Receive 9 bytes (9 bytes data) Send 9 bytes (9 bytes HCI data)	22 / 32	8.125	Receive 10	8.75 / 5.0
		22 / 22	8.75	Send 10	8.75 / 6.25
9	Receive 9 bytes (3 header, 6 data) Send 9 bytes (3 header, 6 data)	26 / 28	9.375	Receive 10	10.0 / 6.25
10	Receive 9 bytes (9 bytes data) Send 9 bytes (9 bytes HCI data)	17 / 27	10	Send 10	10.0 / 7.5
		27 / 27	10.625	Receive 10	11.25 / 7.5
11	Receive 9 bytes (9 bytes data) Send 9 bytes (9 bytes HCI data)	18 / 26	11.25	Send 10	11.25 / 8.75

Table 2.2: USB transactions

Convergence is expected because the radio is sending out an average of eight bytes of voice data every one ms and USB is sending eight bytes of voice data every one ms.

Number of voice channels	Duration of voice data	Encoding
Two	3 ms per IO Request	8-bit

Time (ms	USB data (header refers to HCI header) (Receive & Send from the host)	Queued data (read / write)	Time (ms)	Air data	Amount Received / Sent (ms)
0	Receive 0 bytes for Channel #1 Send 17 bytes (3 header, 14 data) for Channel #1	C1- 0/14 C2- 0/0	0	Send 0 for C1	C1- 0/0 C2- 0/0
		C1- 20/14 C2- 0/0	0.625	Receive 20 for C1	C1- 2.5/0 C2- 0/0

Table 2.3: Convergence of radio and USB data

Time (ms	USB data (header refers to HCI header) (Receive & Send from the host)	Queued data (read / write)	Time (ms)	Air data	Amount Received / Sent (ms)
1	Receive 0 bytes for Channel #1 Send 17 bytes (17 bytes HCI data) for Channel #1	C1- 20/31 C2- 0/0	1.25	Send 0 for C2	C1- 2.5/0 C2- 0/0
		C1- 20/31 C2- 20/0	1.875	Receive 20 for C2	C1- 2.5/0 C2- 2.5/0
2	Receive 0 bytes for Channel #1 Send 17 bytes (17 bytes HCI data) for Channel #1	C1- 20/28 C2- 20/0	2.50	Send 20 for C1	C1- 2.5/2.5 C2- 2.5/0
		C1- 40/28 C2- 0/0	3.125	Receive 20 for C1	C1- 5.0/2.5 C2- 2.5/0
3	Receive 0 bytes for Channel #2 Send 17 bytes (3 header, 14 data) for Channel #2	C1- 40/28 C2- 20/14	3.75	Send 0 for C2	C1- 5.0/2.5 C2- 2.5/0
4	Receive 0 bytes for Channel #2 Send 17 bytes (17 bytes HCI data) for Channel #2	C1- 40/28 C2- 40/31	4.375	Receive 20 for C2	C1- 5.0/2.5 C2- 5.0/0
5	Receive 0 bytes for Channel #2 Send 17 bytes (17 bytes HCI data) for Channel #2	C1- 40/8 C2- 40/48	5.0	Send 20 for C1	C1- 5.0/5.0 C2- 5.0/0
		C1- 60/8 C2- 40/48	5.625	Receive 20 for C1	C1- 7.5/5.0 C2- 5.0/0
6	Receive 17 bytes (3 header, 14 data) for Channel #1 Send 17 bytes (3 header, 14 data) for Channel #1	C1- 46/22 C2- 40/28	6.25	Send 20 for C2	C1- 7.5/5.0 C2- 5.0/2.5
		C1- 46/22 C2- 60/28	6.875	Receive 20 for C2	C1- 7.5/5.0 C2- 7.5/2.5
7	Receive 17 bytes (17 bytes data) for Channel #1 Send 17 bytes (17 bytes HCI data) for Channel #1	C1- 29/19 C2- 60/28	7.5	Send 20 for C1	C1- 7.5/7.5 C2- 7.5/2.5

Table 2.3: Convergence of radio and USB data

Time (ms	USB data (header refers to HCI header) (Receive & Send from the host)	Queued data (read / write)	Time (ms)	Air data	Amount Received / Sent (ms)
8	Receive 17 bytes (17 bytes data) for Channel #1 Send 17 bytes (17 bytes HCI data) for Channel #1	C1- 32/36 C2- 60/28	8.125	Receive 20 for C1	C1- 10/7.5 C2- 7.5/2.5
		C1- 32/36 C2- 60/8	8.75	Send 20 for C2	C1- 10/7.5 C2- 7.5/5.0
9	Receive 17 bytes (3 header, 14 data) for Channel #2 Send 17 bytes (3 header, 14 data) for Channel #2	C1- 32/36 C2- 54/22	9.375	Receive 20 for C2	C1- 10/7.5 C2- 10/5.0
10	Receive 17 bytes (17 bytes data) for Channel #2 Send 17 bytes (17 bytes HCI data) for Channel #2	C1- 32/16 C2- 37/39	10	Send 20 for C1	C1- 10/10 C2- 10/5.0
		C1- 52/16 C2- 37/39	10.625	Receive 20 for C1	C1- 12.5/10 C2- 10/5.0
11	Receive 17 bytes (17 bytes data) for Channel #2 Send 17 bytes (17 bytes HCI data) for Channel #2	C1- 52/16 C2- 20/36	11.25	Send 20 for C2	C1- 12.5/10 C2- 10/7.5

Table 2.3: Convergence of radio and USB data

2.2 CONTROL ENDPOINT EXPECTATIONS

Endpoint 0 is used to configure and control the USB device. Endpoint 0 will also be used to allow the host to send HCI-specific commands to the host controller. When the USB firmware receives a packet over this endpoint that has the Bluetooth class code, it should treat the packet as an HCI command packet.

2.3 BULK ENDPOINTS EXPECTATIONS

Data integrity is a critical aspect for ACL data. This, in combination with bandwidth requirements, is the reason for using a bulk endpoint. Multiple 64-byte packets can be shipped, per millisecond, across the bus.

Suggested bulk max packet size is 64 bytes. Bulk has the ability to transfer multiple 64-byte buffers per one millisecond frame, depending on available bus bandwidth.

Bulk has the ability to detect errors and correct them. Data flowing through this pipe might be destined for several different slaves. In order to avoid starvation, a flow control model similar to the shared endpoint model is recommended for the host controller.

2.4 INTERRUPT ENDPOINT EXPECTATIONS

An interrupt endpoint is necessary to ensure that events are delivered in a predictable and timely manner. Event packets can be sent across USB with a guaranteed latency.

The interrupt endpoint should have an interval of 1 ms.

The USB software and firmware requires no intimate knowledge of the events passed to the host controller.

2.5 ISOCHRONOUS ENDPOINTS EXPECTATIONS

These isochronous endpoints transfer SCO data to and from the host controller of the radio.

Time is the critical aspect for this type of data. The USB firmware should transfer the contents of the data to the host controllers' SCO FIFOs. If the FIFOs are full, the data should be overwritten with new data.

These endpoints have a one (1) ms interval, as required by Chapter 9 of the USB Specification, Versions 1.0 and 1.1.

The radio is capable of three (3) 64Kb/s voice channels (and can receive the data coded in different ways – 16-bit linear audio coding is the method that requires the most data). A suggested max packet size for this endpoint would be at least 64 bytes. (It is recommended that max packet sizes be on power of 2 boundaries for optimum throughput.) However, if it is not necessary to support three voice channels with 16-bit coding, 32 bytes could also be considered an acceptable max packet size.

3 CLASS CODE

A class code will be used that is specific to all USB Bluetooth devices. This will allow the proper driver stack to load, regardless of which vendor built the device. It also allows HCI commands to be differentiated from USB commands across the control endpoint.

The class code (bDeviceClass) is 0xE0 – Wireless Controller.

The SubClass code (bDeviceSubClass) is 0x01 – RF Controller.

The Protocol code (bDeviceProtocol) is 0x01 – Bluetooth programming.

4 DEVICE FIRMWARE UPGRADE

Firmware upgrade capability is not a required feature. But if implemented, the firmware upgrade shall be compliant with the "Universal Serial Bus Device Class Specification for Device Firmware Upgrade" (version 1.0 dated May 13, 1999) available on the USB Forum web site at http://www.usb.org.

5 LIMITATIONS

5.1 POWER SPECIFIC LIMITATIONS

Today, the host controller of USB-capable machines resides inside a chip known as PIIX4. Unfortunately, because of errata, the USB host controller will not receive power while the system is in S3 or S4. This means that a USB wake-up can only occur when the system is in S1 or S2.

Another issue with the USB host controller is that, while a device is attached, it continually snoops memory to see if there is any work that needs to be done. The frequency that it checks memory is 1ms. This prevents the processor from dropping into a low power state known as C3. Because the notebook processor is not able to enter the C3 state, significant power loss will occur. This is a real issue for business users – as a typical business user will spend almost 90% of their time in the C3 state.

5.2 OTHER LIMITATIONS

Data corruption may occur across isochronous endpoints. Endpoints one and two may suffer from data corruption.

USB provides 16-CRC on all data transfers. The USB has a bit error rate of 10^{-13}.

Note that when a dongle is removed from the system, the radio will lose power (assuming this is a bus-powered device). This means that devices will lose connection.

Host Controller Interface [Transport Layer]

Part D

THREE-WIRE UART TRANSPORT LAYER

This document describes the Three-Wire UART transport layer (between the Host and Controller). HCI command, event and data packets flow through this layer, but the layer does not decode them.

Revision 0.95[1]

[1] This specification is v0.95 and though included in the Bluetooth SIG's qualification program, the specification should not be considered finalized until interoperable prototypes have been developed and the specification is adopted at the v1.0 level.

CONTENTS

1 GENERAL

The HCI Three-Wire UART Transport Layer makes it possible to use the Bluetooth HCI over a serial interface between two UARTs. The HCI Three-Wire UART Transport Layer assumes that the UART communication may have bit errors, overrun errors or burst errors. See also “UART Transport Layer” on page 11[vol. 4].

2 OVERVIEW

The HCI Three-Wire UART Transport Layer is a connection based protocol that transports HCI commands, events, ACL and Synchronous packets between the Bluetooth Host and the Bluetooth Controller. Packet construction is in done in two steps. First, it adds a packet header onto the front of every HCI Packet which describes the payload. Second, it frames the packets using a SLIP protocol. Finally, it sends this packet over the UART interface.

The SLIP layer converts an unreliable octet stream into an unreliable packet stream. The SLIP layer places start and end octets around the packet. It then changes all occurrences of the frame start or end octet in the packet to an escaped version.

The packet header describes the contents of the packet, and if this packet needs to be reliably transferred, a way of identifying the packet uniquely, allowing for retransmission of erroneous packets.

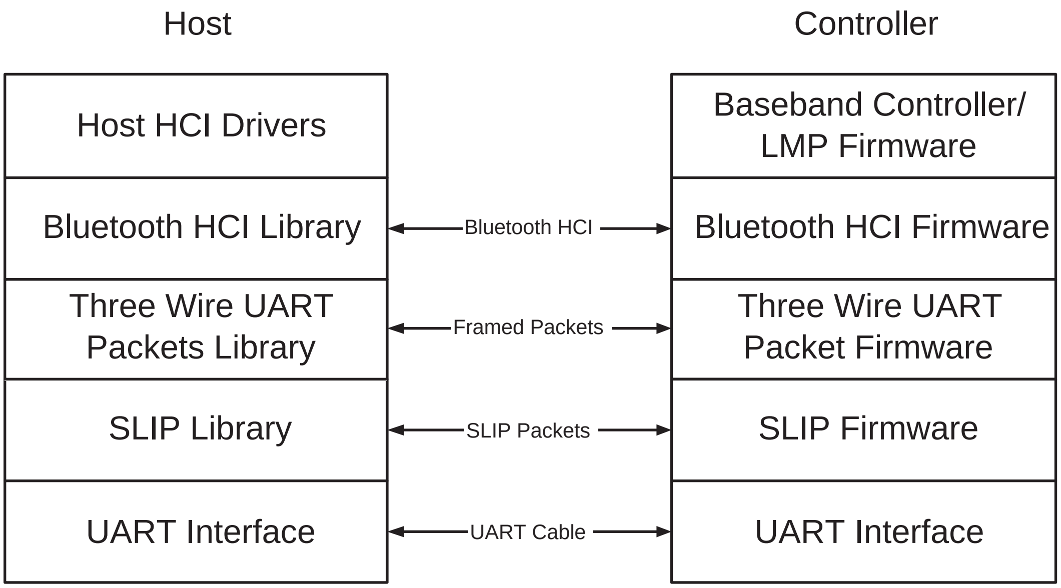

Figure 2.1: The Relationship Between the Bluetooth Host and the Bluetooth Controller

3 SLIP LAYER

The SLIP layer places packet framing octets around each packet being transmitted over the Three-Wire UART Transport Layer. This delimits the packets and allows packet boundaries to be detected if the receiver loses synchronization. The SLIP layer is based upon the RFC 1055 Standard [1].

3.1 ENCODING A PACKET

The SLIP layer performs octet stuffing on the octets entering the layer so that specific octet codes which may occur in the original data do not occur in the resultant stream.

The SLIP layer places octet 0xC0 at the start and end of every packet it transmits. Any occurrence of 0xC0 in the original packet is changed to the sequence 0xDB 0xDC before being transmitted. Any occurrence of 0xDB in the original packet is changed to the sequence 0xDB 0xDD before being transmitted. These sequences, 0xDB 0xDC and 0xDB 0xDD are SLIP escape sequences. All SLIP escape sequences start with 0xDB. All SLIP escape sequences are listed in Table 3.1.

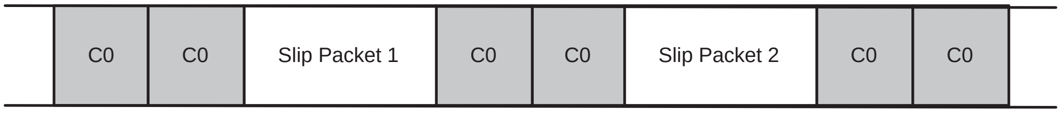

Figure 3.1: SLIP Packets with 0xC0 at the Start and End of Each Packet

3.2 DECODING A PACKET

When decoding a SLIP stream, a device will first be in an unknown state, not knowing if it is at the start of a packet or in the middle of a packet. The device must therefore discard all octets until it finds a 0xC0. If the 0xC0 is followed immediately by a second 0xC0, then the device will discard the first 0xC0 as it was presumably the end of the last packet, and the second 0xC0 was the start of the next packet. The device will then be in the decoding packet state. It can then decode the octets directly changing any SLIP escape sequences back into their unencoded form. When the device decodes the 0xC0 at the end of the packet, it will calculate the length of the SLIP packet, and pass the packet data into the packet decoder. The device will then seek the next packet. If the device does not receive an 0xC0 for the start of the next packet, then all octets up to and including the next 0xC0 will be discarded.

SLIP Escape Sequence	Unencoded form	Notes
0xDB 0xDC	0xC0	
0xDB 0xDD	0xDB	
0xDB 0xDE	0x11	Only valid when OOF Software Flow Control is enabled
0xDB 0xDF	0x13	Only valid when OOF Software Flow Control is enabled

Table 3.1: SLIP Escape Sequences

4 PACKET HEADER

Every packet that is sent over the Three-Wire UART Transport Layer has a packet header. It also has an optional Data Integrity Check at the end of the payload. The Transport Layer does not support packet segmentation and reassembly. Each transport packet will contain at most one higher layer packet.

A packet consists of a Packet Header of 4 octets, a Payload of 0 to 4095 octets, and an optional Data Integrity Check of 2 octets. See Figure 4.1.

The Packet header consists of a Sequence Number of 3 bits, an Acknowledge Number of 3 bits, a Data Integrity Check Present bit, a Reliable Packet bit, a Packet Type of 4 bits, a Payload Length of 12 bits and an 8 bit Header Checksum. See Figure 4.2.

LSB 4 Octets	0-4095	2 MSB
Packet Header	Payload	Data Integrity Check

Figure 4.1: Packet Format

LSB 3 bits	3	1	1	4	12	8 MSB
Sequence Number	**Acknowledgement Number**	**Data Integrity Check Present**	**Reliable Packet**	**Packet Type**	**Payload Length**	**Header Checksum**

Figure 4.2: Packet Header Format

4.1 SEQUENCE NUMBER

For unreliable packets this field will be set to 0 on transmit and ignored on receive.

Each new reliable packet will be assigned a sequence number which will be equal to the sequence number of the previous reliable packet plus one modulo eight. A packet will use the same sequence number each time it is retransmitted.

4.2 ACKNOWLEDGE NUMBER

The acknowledge number must be set to the sequence number of the next reliable packet this device expects to receive. See Section 6.4.

4.3 DATA INTEGRITY CHECK PRESENT

If a 16 bit CCITT-CRC Data Integrity Check is appended to the end of the payload, this bit shall be set to 1.

4.4 RELIABLE PACKET

If this bit it set to 1, then this packet is reliable. This means that the sequence number field is valid, and the receiving end must acknowledge its receipt. If this bit is set to 0, then this packet is unreliable.

4.5 PACKET TYPE

There are four kinds of HCI packets that can be sent via the Three-Wire UART Transport Layer; these are HCI Command Packet, HCI Event Packet, HCI ACL Data Packet and HCI Synchronous Data Packet (see “Host Controller Interface Functional Specification” in the Bluetooth Core Specification v1.2 or later). HCI Command Packets can be sent only to the Bluetooth Controller, HCI Event Packets can be sent only from the Bluetooth Controller, and HCI ACL/ Synchronous Data Packets can be sent both to and from the Bluetooth Controller.

HCI packet coding does not provide the ability to differentiate the four HCI packet types. Therefore, the Packet Type field is used to distinguish the different packets. The acceptable values for this Packet Type field are given in Table 4.1.

HCI Packet Type	Packet Type
Acknowledgement Packets	0
HCI Command Packet	1

HCI Packet Type	Packet Type
HCI ACL Data Packet	2
HCI Synchronous Data Packet	3
HCI Event Packet	4
Reserve	5-13
Vendor Specific	14
Link Control Packet	15

Table 4.1: Three-Wire UART Packet Type

HCI Command Packets, HCI ACL Data Packets and HCI Event Packets are always sent as reliable packets. HCI Synchronous Data Packets are sent as unreliable packets unless HCI Synchronous Flow Control is enabled, in which case they are sent as reliable packets.

In addition to the four HCI packet types, other packet types are defined. One packet type is defined for pure Acknowledgement Packets, and one additional packet type is to support link control. One packet type is made available to vendors for their own use. All other Three-Wire UART Packet Types are reserved for future use.

4.6 PAYLOAD LENGTH

The payload length is the number of octets in the payload data. This does not include the length of the packet header, or the length of the optional data integrity check.

4.7 PACKET HEADER CHECKSUM

The packet header checksum validates the contents of the packet header against corruption. This is calculated by setting the Packet Header Checksum to a value such that the 2's complement sum modulo 256 of the four octets of the Packet Header including the Packet Header Checksum is 0xFF.

5 DATA INTEGRITY CHECK

The Data Integrity Check field is optional. It can be used to ensure that the packet is valid. The Data Integrity Check field is appended onto the end of the packet. Each octet of the Packet Header and Packet Payload is used to compute the Data Integrity Check.

5.1 16 BIT CCITT-CRC

The CRC is defined using the CRC-CCITT generator polynomial

$g(D) = D^{16} + D^{12} + D^5 + 1$

(see Figure 5.1)

The CRC shift register is filled with 1s before calculating the CRC for each packet. Octets are fed through the CRC generator least significant bit first.

The most significant parity octet is transmitted first (where the CRC shift register is viewed as shifting from the least significant bit towards the most significant bit). Therefore, the transmission order of the parity octets within the CRC shift register is as follows:

x[8] (first), x[9],..., x[15], x[0], x[1],..., x[7] (last)

where x[15] corresponds to the highest power CRC coefficient and x[0] corresponds to the lowest power coefficient.

The switch S shall be set in position 1 while the data is shifted in. After the last bit has entered the LFSR, the switch shall be set in position 2, and the registers contents shall be read out for transmission.

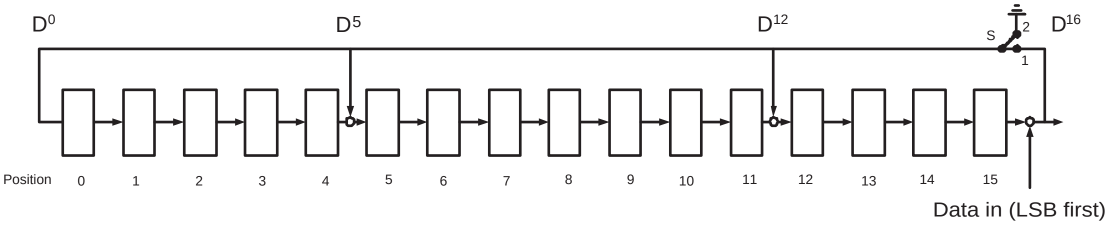

Figure 5.1: The LFSR Circuit Generating the CRC

6 RELIABLE PACKETS

To allow the reliable transmission of packets through the transport, a method needs to be defined to recover from packet errors. The Host or Controller can detect a number of different errors in the packet.

6.1 HEADER CHECKSUM ERROR

The header of the packet is protected by a Packet Header Checksum. If the 2's complement sum modulo 256 of the four octets of the header is not 0xFF, then the packet has an unrecoverable error and all information contained in the packet shall be discarded.

6.2 SLIP PAYLOAD LENGTH ERROR

The length of the SLIP packet shall be checked against the Packet Payload Length. If the Data Integrity Check Present bit is set to 1, then the SLIP packet length should be 6 + Packet Payload Length. If the Data Integrity Check Present bit is set to 0, then the SLIP packet length should be 4 + Packet Payload Length. If this check fails, then all information contained in the packet shall be discarded. The SLIP packet length is the length of the data received from the SLIP layer after the SLIP framing, and SLIP escape codes have been processed.

6.3 DATA INTEGRITY CHECK ERROR

The packet may have a Data Integrity Check at the end of the payload. This is controlled by the Data Integrity Check Present bit in the header. If this is set to 1, then the Data Integrity Check at the end of the payload is checked. If this is different from the value expected, then the packet shall be discarded. If the link is configured to not use data integrity checks, and a packet is received with the Data Integrity Check Present bit set to 1, then the packet shall be discarded.

6.4 OUT OF SEQUENCE PACKET ERROR

Each device keeps track of the sequence number it expects to receive next. This will be one more than the sequence number of the last successfully received reliable packet, modulo eight. If a reliable packet is received which has the expected sequence number, then this packet shall be accepted.

If a reliable packet is received which does not have the expected sequence number, then the packet shall be discarded.

6.5 ACKNOWLEDGEMENT

Whenever a reliable packet is received, an acknowledgement shall be generated.

If a packet is available to be sent, the Acknowledgement Number of that packet shall be updated to the latest expected sequence number.

If a requirement to send an acknowledgement value is pending, but there are no other packets available to be sent, the device can send a pure Acknowledgement Packet. This is an Unreliable Packet, with the Packet Type set to 0, Payload Length set to 0, and the Sequence Number set to 0. The Acknowledge Number must be set correctly.

The maximum number of reliable packets that can be sent without acknowledgement defines the sliding window size of the link. This is configured during link establishment. See Sections 8.6, 8.7 and 8.8.

6.6 RESENDING PACKETS

A Reliable Packet shall be resent until it is acknowledged. Devices should refrain from resending packets too quickly to avoid saturating the link with retransmits. See Section 12.1.2.

6.7 EXAMPLE RELIABLE PACKET FLOW

Figure 6.1 shows the transmission of reliable packets between two devices. Device A sends a packet with a Sequence Number of 6, and an Acknowledgement Number of 3. Device B receives this packet correctly, so needs to generate an acknowledgement. Device B then sends a packet with Sequence Number 3 with its Acknowledgement Number set to the next expected packet Sequence Number from Device A of 7.

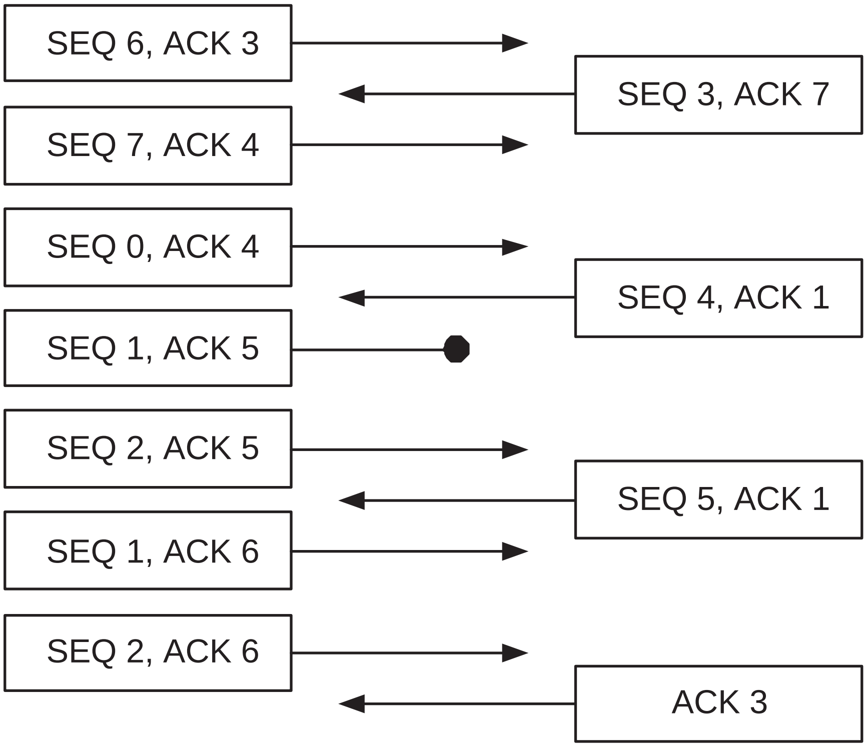

Figure 6.1: Message Diagram Showing Transmission of Reliable Packets

Device A receives a packet with Sequence Number 3 and an Acknowledgement Number of 7. Device A was expecting this sequence number so needs to generate an acknowledgement. The Acknowledgement Number of 7 is one greater than the last Sequence Number that was sent, meaning that this packet was received correctly (see Section 6.6).

Device A sends two packets, Sequence Numbers 7 and 0. Both packets have the Acknowledgement Number of 4, the next sequence number it expects from Device B. Device B receives the first correctly, and increments its next expected sequence number to 0. It then receives the second packet correctly, and increments the next expected sequence number to 1.

Device B sends a packet with Sequence Number 4, and the Acknowledgement Number of 1. This will acknowledge both of the previous two packets sent by Device A.

Device A now sends two more packets, Sequence Numbers 1 and 2. Unfortunately, the first packet is corrupted. Device B receives the first packet, and discovers the error, so discards this packet (see Section 6.1, Section 6.2 or Section 6.3). It must generate an acknowledgement of this erroneously received reliable packet. Device B then receives the second packet. This is received out of sequence, as it is currently expecting Sequence Number 1, but has received Sequence Number 2 (see 6.4). Again, it must generate an acknowledgement.

Device B sends another packet with Sequence Number 5. It is still expecting a packet with Sequence Number 1 next, so the Acknowledgement Number is set to 1. Device A receives this, and accepts this packet.

Device A has not had either of its last two packets acknowledged, so it must resend them (see 6.6). It does this, but must update the Acknowledgement Number of the original packets that were sent (see Section 6.5). The Sequence Numbers of these packets must stay the same (see Section 4.1).

Device B receives these packets correctly, and schedules the sending of an acknowledgement. Because Device B doesn’t have any data packets that need to be sent, it sends a pure Acknowledgement Packet (see Section 6.5).

7 UNRELIABLE PACKETS

To allow the transmission of unreliable packets through the transport, the following method shall be used.

7.1 UNRELIABLE PACKET HEADER

An unreliable packet header always has the Reliable Packet bit set to 0. The sequence number shall be set to 0. The Data Integrity Check Present, Acknowledgement Number, Packet Type, Payload Length and Packet Header Checksum shall all be set the same as a Reliable Packet.

7.2 UNRELIABLE PACKET ERROR

If a packet that is marked as unreliable and the packet has an error, then the packet shall be discarded.

8 LINK ESTABLISHMENT

Before any packets except Link Control Packets can be sent, the Link Establishment procedure must be performed. This ensures that the sequence numbers are initialized correctly, it also ensures that the two sides are using the same baud rate, allow detection of peer reset, and allows the device to be configured.

Link Establishment is defined by a state machine with three states: Uninitialized, Initialized and Active. When the transport is first started, the link is in the Uninitialized State. There are four messages that are defined: SYNC, SYNC RESPONSE, CONFIG and CONFIG RESPONSE. All four link establishment messages shall be sent with the Data Integrity Present flag set to 0.

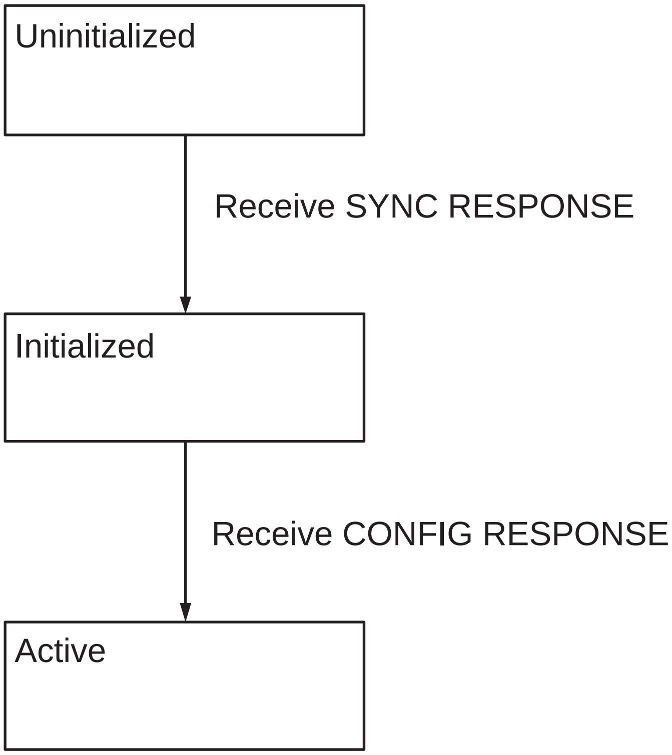

Figure 8.1: Link Establishment State Diagram

8.1 UNINITIALIZED STATE

In the Uninitialized State a device periodically[1] sends SYNC messages. If a SYNC message is received, the device shall respond with a SYNC RESPONSE message. If a SYNC RESPONSE message is received, the device shall move to the Initialized State. In the Initialized State only SYNC and SYNC RESPONSE messages are valid, all other messages that are received must be discarded. If an invalid packet is received, the device shall respond with a SYNC message. The device shall not send any acknowledgement packets in the Uninitialized State[2].

In the Uninitialized State the Controller may wait until it receives a SYNC message before sending its first SYNC message. This allows the Host to control when the Controller starts to send data.

The SYNC message can be used for automatic baud rate detection. It is assumed that the Controller shall stay on a single baud rate, while the Host could hunt for the baud rate. Upon receipt of a SYNC RESPONSE message, the Host can assume that the correct baud rate has been detected.

8.2 INITIALIZED STATE

In the Initialized State a device periodically sends CONFIG messages. If a SYNC message is received, the device shall respond with a SYNC RESPONSE message. If a CONFIG message is received, the device shall respond with a CONFIG RESPONSE message. If a CONFIG RESPONSE message is received, the device will move to the Active State. All other messages that are received must be ignored.

8.3 ACTIVE STATE

In the Active State, a device can transfer higher layer packets through the transport. If a CONFIG message is received, the device shall respond with a CONFIG RESPONSE message. If a CONFIG RESONSE message is received, the device shall discard this message.

[1] During link establishment, various messages are sent periodically. It is suggested to send 4 messages per second.

[2] Any packet that was erroneous would normally be acknowledged, as the recipient does not know if the packet was a reliable packet or not. The recipient cannot do this in the Uninitialized State, as it is possible to receive corrupt data while the Uninitialized state.

If a SYNC message is received while in the Active State, it is assumed that the peer device has reset. The local device should therefore perform a full reset of the upper stack, and start Link Establishment again at the Uninitialized State.

Upon entering the Active State, the first packet sent shall have its SEQ and ACK numbers set to zero.

8.4 SYNC MESSAGE

The SYNC message is an unreliable message sent with the Packet Type of 15 and a Payload Length of 2.

The payload is composed of the octet pattern 0x01 0x7E[1].

LSB	MSB
0x01	0x7E

Figure 8.2: Sync Message Format

8.5 SYNC RESPONSE MESSAGE

The SYNC RESPONSE message is an unreliable message sent with the Packet Type of 15 and a Payload Length of 2. The payload is composed of the octet pattern 0x02 0x7D.

LSB	MSB
0x02	0x7D

Figure 8.3: Sync Response Message Format

[1] The second octet for all Link Control Packets equals the least significant 7 bits of the first octet, inverted, with the most significant bit set to ensure even parity.

8.6 CONFIG MESSAGE

The CONFIG message is an unreliable message sent with the Packet Type of 15 and a Payload Length of 2 plus the size of the Configuration Field. The payload is composed of the octet pattern 0x03 0xFC and the Configuration Field.

LSB		MSB
0x03	0xFC	Configuration Field

Figure 8.4: Configuration Message Format

8.7 CONFIG RESPONSE MESSAGE

The CONFIG RESPONSE message is an unreliable message sent with the Packet Type of 15 and a Payload Length of 2 plus the size of the Configuration Field. The payload is composed of the octet pattern 0x04 0x7B and the Configuration Field.

LSB		MSB
0x04	0x7B	Configuration Field

Figure 8.5: Configuration Response Message Format

8.8 CONFIGURATION FIELD

The Configuration Field contains the Version Number, Sliding Window Size, the Data Integrity Check Type, and if Out Of Frame (OOF) Software Flow Control is allowed.

The Configuration Field in a CONFIG message sent by the Host determines what the Host can transmit and accept. The Configuration Field in a CONFIG RESPONSE message sent by the Controller determines what the Host and Controller shall transmit and can expect to receive.

The Controller sends CONFIG messages without a Configuration Field. The Host sends CONFIG RESPONSE messages without a Configuration Field.

LSB 3 Bits	1	1	3 MSB
Sliding Window Size	OOF Flow Control	Data Integrity Check Type	Version Number

Figure 8.6: Configuration Field Detail

To allow for future extension of the Configuration Field, the size of the message determines the number of significant Configuration Octets in the payload. Future versions of this specification may use extra octets. Any bits that are not included in the message shall be set to 0. Any bits that are not defined are reserved and shall be set to 0.

A device shall not change the values if sends in the Configuration Field during Link Establishment.

8.8.1 Configuration Messages

The CONFIG – CONFIG RESPONSE message sequence configures the link in both directions. Until a CONFIG RESPONSE message is received only unreliable Link Establishment messages may be sent. Once CONFIG RESPONSE message has been received all other packet types may be sent, and received messages passed up to the Host.

The CONFIG and CONFIG RESPONSE messages contain a set of options for both devices on the link. The Host sends a CONFIG message with the set of options that the Host would like to use. The Controller

responds with a CONFIG RESPONSE message with the set of options that the Host and the Controller will use. This means that the Controller is in full control of the set of options that will be used for all messages sent by both the Host and Controller.

8.8.2 Sliding Window Size

This is the maximum number of reliable packets a sender of the CONFIG message can send without requiring an acknowledgement. The value of this field shall be in the range one to seven. The value in the CONFIG RESPONSE message shall be less than or equal to the value in the CONFIG message. For example, the Host may suggest a window size of five in its CONFIG message and the Controller may respond with a value of three in its CONFIG

RESPONSE message, but not six or seven. Both devices will then use a maximum sliding window size of three.

8.8.3 Level of Data Integrity Check

The CONFIG message contains a bit field describing the types of Data Integrity Checks the sender is prepared to transmit. The peer will select the one it is prepared to use and send its choice in the CONFIG RESPONSE message.

If data integrity checks are not required, then the Data Integrity Check Present bit shall be set to 0 by the Host and Controller.

Level of Data Integrity	Parameter Description for CONFIG Message
0	No Data Integrity Check is supported.
1	16 bit CCITT-CRC may be used.

Figure 8.7: Data Integrity Check Type in the CONFIG Message

Level of Data Integrity	Parameter Description for CONFIG RESPONSE Message
0	No Data Integrity Check must be used.
1	16 bit CCITT-CRC may be used.

Figure 8.8: Data Integrity Check Type in the CONFIG RESPONSE Message

8.8.4 Out of Frame Software Flow Control

By default, the transport uses no flow control except that mandated by the HCI Functional Specification and the flow control achieved by not acknowledging reliable Host messages. If Software Flow Control is to be used, this needs to be negotiated.

The CONFIG message specifies whether the sender of the CONFIG message is prepared to receive Out of Frame Software Flow Control messages. The CONFIG RESPONSE message specifies whether the peer can send Out of Frame Software Flow Control messages. The CONFIG RESPONSE message may have the field set to 1 only if the CONFIG message had it set to 1. (See Section 10.1)

8.8.5 Version Number

The Version Number of this protocol shall determine which facilities are available to be used.

The CONFIG message specifies the Version Number supported by the Host. The CONFIG RESPONSE message specifies the Version Number that shall be used by the Host and Controller when sent by the Controller. The value in the CONFIG RESPONSE message shall be less than or equal to the value in the CONFIG message. The Version Numbers are enumerated in Figure 8.9. This specification is version 1.0 (Version Number = 0).

Version Number	Parameter Description for CONFIG and CONFIG RESPONSE Message
0	Version 1.0 of this Protocol
1-7	Reserved for future use

Figure 8.9: Version Number in the CONFIG and CONFIG RESPONSE message

9 LOW POWER

After a device is in the Active State, either side of the transport link may wish to enter a low power state. Because recovery from a loss of synchronization is possible, it is allowable to stop listening for incoming packets at any time.

To make the system more responsive after a device has entered a low power state, a system of messages is employed to allow either side to notify the other that they are entering a low power state and to wake a device from that state. These messages are sent as Link Control Packets. It is optional for a device to support the Sleep message. The Wakeup and Woken messages are mandatory.

9.1 WAKEUP MESSAGE

The Wakeup message shall be the first message sent whenever the device believes that the other side is asleep. The device shall then repeatedly send the Wakeup message until the Woken message is received. There must be at least a one character gap between the sending of each Wakeup message to allow the UART to resynchronize. The Wakeup message is an unreliable message sent with a Packet Type of 15, and a Payload Length of 2. The payload is composed of the octet pattern 0x05 0xFA. The Wakeup message shall be used after a device has sent a Sleep message. It is mandatory to respond to the Wakeup message.

LSB	MSB
0x05	0xFA

Figure 9.1: Wakeup Message Payload Format

9.2 WOKEN MESSAGE

The Woken message shall be sent whenever a Wakeup message is received even if the receiver is currently not asleep. Upon receiving a Woken message, a device can determine that the other device is not in a low power state and can send and receive data. The Woken message is an unreliable message sent with a Packet Type of 15, and a Payload Length of 2. The payload is composed of the octet pattern 0x06 0xF9.

LSB	MSB
0x06	0xF9

Figure 9.2: Woken Message Payload Format

9.3 SLEEP MESSAGE

A Sleep message can be sent at any time after Link Establishment has finished. It notifies the other side that this device is going into a low power state, and that it may also go to sleep. If a device sends a Sleep message it shall use the Wakeup / Woken message sequence before sending any data. If a device receives a Sleep message, then it should use the Wakeup / Woken message sequence before sending any data. The Sleep message is an unreliable message sent with a Packet Type of 15, and a Payload Length of 2. The payload is composed of the octet pattern 0x07 0x78.

The sending of this message is optional. The receiver of this message need not go to sleep, but cooperating devices may be able to schedule sleeping more effectively with this message.

LSB	MSB
0x07	0x78

Figure 9.3: Sleep Message Payload Format

10 OUT OF FRAME CONTROL

It is possible to embed information in the SLIP data stream after a SLIP ESCAPE character that can allow for Software Flow Control. This feature is optional and must be negotiated in the Link Establishment configuration messages.

10.1 SOFTWARE FLOW CONTROL

If Software Flow Control is enabled, then the standard XON / XOFF (0x11 and 0x13) characters will control the flow of data over the transport. To allow the XON / XOFF characters to be sent in the payload, they shall be escaped as follows: 0x11 shall be changed to 0xDB 0xDE, 0x13 shall be changed to 0xDB DF. This means that the XON / XOFF characters in the data stream are used only by software flow control.

If Software Flow Control is disabled, then the SLIP escape sequences 0xDB 0xDE and 0xDB 0xDF are undefined. In this case, the original octets of 0x11 and 0x13 shall not be changed. Flow control should always be provided by the tunneled protocols, e.g. HCI Flow Control. Flow control is still available using the standard Sequence Number / Acknowledge Number. This can be done by not acknowledging packets until traffic can resume.

11 HARDWARE CONFIGURATION

The HCI Three-Wire UART Transport uses the following configurations.

11.1 WIRES

There are three wires used by the HCI Three-Wire UART Transport. These are Transmit, Receive, and Ground.

11.1.1 Transmit & Receive

The transmit line from one device shall be connected to the receive line of the other device.

11.1.2 Ground

A common ground reference shall be used.

11.2 HARDWARE FLOW

Hardware flow control may be used. The signaling shall be the same as a standard RS232 flow control lines. If used, the signals shall be connected in a null-modem fashion; for example, the local RTS shall be connected to the remote CTS and vice versa.

11.2.1 RTS & CTS

Request to Send indicates to the remote side that the local device is able to accept more data.

Clear to Send indicates if the remote side is able to receive data.

(See ITU.T recommendations V.24 [2] and V.28 [3])

12 RECOMMENDED PARAMETERS

12.1 TIMING PARAMETERS

Because this transport protocol can be used with a wide variety of baud rates, it is not possible to specify a single timing value. However, it is possible to specify the time based on the baud rate in use. If T_{max} is defined as the maximum time in seconds it will take to transmit the largest packet over this transport, T_{max} can be expressed as:

T_{max} = maximum size of a packet in bits / baud rate

The maximum size of a packet in bits is either the number of bits in a 4095 octet packet (32,760) or less if required in an embedded system or as determined by the Host or Controller[1]. Thus, at a baud rate of 921,600 and the maximum packet size of 4095 octets, T_{max} is: (4095*10) / 921,600 = 44.434ms.

12.1.1 Acknowledgement of Packets

It is not necessary to acknowledge every packet with a pure acknowledgement packet if there is a data packet that will be sent soon. The recommended maximum time before starting to send an acknowledgement is 2 * T_{max}.

12.1.2 Resending Reliable Packets

A reliable packet must be resent until it is acknowledged. The recommended time between starting to send the same packet is 3 * T_{max}.

[1] This can be determined using the HCI_Read_Buffer_Size command.

13 REFERENCES

[1] [IETF RFC 1055: Nonstandard for transmission of IP datagrams over serial lines: SLIP – http://www.ietf.org/rfc/rfc1055.txt

[2] ITU Recommendation V.24: List of definitions for interchange circuits between data terminal equipment (DTE) and data circuit-terminating equipment (DCE) – http://www.itu.int/rec/recommendation.asp

[3] ITU Recommendation V.28: Electrical characteristics for unbalanced double-current interchange circuits – http://www.itu.int/rec/recommendation.asp