Apple Authentication Coprocessor Transport Layer

Application Programming Interface Reference Manual

Release: 4.0.1
January 10, 2013

Louisville, KY www.stonestreetone.com

Table Of Contents

1. Introduction

Bluetopia®, the Bluetooth Protocol Stack by Stonestreet One provides a software architecture that encapsulates the upper functionality of the Bluetooth Protocol Stack. More specifically, this stack is a software solution that resides above the Physical HCI (Host Controller Interface) Transport Layer and extends through the L2CAP (Logical Link Control and Adaptation Protocol) and the SCO (Synchronous Connection-Oriented) Link layers. In addition to basic functionality at these layers, the Bluetooth Protocol Stack by Stonestreet One provides implementations of the Service Discovery Protocol (SDP), RFCOMM (the Radio Frequency serial COMMunications port emulator), and several of the Bluetooth Profiles. Program access to these layers, services, and profiles is handled via Application Programming Interface (API) calls.

This document focuses on the API reference that contains a description of all programming interfaces for Stonestreet One's Apple Authentication Coprocessor (IACPTRAN) Transport Layer.

1.1 Scope

This reference manual provides information on the Apple Authentication Coprocessor Transport Layer API that is used by Bluetopia®. These APIs are used by Bluetopia® to physically communicate with an attached Apple Authentication Coprocessor. The implementation of these functions can be changed to allow the underlying transport mechanism for the Apple Authentication Coprocessor to change without impact on the Bluetopia® library. This mechanism allows the programmer the opportunity to change transport parameters as needed to support the transport required.

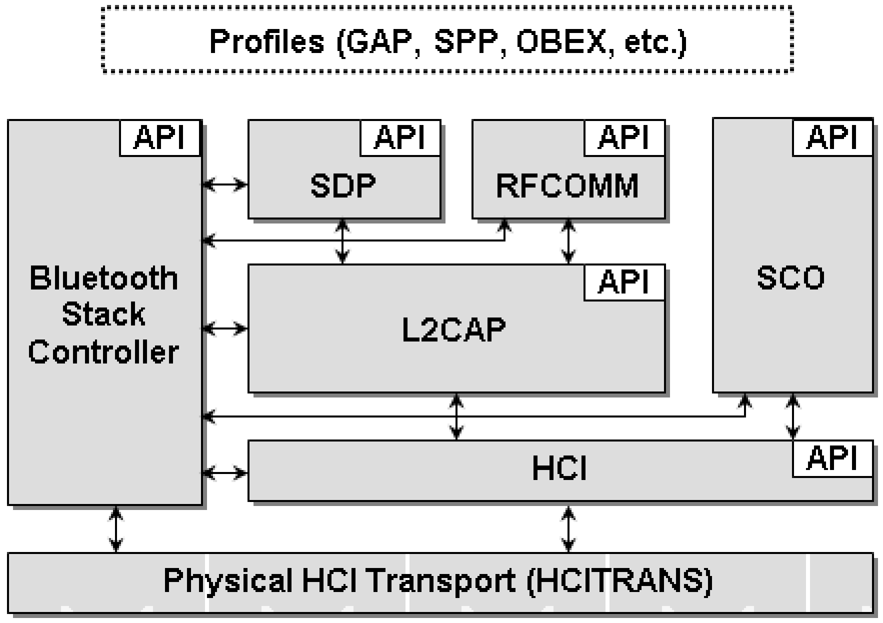
Profiles (GAP, SPP, OBEX, etc.)
API
Bluetooth Stack Controller
API
SDP
API
RFCOMM
API
SCO
API
L2CAP
HCI
API
Physical HCI Transport (HCITRANS)

Figure 1-1 The Stonestreet One Bluetooth Protocol Stack

1.2 Applicable Documents

The following documents may be used for additional background and technical depth regarding the Bluetooth technology.

1 *Specification of the Bluetooth System, Volume 0, Master Table of Contents & Compliance Requirements*, version 2.1+EDR, July 26, 2007.

2 *Specification of the Bluetooth System, Volume 1, Architecture and Terminology Overview*, version 2.1+EDR, July 26, 2007.

3 *Specification of the Bluetooth System, Volume 2, Core System Package [Controller Volume]*, version 2.1+EDR, July 26, 2007.

4 *Specification of the Bluetooth System, Volume 3, Core System Package [Host Volume]*, version 2.1+EDR, July 26, 2007.

5 *Specification of the Bluetooth System, Volume 4, Host Controller Interface*, version 2.1+EDR, July 26, 2007.

6 *Specification of the Bluetooth System, Bluetooth Core Specification Addendum 1*, June 26, 2008.

7 *Specification of the Bluetooth System, Volume 0, Master Table of Contents & Compliance Requirements*, version 3.0+HS, April 21, 2009.

8 *Specification of the Bluetooth System, Volume 1, Architecture and Terminology Overview*, version 3.0+HS, April 21, 2009.

9 *Specification of the Bluetooth System, Volume 2, Core System Package [Controller Volume]*, version 3.0+HS, April 21, 2009.

10 *Specification of the Bluetooth System, Volume 3, Core System Package [Host Volume]*, version 3.0+HS, April 21, 2009.

11 *Specification of the Bluetooth System, Volume 4, Host Controller Interface [Transport Layer]*, version 3.0+HS, April 21, 2009.

12 *Specification of the Bluetooth System, Volume 5, Core System Package [AMP Controller Volume]*, version 3.0+HS, April 21, 2009.

13 *Specification of the Bluetooth System, Volume 0, Master Table of Contents & Compliance Requirements*, version 4.0, June 30, 2010.

14 *Specification of the Bluetooth System, Volume 1, Architecture and Terminology Overview*, version 4.0, June 30, 2010.

15 *Specification of the Bluetooth System, Volume 2, Core System Package [BR/EDR Controller Volume]*, version 4.0, June 30, 2010.

16 *Specification of the Bluetooth System, Volume 3, Core System Package [Host Volume]*, version 4.0, June 30, 2010.

17 *Specification of the Bluetooth System, Volume 4, Host Controller Interface [Transport Layer]*, version 4.0, June 30, 2010.

18 *Specification of the Bluetooth System, Volume 5, Core System Package [AMP Controller Volume]*, version 4.0, June 30, 2010.

19 *Specification of the Bluetooth System, Volume 6, Core System Package [Low Energy Controller Volume]*, version 4.0, June 30, 2010.

20 *Bluetopia® Protocol Stack, System Call Requirements,* version 4.0, June 30, 2011

21 *Bluetopia® Protocol Stack, Application Programming Interface Reference Manual,* version 4.0, June 30, 2011.

1.3 Acronyms and Abbreviations

Acronyms and abbreviations used in this document and other Bluetooth specifications are listed in the table below.

Term	Meaning
API	Application Programming Interface
BD_ADDR	Bluetooth Device Address
BT	Bluetooth
HCI	Host Controller Interface
HS	High Speed
LE	Low Energy

Term	Meaning
LSB	Least Significant Bit
MSB	Most Significant Bit
ACP	Apple Authentication Coprocessor
IACP	Apple Authentication Coprocessor Transport

2. IACP Transport Layer API

The IACP Transport Layer programming interface defines the protocols and procedures to be used to implement the Apple Authentication Coprocessor transport layer. The IACP Transport Layer commands are listed in section 2.1. The actual prototypes and constants outlined in this section can be found in the **IACPTRAN.H** header file in the Bluetopia distribution.

2.1 IACP Transport Layer Commands

The available HCI Transport Layer over UART functions are listed in the table below and are described in the text that follows.

Function	Description
IACPTR_Initialize	Opens the IACP Transport Layer.
IACPTR_Cleanup	Closes the IACP Transport Layer and performs any necessary cleanup.
IACPTR_Read	Reads data from a specified register from the attached Authentication Coprocessor.
IACPTR_Write	Writes data into a specified register to the attached Authentication Coprocessor.
IACPTR_Reset	Reset the attached Authentication Coprocessor.

IACPTR_Initialize .

The following function is responsible for opening the IACP Transport layer that will be used by Bluetopia to communication with the Apple Authentication Coprocessor. This function must be successfully issued in order for IACP to function correctly. This function accepts as it’s only parameter a pointer to an opaque object that contains hardware specific initialization information. A successful call to this function will return zero if successful. This function returns a negative return value to signify an error.

Prototype:

int BTPSAPI **IACPTR_Initialize**(void *Parameters);

Parameters:

Parameters	Pointer to an opaque object that is used to pass hardware specific information to the module that is needed for proper operration. Each implementation can define what data is passed in this parameter.

Return:

Zero (0) if successful.

Negative value if an error occurred. Possible values are:

IACP_INVALID_PARAMETER
IACP_RESPONSE_TIMEOUT

IACPTR_Cleanup .

The following function is responsible for closing the IACP Transport layer that was previously opened. This function is used to close the IACP Transport layer and perform any necessary cleanup. After calling this function all IACP Transport layer commands will fail until another successful call to **IACPTR_Initialize** is made.

Prototype:

void BTPSAPI **IACPTR_Cleanup**(void);

Parameters:

None.

Return:

None.

IACPTR_Read .

The following function is responsible for actually reading data through the opened IACP Transport layer. Bluetopia® uses this function to read data from the attached Apple Authentication Coprocessor. This function **MUST NOT** return until it has read the number of requested bytes from the Apple Authentication Coprocessor (or an error condition occurs). Bluetopia® **WILL NOT** attempt to call this function repeatedly if the requested number of bytes is not returned. This function **MUST** block until it has read all of the requested data or an error condition occurs. This function returns zero if successful or a negative error if an error occurred. If this function returns zero (0) then the Status parameter will contain the actual status of the operation. For this function to be considered to have executed successfully it must return zero (0) **AND** return a successful status in the Status parameter (see the Status parameter below for more information).

Prototype:

int BTPSAPI **IACPTR_Read**(unsigned char Register, unsigned char BytesToRead,
    unsigned char *ReadBuffer, unsigned char *Status);

Parameters:

Register	The register in the Apple Authentication Coprocessor to read the data from. The possible values for this parameter are defined in **IACPType.h.** The values for this parameter will be in the following form: IACP_**XXX**_REGISTER
BytesToRead	The number of bytes that must be read from the Authentication Coprocessor.

ReadBuffer	Buffer to place the data that was read from the Authentication Coprocessor into. This buffer is guaranteed to be at least BytesToRead in length.
Status	This parameter is used to return the status of the read. If this function returns zero (0), the value that is returned in this parameter is used to determine the whether or not the read was successful. The following values may be returned in this parameter: IACP_STATUS_SUCCESS IACP_STATUS_READ_FAILURE IACP_STATUS_WRITE_FAILURE If successful this parameter will contain the following: IACP_STATUS_SUCCESS

Return:

Zero (0) if successful.

Negative value if an error occurred. Possible values are:

IACP_INVALID_PARAMETER
IACP_MODULE_NOT_INITIALIZED
IACP_RESPONSE_TIMEOUT

IACPTR_Write .

The following function is responsible for actually writing data through the opened IACP Transport layer. Bluetopia® uses this function to write data to the attached Apple Authentication Coprocessor. This function **MUST NOT** return until it has written the number of requested bytes to the Apple Authentication Coprocessor (or an error condition occurs). Bluetopia® **WILL NOT** attempt to call this function repeatedly if the requested number of bytes is not written to the attached Authentication Coprocessor. This function **MUST** block until it has written all of the requested data or an error condition occurs. This function returns zero if successful or a negative error if an error occurred. If this function returns zero (0) then the Status parameter will contain the actual status of the operation. For this function to be considered to have executed successfully it must return zero (0) **AND** return a successful status in the Status parameter (see the Status parameter below for more information).

Prototype:

int BTPSAPI **IACPTR_Write**(unsigned char Register, unsigned char BytesToWrite,
    unsigned char *WriteBuffer, unsigned char *Status);

Parameters:

Register	The register in the Apple Authentication Coprocessor to read the data from. The possible values for this parameter are defined in **IACPType.h.** The values for this parameter will be in the following form:

	IACP_**XXX**_REGISTER
BytesToWrite	The number of bytes that are to be written to the Authentication Coprocessor, pointed to by the second parameter, WriteBuffer.
WriteBuffer	Pointer to buffer that contains the data to be written.
Status	This parameter is used to return the status of the write. If this function returns zero (0), the value that is returned in this parameter is used to determine the whether or not the write was successful. The following values may be returned in this parameter: IACP_STATUS_SUCCESS IACP_STATUS_READ_FAILURE IACP_STATUS_WRITE_FAILURE If successful this parameter will contain the following: IACP_STATUS_SUCCESS

Return:

Zero (0) if successful.

Negative value if an error occurred. Possible values are:

IACP_INVALID_PARAMETER
IACP_MODULE_NOT_INITIALIZED
IACP_RESPONSE_TIMEOUT

IACPTR_Reset .

The following function is responsible for resetting the attached Apple Authentication Coprocessor. Bluetopia® uses this function to wake the ACP when it is sleeping. This function **MUST NOT** return until the reset process is complete. This function only applies to ACP versions 2.0B and older. For ACP versions 2.0C and newer, this function should be left empty, as these versions do not require manual power management.

Prototype:

void BTPSAPI **IACPTR_Reset**(void);

Parameters:

None.

Return:

None.

3. File Distributions

The source and header files required for the Apple Authentication CoprocessorTransport layer to be used with Bluetopia®, the Bluetooth Protocol Stack by Stonestreet One, are listed in the table below.

File	Contents/Description
IACPType.h	Bluetopia® Apple Authentication Coprocessor Transport types header.
IACPTRAN.h	Bluetopia® Apple Authentication Coprocessor Transport layer header module.
IACPTRAN.c	Bluetopia® Apple Authentication Coprocessor Transport layer source code module.

4. Authentication Copressor Transport Header File

```
/*****< iacptran.h >**********************************************************/
/*      Copyright 2001 - 2011 Stonestreet One.                                */
/*      All Rights Reserved.                                                  */
/*                                                                            */
/*  IACPTRAN - Apple Authentication Coprocessor Transport Type Definitions,   */
/*             Prototypes, and Constants.                                     */
/*                                                                            */
/*  Author:  Tim Thomas                                                       */
/*                                                                            */
/*** MODIFICATION HISTORY *****************************************************/
/*                                                                            */
/*   mm/dd/yy  F. Lastname    Description of Modification                     */
/*   --------  -----------    ------------------------------------------------*/
/*   07/08/11  T. Thomas      Initial creation.                               */
/******************************************************************************/
#ifndef __IACPTRANH__
#define __IACPTRANH__

#include "BTPSKRNL.h"      /* Bluetooth Kernel Prootypes/Constants.           */
#include "IACPType.h"      /* Types for the Authentication Coprocessor.       */

#define IACP_INVALID_PARAMETER               (-1)  /* Denotes an invalid      */
                                                   /* parameter was passed to */
                                                   /* a function.             */
#define IACP_MODULE_NOT_INITIALIZED          (-2)  /* Denotes that a function */
                                                   /* has been called before  */
                                                   /* the initialize function.*/
#define IACP_RESPONSE_TIMEOUT                (-3)  /* Denotes that a response */
                                                   /* was not received from   */
                                                   /* the chip in the allotted*/
                                                   /* amount of time.         */

   /* The following values are used with the Status parameter (return   */
   /* value) of the IACPTR_Read() and IACPTR_Write() functions.         */
   /* * NOTE * These are the ONLY values that can be be specified       */
   /*          for this return parameter.                               */
#define IACP_STATUS_SUCCESS                  (0)   /* Denotes a function      */
                                                   /* completed without error.*/
#define IACP_STATUS_READ_FAILURE             (1)   /* Denotes a Register Read */
                                                   /* failure.                */
#define IACP_STATUS_WRITE_FAILURE            (2)   /* Denotes a Register      */
                                                   /* Write failure.          */

   /* The following function is responsible for Initializing the ACP    */
   /* hardware.  The function is passed a pointer to an opaque object.  */
   /* Since the hardware platform is not know, it is intended that the  */
   /* parameter be used to pass hardware specific information to the    */
   /* module that would be needed for proper operation.  Each           */
   /* implementation will define what gets passed to this function.     */
   /* This function returns zero if successful or a negative value if   */
   /* the initialization fails.                                         */
int BTPSAPI IACPTR_Initialize(void *Parameters);

   /* The following function is responsible for performing any cleanup  */
   /* that may be required by the hardware or module.  This function    */
   /* returns no status.                                                */
void BTPSAPI IACPTR_Cleanup(void);

   /* The following function is responsible for Reading Data from a     */
   /* device via the ACP interface.  The first parameter to this        */
   /* function is the Register/Address that is to be read.  The second  */
   /* parameter indicates the number of bytes that are to be read from  */
   /* the device.  The third parameter is a pointer to a buffer where   */
   /* the data read is placed.  The size of the Read Buffer must be     */
   /* large enough to hold the number of bytes being read.  The last    */
   /* parameter is a pointer to a variable that will receive status     */
   /* information about the result of the read operation.  This function*/
   /* should return a non-negative return value if successful (and      */
   /* return IACP_STATUS_SUCCESS in the status parameter).  This        */
   /* function should return a negative return error code if there is an*/
   /* error with the parameters and/or module initialization.  Finally, */
   /* this function can also return success (non-negative return value) */
   /* as well as specifying a non-successful Status value (to denote    */
   /* that there was an actual error during the write process, but the  */
   /* module is initialized and configured correctly).                  */
   /* * NOTE * If this function returns a non-negative return value     */
   /*          then the caller will ALSO examine the value that was     */
   /*          placed in the Status parameter to deteriminе the actual  */
   /*          status of the operation (success or failure).  This      */
   /*          means that a successful return value will be a           */
   /*          non-negative return value and the status member will     */
   /*          contain the value:                                       */
   /*             IACP_STATUS_SUCCESS                                   */
int BTPSAPI IACPTR_Read(unsigned char Register, unsigned char BytesToRead, unsigned char
*ReadBuffer, unsigned char *Status);

   /* The following function is responsible for Writing Data to a device*/
   /* via the ACP interface.  The first parameter to this function is   */
   /* the Register/Address where the write operation is to start.  The  */
   /* second parameter indicates the number of bytes that are to be     */
   /* written to the device.  The third parameter is a pointer to a     */
   /* buffer where the data to be written is placed.  The last parameter*/
   /* is a pointer to a variable that will receive status information   */
   /* about the result of the write operation.  This function should    */
   /* return a non-negative return value if successful (and return      */
   /* IACP_STATUS_SUCCESS in the status parameter).  This function      */
   /* should return a negative return error code if there is an error   */
   /* with the parameters and/or module initialization.  Finally, this  */
   /* function can also return success (non-negative return value) as   */
   /* well as specifying a non-successful Status value (to denote that  */
   /* there was an actual error during the write process, but the       */
   /* module is initialized and configured correctly).                  */
   /* * NOTE * If this function returns a non-negative return value     */
   /*          then the caller will ALSO examine the value that was     */
   /*          placed in the Status parameter to deteriminе the actual  */
   /*          status of the operation (success or failure).  This      */
   /*          means that a successful return value will be a           */
   /*          non-negative return value and the status member will     */
   /*          contain the value:                                       */
   /*             IACP_STATUS_SUCCESS                                   */
int BTPSAPI IACPTR_Write(unsigned char Register, unsigned char BytesToWrite, unsigned char
*WriteBuffer, unsigned char *Status);

   /* The following function is responsible for resetting the ACP       */
   /* hardware. The implementation should not return until the reset is */
   /* complete.                                                         */
   /* * NOTE * This function only applies to chips version 2.0B and     */
   /*          earlier. For chip version 2.0C and newer, this function  */
   /*          should be implemented as a NO-OP.                        */
void BTPSAPI IACPTR_Reset(void);

#endif
```