ARM® Cordio Profiles

ARM-EPM-115884 1.0

Profile and Service API

Confidential

Copyright © 2011-2016 ARM. All rights reserved.

ARM® Cordio Profile and Service API

Reference Guide

Copyright © 2011-2016 ARM. All rights reserved.

Release Information

The following changes have been made to this book:

Document History

Date	Issue	Confidentiality	Change
30 September 2015	-	Confidential	First Wicentric release for 1.1 as 2012-0021
1 March 2016	A	Confidential	First ARM release for 1.1
24 August 2016	A	Confidential	AUSPEX # / Added new profiles and services

Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of ARM. **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information: (i) for the purposes of determining whether implementations infringe any third party patents; (ii) for developing technology or products which avoid any of ARM's intellectual property; or (iii) as a reference for modifying existing patents or patent applications or creating any continuation, continuation in part, or extension of existing patents or patent applications; or (iv) for generating data for publication or disclosure to third parties, which compares the performance or functionality of the ARM technology described in this document with any other products created by you or a third party, without obtaining ARM's prior written consent.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, ARM makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to ARM's customers is not intended to create or refer to any partnership relationship with any other company. ARM may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any signed written agreement covering this document with ARM, then the signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited or its affiliates in the EU and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow ARM's trademark usage guidelines at http://www.arm.com/about/trademark-usage-guidelines.php

Where the term ARM is used it means "ARM or any of its subsidiaries as appropriate".

Copyright © 2011-2016, ARM Limited or its affiliates. All rights reserved.

ARM Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20348

Confidentiality Status

This document is Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

The information in this document is final, that is for a developed product.

Web Address

http://www.arm.com

Copyright © 2011-2016 ARM. All rights reserved.

Contents

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

Copyright © 2011-2016 ARM. All rights reserved.

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

1 Preface

This preface introduces the *Cordio Profile and Service API Reference Manual*.

1.1 About this book

This document describes the Cordio Profiles and Service API and lists the API functions and their parameters.

1.1.1 Using this book

This book is organized into the following chapters:

- **Introduction**
  Read this for an overview of the API for profiles and services.
- **Service API**
  Read this for an overview of the modules in the service API.
- **Service Example Walkthrough**
  Read this for an overview of a typical service.
- **Profile API**
  Read this for a description of the Profile API types and functions.
- **Revisions**
  Read this chapter for descriptions of the changes between document versions.

1.1.2 Intended audience

This book is written for experienced software engineers who might or might not have experience with ARM products. Such engineers typically have experience of writing Bluetooth applications but might have limited experience of the Cordio software stack.

It is also assumed that the readers have access to all necessary tools.

1.1.3 Terms and abbreviations

For a list of ARM terms, see the ARM glossary.

Terms specific to the Cordio software are listed below:

Term	Description
ACL	Asynchronous Connectionless data packet
AD	Advertising Data
ARQ	Automatic Repeat reQuest
ATT	Attribute Protocol, also attribute protocol software subsystem
ATTC	Attribute Protocol Client software subsystem
ATTS	Attribute Protocol Server software subsystem
CCC or CCCD	Client Characteristic Configuration Descriptor
CID	Connection Identifier
CSRK	Connection Signature Resolving Key
DM	Device Manager software subsystem

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

GAP	Generic Access Profile
GATT	Generic Attribute Profile
HCI	Host Controller Interface
IRK	Identity Resolving Key
JIT	Just In Time
L2C	L2CAP software subsystem
L2CAP	Logical Link Control Adaptation Protocol
LE	(Bluetooth) Low Energy
LL	Link Layer
LLPC	Link Layer Control Protocol
LTK	Long Term Key
MITM	Man In The Middle pairing (authenticated pairing)
OOB	Out Of Band data
SMP	Security Manager Protocol, also security manager protocol software subsystem
SMPI	Security Manager Protocol Initiator software subsystem
SMPR	Security Manager Protocol Responder software subsystem
STK	Short Term Key
WSF	Wireless Software Foundation software service and porting layer.

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

1.1.4 Conventions

The following table describes the typographical conventions:

Typographical conventions

Style	Purpose
Italic	Introduces special terminology, denotes cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`MONOSPACE`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`MONO`SPACE	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`monospace` *`italic`*	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the *ARM® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

1.1.5 Additional reading

This section lists publications by ARM and by third parties.

See *Infocenter* for access to ARM documentation.

Other publications

This section lists relevant documents published by third parties:

- Bluetooth SIG, "*Specification of the Bluetooth System*", Version 4.2, December 2, 2015.

1.2 Feedback

ARM welcomes feedback on this product and its documentation.

1.2.1 Feedback on content

If you have comments on content then send an e-mail to `errata@arm.com`. Give:

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

- The title.
- The number, ARM-EPM-115152.
- The page numbers to which your comments apply.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Note: ARM tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

2 Introduction

This document describes the API for profiles and services. The profiles and services are interoperable components that are designed to Bluetooth profile and service specification requirements. The profiles and services are used in applications to implement particular profile and service features.

2.1 Overview

The profiles are implemented in separate files for each profile role. The services may be grouped together in files based on their logical function and the profile they are used by.

The following standard services are supported:

- *Battery Service* (BAS)
- *Blood Pressure Service* (BPS)
- *Cycling Power Service (CPS)*
- *Cycling Speed and Cadence Service (CSCS)*
- D*evice Information Service* (DIS)
- *Glucose Service* (GLS)
- *Heart Rate Service* (HRS)
- *Health Thermometer Service* (HTS)
- *HID Service* (HIDS)
- *Immediate Alert Service* (IAS)
- *Internet Profiles Support Service (IPSS)*
- *Link Loss Service* (LLS)
- *Pulse Oximeter Service (PLXS)*
- *Runner's Speed and Cadence Service (RSCS)*
- *Scan Parameters Service (SCPS)*
- *TX Power Service* (TPS)
- *Weight Scale Service* (WSS)

The following standard profiles are supported:

- *Alert Notification Profile* (ANP)
- *Blood Pressure Profile* (BLP)

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

- *Cycling Power Profile (CPP)*
- *Cycling Speed and Cadence Profile (CSCP)*
- *Find Me Profile* (FMP)
- *Glucose Profile* (GLP)
- *Heart Rate Profile* (HRP)
- *Health Thermometer Profile* (HTP)
- *HID Over GATT Profile* (HOGP)
- *Phone Alert Status Profile* (PASP)
- *Proximity Profile* (PXP)
- Pulse Oximeter Profile (PLXP)
- Runner's Speed and Cadence Profile (RSCP)
- *Time Profile* (TIP)
- Weig*ht Scale Profile* (WSP)

3 Service API

Services are divided into separate modules as follows:

Table 1 Service modules

Service	Interface file
Battery Service	`svc_batt.h`
Blood Pressure Service	`svc_bps.h`
Cordio Proprietary Service	`svc_wp.h`
Cycling Power Service	`svc_cps.h`
Cycling Speed and Cadence Service	`svc_cscs.h`
Device Information Service	`svc_dis.h`
GAP Service GATT Service	`svc_core.h`
Glucose Service	`svc_gls.h`
Gyro Proprietary Service	`svc_gyro.h`
Heart Rate Service	`svc_hrs.h`
Health Thermometer Service	`svc_hts.h`

HID Service
Generic
Keyboard
Mouse
	`svc_hid.h`
	`svc_hidg.c`
	`svc_hidkb.c`
	`svc_hidm.c`

Immediate Alert Service Link Loss Service TX Power Service	`svc_px.h`
Internet Protocol Support Service	`svc_ipss.h`

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

Pulse Oximeter Service	svc_plxs.h
Runner's Speed and Cadence Service	svc_rscs.h
Scan Parameters Service	svc_scpss.h
Temperature Proprietary Service	svc_temp.h
Weight Scale Service	svc_wss.h

3.1 Functions

All service modules have equivalent API functions to add, remove, and configure the callbacks for a service. The generalized functions described in this section are applicable to all service modules.

Note that these generalized API functions are for illustration purposes only and not actually implemented in the code. The actual API functions for a service module are given in its interface file.

3.1.1 SvcAddGroup()

Add the attribute group for the service to the attribute database.

Syntax:

```
void SvcAddGroup(void)
```

This function is typically called once at system startup to set up and initialize a service.

3.1.2 SvcRemoveGroup()

Remove the attribute group for the service from the attribute database.

Syntax:

```
void SvcRemoveGroup(void)
```

The service will no longer be available to peer devices.

3.1.3 SvcCbackRegister()

Register the attribute read and write attribute callback functions for a service. These callback functions will be executed an attribute configured to use callbacks in its settings is accessed by a peer device.

If a callback is not used it can be set to NULL.

Syntax:

```
void SvcCbackRegister(attsReadCback_t readCback, attsWriteCback_t writeCback)
```

Where:

- `readCback`: Read callback. See the *Cordio Attribute Protocol API Reference Manual*.
- `writeCback`: Write callback. See the *Cordio Attribute Protocol API Reference Manual.*

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

4 Service Example Walkthrough

The best way to understand a service is by walking through an example. This walkthrough uses the battery service in files **svc_batt.h** and **svc_batt.c**.

4.1 Service Overview

The battery service is a simple service with only a few attributes. A diagram of the attributes in the battery service is shown in Figure 1.

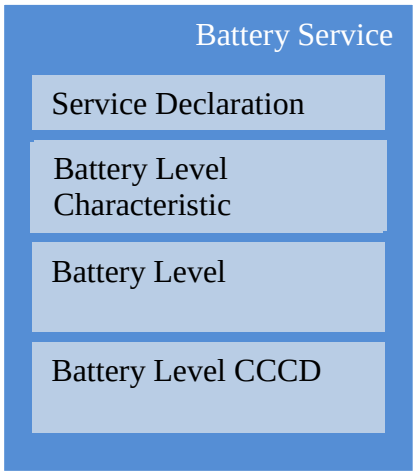

Figure 1. Battery service attributes.

The battery service contains four attributes: The service declaration, battery level characteristic declaration, the battery level, and the battery level client characteristic configuration descriptor (CCCD).

4.2 Attribute Handles

The attribute handles for the service are defined in `svc_batt.h`. Macro BATT_START_HDL defines the start value for the handle range used by the service. The start handle must be set so it does not overlap with the handles of any other service used by an application. The enumeration that follows defines the handle value for each attribute.

```
/* Battery Service */
#define BATT_START_HDL                    0x60
#define BATT_END_HDL                      (BATT_MAX_HDL - 1)


/* Battery Service Handles */
enum
{
  BATT_SVC_HDL = BATT_START_HDL,        /* Battery service declaration */
  BATT_LVL_CH_HDL,                      /* Battery level characteristic */
  BATT_LVL_HDL,                         /* Battery level */
  BATT_LVL_CH_CCC_HDL,                  /* Battery level CCCD */
  BATT_MAX_HDL
};
```

4.3 Read and Write Permissions

Two macros are provided to simplify the configuration of read and write security permissions for the attributes of the service. The security permissions control whether encryption is required before an attribute can be read or written. The macros are in `svc_batt.c`.

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

```
/*! Characteristic read permissions */
#ifndef BATT_SEC_PERMIT_READ
#define BATT_SEC_PERMIT_READ SVC_SEC_PERMIT_READ
#endif

/*! Characteristic write permissions */
#ifndef BATT_SEC_PERMIT_WRITE
#define BATT_SEC_PERMIT_WRITE SVC_SEC_PERMIT_WRITE
#endif
```

By default, the read and write permissions for the service are set to the global read and write settings in `svc_cfg.h`.

4.4 Data Structures

A service implementation consists of ATT protocol layer data structures containing attribute data. The data structures are related as shown in Figure 2.

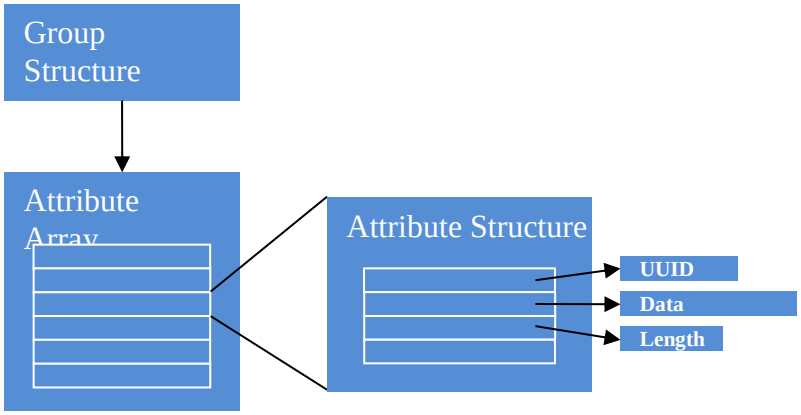

Figure 2. Service data structures.

The group structure contains a pointer to an attribute array and the handle range of the attributes it references. The attribute array is an array of structures with each structure containing a UUID, data, length, and other information for the attribute.

4.4.1 Attribute Variables

Variables containing the value and length of each attribute are defined in `svc_batt.c`.

```
/* Battery service declaration */
static const uint8_t battValSvc[] = {UINT16_TO_BYTES(ATT_UUID_BATTERY_SERVICE)};
static const uint16_t battLenSvc = sizeof(battValSvc);

/* Battery level characteristic */
static const uint8_t battValLvlCh[] = {ATT_PROP_READ | ATT_PROP_NOTIFY,
  UINT16_TO_BYTES(BATT_LVL_HDL), UINT16_TO_BYTES(ATT_UUID_BATTERY_LEVEL)};
static const uint16_t battLenLvlCh = sizeof(battValLvlCh);

/* Battery level */
static uint8_t battValLvl[] = {0};
static const uint16_t battLenLvl = sizeof(battValLvl);

/* Battery level client characteristic configuration */
```

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

```
static uint8_t battValLvlChCcc[] = {UINT16_TO_BYTES(0x0000)};
static const uint16_t battLenLvlChCcc = sizeof(battValLvlChCcc);
```

The value of the battery service declaration is the UUID for the battery service.

The value of the battery level characteristic declaration is a byte array with contents defined by the Bluetooth specification. It contains the properties, handle, and UUID of the battery level. The properties are configured to allow read and notification, as defined by the battery service specification.

The battery level value is a single byte as defined by the battery service specification.

The battery level CCCD is a 16-bit integer formatted as a little-endian byte array.

Note that the variables that cannot be changed are defined as `const` (to save RAM), while variables that can be changed, like the battery level and CCCD, are not const.

4.4.2 Attribute Array

The attribute variables defined above are used to construct an attribute structure for each attribute. These structures are contained in an array of type `attsAttr_t`.

```
typedef struct
{
  uint8_t const   *pUuid;      /*! Pointer to the attribute's UUID */
  uint8_t         *pValue;     /*! Pointer to the attribute's value */
  uint16_t        *pLen;       /*! Pointer to the length of the
                                   attribute's value */
  uint16_t        maxLen;      /*! Maximum length of attribute's value */
  uint8_t         settings;    /*! Attribute settings */
  uint8_t         permissions; /*! Attribute permissions */
} attsAttr_t;
```

The attribute array for battery service is shown below.

```
static const attsAttr_t battList[] =
{
  /* Service declaration */
  {
    attPrimSvcUuid,
    (uint8_t *) battValSvc,
    (uint16_t *) &battLenSvc,
    sizeof(battValSvc),
    0,
    ATTS_PERMIT_READ
  },
  /* Characteristic declaration */
  {
    attChUuid,
    (uint8_t *) battValLvlCh,
    (uint16_t *) &battLenLvlCh,
    sizeof(battValLvlCh),
    0,
    ATTS_PERMIT_READ
  },
  /* Characteristic value */
  {
    attBlChUuid,
    battValLvl,
    (uint16_t *) &battLenLvl,
    sizeof(battValLvl),
    ATTS_SET_READ_CBACK,
```

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

```
    BATT_SEC_PERMIT_READ
  },
  /* Characteristic CCC descriptor */
  {
    attCliChCfgUuid,
    battValLvlChCcc,
    (uint16_t *) &battLenLvlChCcc,
    sizeof(battValLvlChCcc),
    ATTS_SET_CCC,
    (ATTS_PERMIT_READ | BATT_SEC_PERMIT_WRITE)
  }
};
```

Note the following:

- The attribute permissions for the service declaration and the characteristic declaration are set to read-only and not overridden by the permissions macros. The Bluetooth specification requires that these types of attributes are always readable.
- The battery level characteristic value is set to use a read callback (`ATTS_SET_READ_CBACK`). When the battery level is read by a peer device it will execute a callback defined by the application. This callback must return the value of the battery level.
- The characteristic CCC descriptor has setting `ATTS_SET_CCC`. This identifies it as a CCC descriptor to the ATT protocol layer.

4.4.3 Group Structure

The group structure contains the attribute array and the handle start and end range for the service.

```
/* Battery group structure */
static attsGroup_t svcBattGroup =
{
  NULL,
  (attsAttr_t *) battList,
  NULL,
  NULL,
  BATT_START_HDL,
  BATT_END_HDL
};
```

5 Profile API

This section describes the types and functions that are in the profile API.

5.1 Alert Notification Profile Client

5.1.1 Handle Index Enumeration

Enumeration of handle indexes of characteristics to be discovered. These indexes give the location of their respective characteristic handles in the handle list returned from service and characteristic discovery.

Table 2 Alert notification profile handle index enumeration

Name	Description
ANPC_ANS_SNAC_HDL_IDX	Supported new alert category.
ANPC_ANS_NA_HDL_IDX	New alert.

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

ANPC_ANS_NA_CCC_HDL_IDX	New alert CCC descriptor.
ANPC_ANS_SUAC_HDL_IDX	Supported unread alert category.
ANPC_ANS_UAS_HDL_IDX	Unread alert status.
ANPC_ANS_UAS_CCC_HDL_IDX	Unread alert status CCC descriptor.
ANPC_ANS_ANCP_HDL_IDX	Alert notification control point.
ANPC_ANS_HDL_LIST_LEN	Handle list length.

5.1.2 AnpcAnsDiscover()

Perform service and characteristic discovery for Alert Notification service.

Syntax:

```
void AnpcAnsDiscover(dmConnId_t connId, uint16_t *pHdlList)
```

Where:

- connId: Connection identifier. See *Cordio Device Manager API Reference Manual*.
- pHdlList: Characteristic handle list.

Parameter pHdlList must point to an array of length ANPC_ANS_HDL_LIST_LEN. If discovery is successful the handles of discovered characteristics and descriptors will be set in pHdlList.

5.1.3 AnpcAnsControl()

Send a command to the alert notification control point.

Syntax:

```
void AnpcAnsControl(dmConnId_t connId, uint16_t handle, uint8_t command, uint8_t
        catId)
```

Where:

- connId: Connection identifier.
- handle: Attribute handle.
- command: Control point command.
- catId: Alert category ID.

5.1.4 AnpcAnsValueUpdate()

Process a value received in an ATT read response, notification, or indication message.

Syntax:

```
uint8_t AnpcAnsValueUpdate(uint16_t *pHdlList, attEvt_t *pMsg)
```

Where:

- pHdlList: Characteristic handle list.
- pMsg: ATT callback message.

Copyright © 2011-2016 ARM. All rights reserved.

Parameter pHdlList must point to an array of length ANPC_ANS_HDL_LIST_LEN. If the ATT handle of the message matches a handle in the handle list the value is processed, otherwise it is ignored.

Returns ATT_SUCCESS if handle is found, ATT_ERR_NOT_FOUND otherwise.

5.2 Battery Service Server

5.2.1 basCfg_t

Battery server configurable parameters.

Table 3 Batter service parameters

Type	Name	Description
wsfTimerTicks_t	period	Battery measurement timer expiration period in seconds.
uint16_t	count	Perform battery measurement after this many timer periods.
uint8_t	threshold	Send battery level notification to peer when below this level.

5.2.2 BasInit()

Initialize the battery service server.

Syntax:

```
void BasInit(wsfHandlerId_t handlerId, basCfg_t *pCfg)
```

Where:

- handerId: WSF handler ID of the application using this service.
- pCfg: Battery service configurable parameters.

5.2.3 BasMeasBattStart()

Start periodic battery level measurement.

This function starts a timer to perform periodic battery measurements.

Syntax:

```
void BasMeasBattStart(dmConnId_t connId, uint8_t timerEvt, uint8_t battCccIdx)
```

Where:

- connId: DM connection identifier.
- timerEvt: WSF event designated by the application for the timer.
- battCccIdx: Index of battery level CCC descriptor in CCC descriptor handle table.

5.2.4 BasMeasBattStop()

Stop periodic battery level measurement.

Syntax:

```
void BasMeasBattStop(void)
```

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

5.2.5 BasProcMsg()

This function is called by the application when the battery periodic measurement timer expires.

Syntax:

```
void BasProcMsg(wsfMsgHdr_t *pMsg)
```

Where:

- `pMsg`: Event message.

5.2.6 BasSendBattLevel()

Send the battery level to the peer device.

Syntax:

```
void BasSendBattLevel(dmConnId_t connId, uint8_t battCccIdx, uint8_t level)
```

Where:

- `connId`: DM connection identifier.
- `battCccIdx`: Index of battery level CCC descriptor in CCC descriptor handle table.
- `level`: The battery level.

5.2.7 BasReadCback()

ATTS read callback for battery service used to read the battery level.

Syntax:

```
uint8_t BasReadCback(dmConnId_t connId, uint16_t handle, uint8_t operation,
          uint16_t offset, attsAttr_t *pAttr)
```

Where:

- `connId`: DM connection identifier.
- `handle`: Attribute handle.
- `operation`: ATT operation.
- `offset`: Attribute read/write offset.
- `pAttr`: Pointer to attribute structure.

Use this function as a parameter to `SvcBattCbackRegister()`.

5.3 Blood Pressure Profile Client

5.3.1 Handle Index Enumeration

Enumeration of handle indexes of characteristics to be discovered. These indexes give the location of their respective characteristic handles in the handle list returned from service and characteristic discovery.

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

Table 4 Blood pressure profile handle index enumeration

Name	Description
BLPC_BPS_BPM_HDL_IDX	Blood pressure measurement.
BLPC_BPS_BPM_CCC_HDL_IDX	Blood pressure measurement CCC descriptor.
BLPC_BPS_ICP_HDL_IDX	Intermediate cuff pressure.
BLPC_BPS_ICP_CCC_HDL_IDX	Intermediate cuff pressure CCC descriptor.
BLPC_BPS_BPF_HDL_IDX	Blood pressure feature.
BLPC_BPS_HDL_LIST_LEN	Handle list length.

5.3.2 BlpcBpsDiscover()

Perform service and characteristic discovery for Blood Pressure service.

Syntax:

```
void BlpcBpsDiscover(dmConnId_t connId, uint16_t *pHdlList)
```

Where:

- connId: Connection identifier.
- pHdlList: Characteristic handle list.

Parameter pHdlList must point to an array of length BLPC_BPS_HDL_LIST_LEN. If discovery is successful the handles of discovered characteristics and descriptors will be set in pHdlList.

5.3.3 BlpcBpsValueUpdate()

Process a value received in an ATT read response, notification, or indication message.

Syntax:

```
uint8_t BlpcBpsValueUpdate(uint16_t *pHdlList, attEvt_t *pMsg)
```

Where:

- connId: Connection identifier.
- pMsg: ATT callback message.

Parameter pHdlList must point to an array of length BLPC_BPS_HDL_LIST_LEN. If the ATT handle of the message matches a handle in the handle list the value is processed, otherwise it is ignored.

Returns ATT_SUCCESS if handle is found, ATT_ERR_NOT_FOUND otherwise.

5.4 Blood Pressure Profile Sensor

5.4.1 blpsCfg_t

Blood pressure sensor configurable parameters.

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

Table 5 Blood pressure profile parameters

Type	Name	Description
wsfTimerTicks_t	period	Measurement timer expiration period in ms.

5.4.2 BlpsInit()

Initialize the Blood Pressure profile sensor.

Syntax:

```
void BlpsInit(wsfHandlerId_t handlerId, blpsCfg_t *pCfg)
```

Where:

- handerId: WSF handler ID of the application using this service.
- pCfg: Configurable parameters.

5.4.3 BlpsMeasStart()

Start periodic blood pressure measurement. This function starts a timer to perform periodic measurements.

Syntax:

```
void BlpsMeasStart(dmConnId_t connId, uint8_t timerEvt, uint8_t icpCccIdx)
```

Where:

- connId: DM connection identifier.
- timerEvt: WSF event designated by the application for the timer.
- icpCccIdx: Index of intermediate cuff pressure CCC descriptor in CCC descriptor handle table.

5.4.4 BlpsMeasStop()

Stop periodic blood pressure measurement.

Syntax:

```
void BlpsMeasStop(void)
```

5.4.5 BlpsMeasComplete()

Blood pressure measurement complete.

Syntax:

```
void BlpsMeasComplete(dmConnId_t connId, uint8_t bpmCccIdx)
```

Where:

- connId: DM connection identifier.

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

- bpmCccIdx: Index of blood pressure measurement CCC descriptor in CCC descriptor handle table.

5.4.6 BlpsProcMsg()

This function is called by the application when the periodic measurement timer expires.

Syntax:

```
void BlpsProcMsg(wsfMsgHdr_t *pMsg)
```

Where:

- pMsg: Event message.

5.4.7 BlpsSetBpmFlags()

Set the blood pressure measurement flags.

Syntax:

```
void BlpsSetBpmFlags(uint8_t flags)
```

Where:

- flags: Blood pressure measurement flags.

5.4.8 BlpsSetIcpFlags()

Set the intermediate cuff pressure flags.

Syntax:

```
void BlpsSetIcpFlags(uint8_t flags)
```

Where:

- flags: Intermediate cuff pressure flags.

5.5 Device Information Service Client

5.5.1 Handle Index Enumeration

Enumeration of handle indexes of characteristics to be discovered. These indexes give the location of their respective characteristic handles in the handle list returned from service and characteristic discovery.

Table 6 Device information handle index enumeration

Name	Description
DIS_MFNS_HDL_IDX	Manufacturer name string.
DIS_MNS_HDL_IDX	Model number string.
DIS_SNS_HDL_IDX	Serial number string.
DIS_HRS_HDL_IDX	Hardware revision string.
DIS_FRS_HDL_IDX	Firmware revision string.

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

DIS_SRS_HDL_IDX	Software revision string.
DIS_SID_HDL_IDX	System ID.
DIS_HDL_LIST_LEN	Handle list length.

5.5.2 DisDiscover()

Perform service and characteristic discovery for DIS service.

Syntax:

```
void DisDiscover(dmConnId_t connId, uint16_t *pHdlList)
```

Where:

- connId: Connection identifier.
- pHdlList: Characteristic handle list.

Note that pHdlList must point to an array of handles of length DIS_HDL_LIST_LEN. If discovery is successful the handles of discovered characteristics and descriptors will be set in pHdlList.

5.5.3 DisValueUpdate()

Process a value received in an ATT read response, notification, or indication message.

Syntax:

```
uint8_t DisValueUpdate(uint16_t *pHdlList, attEvt_t *pMsg)
```

Where:

- pHdlList: Characteristic handle list.
- pMsg: ATT callback message.

Parameter pHdlList must point to an array of length DIS_HDL_LIST_LEN. If the attribute handle of the message matches a handle in the handle list the value is processed, otherwise it is ignored.

Returns ATT_SUCCESS if handle is found, ATT_ERR_NOT_FOUND otherwise.

5.6 Find Me Profile Locator

5.6.1 Handle Index Enumeration

Enumeration of handle indexes of characteristics to be discovered. These indexes give the location of their respective characteristic handles in the handle list returned from service and characteristic discovery.

Table 7 Find Me profile handle index enumeration

Name	Description
FMPL_IAS_AL_HDL_IDX	Alert level .
FMPL_IAS_HDL_LIST_LEN	Handle list length.

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

5.6.2 FmplIasDiscover()

Perform service and characteristic discovery for Immediate Alert service.

Syntax:

```
void FmplIasDiscover(dmConnId_t connId, uint16_t *pHdlList)
```

Where:

- `connId`: Connection identifier.
- `pHdlList`: Characteristic handle list.

Note that `pHdlList` must point to an array of handles of length `FMPL_IAS_HDL_LIST_LEN`.

If discovery is successful the handles of discovered characteristics and descriptors will be set in `pHdlList`.

5.6.3 FmplSendAlert()

Send an immediate alert to the peer device.

Syntax:

```
void FmplSendAlert(dmConnId_t connId, uint16_t handle, uint8_t aler)
```

Where:

- `connId`: DM connection ID.
- `handle`: Attribute handle.
- `alert`: Alert value.

5.7 GAP Client

5.7.1 Handle Index Enumeration

Enumeration of handle indexes of characteristics to be discovered. These indexes give the location of their respective characteristic handles in the handle list returned from service and characteristic discovery.

Table 8 GAP handle index enumeration

Name	Description
GAP_CAR_HDL_IDX	Central address resolution.
GAP_HDL_LIST_LEN	Handle list length.

5.7.2 GapDiscover()

Perform service and characteristic discovery for GAP service.

Syntax:

```
void GapDiscover(dmConnId_t connId, uint16_t *pHdlList)
```

Where:

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

- connId: Connection identifier.
- pHdlList: Characteristic handle list.

Note that pHdlList must point to an array of handles of length GAP_HDL_LIST_LEN.

If discovery is successful the handles of discovered characteristics and descriptors will be set in pHdlList.

5.7.3 GapValueUpdate()

Process a value received in an ATT read response, notification, or indication message.

Syntax:

```
uint8_t GapValueUpdate(uint16_t *pHdlList, attEvt_t *pMsg)
```

Where:

- pHdlList: Characteristic handle list.
- pMsg: ATT callback message.

Parameter pHdlList must point to an array of length GATT_HDL_LIST_LEN. If the attribute handle of the message matches a handle in the handle list the value is processed, otherwise it is ignored.

Returns ATT_SUCCESS if handle is found, ATT_ERR_NOT_FOUND otherwise.

5.8 GATT Client

5.8.1 Handle Index Enumeration

Enumeration of handle indexes of characteristics to be discovered. These indexes give the location of their respective characteristic handles in the handle list returned from service and characteristic discovery.

Table 9 GATT handle index enumeration

Name	Description
GATT_SC_HDL_IDX	Service changed.
GATT_SC_CCC_HDL_IDX	Service changed client characteristic configuration descriptor.
GATT_HDL_LIST_LEN	Handle list length.

5.8.2 GattDiscover()

Perform service and characteristic discovery for GATT service.

Syntax:

```
void GattDiscover(dmConnId_t connId, uint16_t *pHdlList)
```

Where:

- connId: Connection identifier.

Copyright © 2011-2016 ARM. All rights reserved.

- pHdlList: Characteristic handle list.

Note that pHdlList must point to an array of handles of length GATT_HDL_LIST_LEN.

If discovery is successful the handles of discovered characteristics and descriptors will be set in pHdlList.

5.8.3 GattValueUpdate()

Process a value received in an ATT read response, notification, or indication message.

Syntax:

```
uint8_t GattValueUpdate(uint16_t *pHdlList, attEvt_t *pMsg)
```

Where:

- pHdlList: Characteristic handle list.
- pMsg: ATT callback message.

Parameter pHdlList must point to an array of length GATT_HDL_LIST_LEN. If the attribute handle of the message matches a handle in the handle list the value is processed, otherwise it is ignored.

Returns ATT_SUCCESS if handle is found, ATT_ERR_NOT_FOUND otherwise.

5.9 Glucose Profile Client

5.9.1 Handle Index Enumeration

Enumeration of handle indexes of characteristics to be discovered. These indexes give the location of their respective characteristic handles in the handle list returned from service and characteristic discovery.

Table 10 Glusose profile handle index enumeration

Name	Description
GLPC_GLS_GLM_HDL_IDX	Glucose measurement.
GLPC_GLS_GLM_CCC_HDL_IDX	Glucose measurement CCC descriptor.
GLPC_GLS_GLMC_HDL_IDX	Glucose measurement context.
GLPC_GLS_GLMC_CCC_HDL_IDX	Glucose measurement context CCC descriptor.
GLPC_GLS_GLF_HDL_IDX	Glucose feature.
GLPC_GLS_RACP_HDL_IDX	Record access control point.
GLPC_GLS_RACP_CCC_HDL_IDX	Record access control point CCC descriptor.
GLPC_GLS_HDL_LIST_LEN	Handle list length.

5.9.2 glpcFilter_t

Glucose service RACP filter type.

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

Table 11 Glusose service filter type

Type	Name	Description
uint16_t	seqNum	Sequence number filter.
uint8_t	type	Filter type.

5.9.3 GlpcGlsDiscover()

Perform service and characteristic discovery for Glucose service.

Syntax:

```
void GlpcGlsDiscover(dmConnId_t connId, uint16_t *pHdlList)
```

Where:

- connId: Connection identifier.
- pHdlList: Characteristic handle list.

Parameter pHdlList must point to an array of length GLPC_GLS_HDL_LIST_LEN. If discovery is successful the handles of discovered characteristics and descriptors will be set in pHdlList.

5.9.4 GlpcGlsValueUpdate()

Process a value received in an ATT read response, notification, or indication message.

Syntax:

```
uint8_t GlpcGlsValueUpdate(uint16_t *pHdlList, attEvt_t *pMsg)
```

Where:

- pHdlList: Characteristic handle list.
- pMsg: ATT callback message.

Parameter pHdlList must point to an array of length GLPC_GLS_HDL_LIST_LEN. If the ATT handle of the message matches a handle in the handle list the value is processed, otherwise it is ignored.

Return ATT_SUCCESS if handle is found, ATT_ERR_NOT_FOUND otherwise.

5.9.5 GlpcGlsRacpSend()

Send a command to the glucose service record access control point.

Syntax:

```
void GlpcGlsRacpSend(dmConnId_t connId, uint16_t handle, uint8_t opcode, uint8_t
        oper, glpcFilter_t *pFilter)
```

Where:

- connId: Connection identifier.
- handle: Attribute handle.
- opcode: Command opcode.
- oper: Command operator or 0 if no operator required.

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

- pFliter: Command filter parameters or NULL of no parameters required.

5.9.6 GlpcGlsSetLastSeqNum()

Set the last received glucose measurement sequence number.

Syntax:

```
vovoid GlpcGlsSetLastSeqNum(uint16_t seqNumid)
```

Where:

- seqNum: Glucose measurement sequence number.

5.9.7 GlpcGlsGetLastSeqNum()

Return the last received glucose measurement sequence number.

Syntax:

```
uint16_t GlpcGlsGetLastSeqNum(void)
```

5.10 Glucose Profile Sensor

5.10.1 GlpsInit()

Initialize the Glucose profile sensor.

Syntax:

```
void GlpsInit(void)
```

5.10.2 GlpsProcMsg()

This function is called by the application when a message that requires processing by the glucose profile sensor is received.

Syntax:

```
void GlpsProcMsg(wsfMsgHdr_t *pMsg)
```

Where:

- pMsg: Event message.

5.10.3 GlpsRacpWriteCback()

ATTS write callback for glucose service record access control point.

Syntax:

```
uint8_t GlpsRacpWriteCback(dmConnId_t connId, uint16_t handle, uint8_t
         operation, uint16_t offset, uint16_t len, uint8_t *pValue, attsAttr_t
         *pAttr)
```

Where:

- TBD

Use this function as a parameter to SvcGlsCbackRegister().

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

5.10.4 GlpsSetFeature()

Set the supported features of the glucose sensor.

Syntax:

```
void GlpsSetFeature(uint16_t feature)
```

Where:

- `feature`: Feature bitmask.

5.10.5 GlpsSetCccIdx()

Set the CCCD index used by the application for glucose service characteristics.

Syntax:

```
void GlpsSetCccIdx(uint8_t glmCccIdx, uint8_t glmcCccIdx, uint8_t racpCccIdx)
```

Where:

- `glmCccIdx`: Glucose measurement CCCD index.
- `glmcCccIdx`: Glucose measurement context CCCD index.
- `racpCccIdx`: Record access control point CCCD index.

5.11 HID Device Profile

The HID, Human Interface Device, Profile provides functions to support HID Devices such as Keyboards, Computer Mice, and Remote Controls.

5.11.1 HidSendInputReport()

The `HidSendInputReport()` function is used to send a HID input report to the host.

Syntax:

```
void HidSendInputReport(dmConnId_t connId, uint8_t reportId, uint16_t len,
         uint8_t *pValue)
```

Where:

- `connId`: Connection identifier.
- `reportId`: The identifier of the report
- `len`: The length of pValue in bytes
- `pValue`: The contents of the report to be sent to the host

Note: The `reportId` value must correspond to one of the entries in the `hidReportIdMap_t` map defined in the HID application.

5.11.2 HidSetProtocolMode()

The `HidSetProtocolMode()` function is used to set the Protocol Mode to Report or Boot Mode.

Syntax:

```
void HidSetProtocolMode(uint8_t protocolMode)
```

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

Where:

- protocolMode: The protocol mode (HID_PROTOCOL_MODE_REPORT or HID_PROTOCOL_MODE_BOOT).

Note: The protocol mode is only used in devices that support HID boot keyboard and/or boot mouse. This function is generally called once on connection establishment to restore the protocol mode to the default value.

5.11.3 HidGetProtocolMode()

The HidGetProtocolMode() function is used to get the value of the Protocol Mode.

Syntax:

```
uint8_t HidGetProtocolMode(void)
```

Protocol mode is only used in devices that support HID boot keyboard and/or boot mouse. The Host may alter the protocol mode at any time. Applications supporting Protocol Mode may check the protocol mode before sending input reports to the Host to determine the proper format of the report.

5.11.4 HidGetControlPoint()

The HidGetControlPoint() function is used to get the value of the Control Point. The control point indicates if the Host is operational or in suspended mode.

Syntax:

```
uint8_t HidGetControlPoint(void)
```

5.11.5 HidInit()

The HidInit() function is used to initialize the HID profile.

Syntax:

```
void HidInit(const hidConfig_t *pConfig)
```

Where:

- pConfig: The HID Configuration structure. For more information about the HID Configuration structure, see the Cordio Sample App Users Guide.

5.11.6 Callback Functions

5.11.6.1 (*hidOutputReportCback_t)()

The hidOutputReportCback_t callback is called to notify the application of receipt of a HID output report from the Host.

Syntax:

```
typedef void (*hidOutputReportCback_t)(dmConnId_t connId, uint8_t id, uint16_t
         len, uint8_t *pReport)
```

Where:

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

- connId: Connection identifier.
- id: The report ID
- len: The length of the pReport in bytes.
- pReport: The output report data

Note: The id value will correspond to one of the entries in the hidReportIdMap_t map defined in the HID application.

5.11.6.2 (*hidFeatureReportCback_t)()

The hidFeatureReportCback_t callback is called to notify the application of receipt of a HID feature report from the Host.

Syntax:

```
typedef void (*hidFeatureReportCback_t)(dmConnId_t connId, uint8_t id, uint16_t
         len, uint8_t *pReport)
```

Where:

- connId: Connection identifier.
- id: The report ID
- len: The length of the pReport in bytes.
- pReport: The feature report data

Note: The id value will correspond to one of the entries in the hidReportIdMap_t map defined in the HID application.

5.11.6.3 (*hidInfoCback_t)()

The hidInfoCback_t callback is called to notify the application of a change in protocol mode or control point by the host.

Syntax:

```
typedef void (*hidInfoCback_t)(dmConnId_t connId, uint8_t type, uint8_t value)
```

Where:

- connId: Connection identifier.
- type: The type of information
- value: The information.

5.12 Heart Rate Profile Client

5.12.1 Handle Index Enumeration

Enumeration of handle indexes of characteristics to be discovered. These indexes give the location of their respective characteristic handles in the handle list returned from service and characteristic discovery.

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

Table 12 Heart rate profile handle index enumeration

Name	Description
HRPC_HRS_HRM_HDL_IDX	Heart rate measurement.
HRPC_HRS_HRM_CCC_HDL_IDX	Heart rate measurement CCC descriptor.
HRPC_HRS_BSL_HDL_IDX	Body sensor location.
HRPC_HRS_HRCP_HDL_IDX	Heart rate control point.
HRPC_HRS_HDL_LIST_LEN	Handle list length.

5.12.2 HrpcHrsDiscover()

Perform service and characteristic discovery for Heart Rate service.

Syntax:

```
void HrpcHrsDiscover(dmConnId_t connId, uint16_t *pHdlList)
```

Where:

- connId: Connection identifier.
- pHdlList: Characteristic handle list.

Parameter pHdlList must point to an array of length HRPC_HRS_HDL_LIST_LEN. If discovery is successful the handles of discovered characteristics and descriptors will be set in pHdlList.

5.12.3 HrpcHrsControl()

Send a command to the heart rate control point.

Syntax:

```
void HrpcHrsControl(dmConnId_t connId, uint16_t handle, uint8_t command)
```

Where:

- connId: Connection identifier.
- handle: Attribute handle.
- command: Control point command.

5.12.4 HrpcHrsValueUpdate()

Process a value received in an ATT read response, notification, or indication message.

Syntax:

```
uint8_t HrpcHrsValueUpdate(uint16_t *pHdlList, attEvt_t *pMsg)
```

Where:

- pHdlList: Parameter pHdlList must point to an array of length HRPC_HRS_HDL_LIST_LEN.
- pMsg: ATT callback message.

If the ATT handle of the message matches a handle in the handle list the value is processed, otherwise

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

it is ignored.

5.13 Heart Rate Profile Sensor

5.13.1 hrpsCfg_t

Heart rate service configurable parameters.

Table 13 Heart rate service parameters

Type	Name	Description
wsfTimerTicks_t	period	Measurement timer expiration period in ms.

5.13.2 HrpsInit()

Initialize the Heart Rate profile sensor.

Syntax:

```
void HrpsInit(wsfHandlerId_t handlerId, hrpsCfg_t *pCfg)
```

Where:

- handerId: WSF handler ID of the application using this service.
- pCfg: Configurable parameters.

5.13.3 HrpsMeasStart()

Start periodic heart rate measurement. This function starts a timer to perform periodic measurements.

Syntax:

```
void HrpsMeasStart(dmConnId_t connId, uint8_t timerEvt, uint8_t hrmCccIdx)
```

Where:

- connId: DM connection identifier.
- timerEvt: WSF event designated by the application for the timer.
- hrmCccIdx: Index of heart rate CCC descriptor in CCC descriptor handle table.

5.13.4 HrpsMeasStop()

Stop periodic heart rate measurement.

Syntax:

```
void HrpsMeasStop(void)
```

5.13.5 HrpsProcMsg()

This function is called by the application when the periodic measurement timer expires.

Syntax:

```
void HrpsProcMsg(wsfMsgHdr_t *pMsg)
```

Where:

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

- pMsg: Event message.

5.13.6 HrpsWriteCback()

ATTS write callback for heart rate service. Use this function as a parameter to SvcHrsCbackRegister().

Syntax:

```
uint8_t HrpsWriteCback(dmConnId_t connId, uint16_t handle, uint8_t operation,
          uint16_t offset, uint16_t len, uint8_t *pValue, attsAttr_t *pAttr)
```

Where:

5.13.7 HrpsSetFlags()

Set the heart rate measurement flags.

Syntax:

```
void HrpsSetFlags(uint8_t flags)
```

Where:

- flags: Heart rate measurement flags.

5.14 Health Thermometer Profile Client

5.14.1 Handle Index Enumeration

Enumeration of handle indexes of characteristics to be discovered. These indexes give the location of their respective characteristic handles in the handle list returned from service and characteristic discovery.

Table 14 Health thermometer index enumeration

Name	Description
HTPC_HTS_TM_HDL_IDX	Temperature measurement.
HTPC_HTS_TM_CCC_HDL_IDX	Temperature measurement CCC descriptor.
HTPC_HTS_IT_HDL_IDX	Intermediate temperature.
HTPC_HTS_IT_CCC_HDL_IDX	Intermediate temperature CCC descriptor.
HTPC_HTS_TT_HDL_IDX	Temperature type.
HTPC_HTS_HDL_LIST_LEN	Handle list length.

5.14.2 HtpcHtsDiscover()

Perform service and characteristic discovery for Health Thermometer service.

Syntax:

```
void HtpcHtsDiscover(dmConnId_t connId, uint16_t *pHdlList)
```

Where:

- connId: Connection identifier.
- pHdlList: Characteristic handle list.

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

Parameter pHdlList must point to an array of length HTPC_HTS_HDL_LIST_LEN. If discovery is successful the handles of discovered characteristics and descriptors will be set in pHdlList.

5.14.3 HtpcHtsValueUpdate()

Process a value received in an ATT read response, notification, or indication message.

Syntax:

```
uint8_t HtpcHtsValueUpdate(uint16_t *pHdlList, attEvt_t *pMsg)
```

Where:

- connId: Connection identifier.
- pMsg: ATT callback message.

Parameter pHdlList must point to an array of length HTPC_HTS_HDL_LIST_LEN. If the ATT handle of the message matches a handle in the handle list the value is processed, otherwise it is ignored.

Returns ATT_SUCCESS if handle is found, ATT_ERR_NOT_FOUND otherwise.

5.15 Health Thermometer Profile Sensor

5.15.1 htpsCfg_t

Health Thermometer profile sensor configurable parameters.

Table 15 Health thermometer parameters

Type	Name	Description
wsfTimerTicks_t	period	Measurement timer expiration period in ms.

5.15.2 HtpsInit()

Initialize the Health Thermometer profile sensor.

Syntax:

```
void HtpsInit(wsfHandlerId_t handlerId, htpsCfg_t *pCfg)
```

Where:

- handerId: WSF handler ID of the application using this service.
- pCfg: Configurable parameters.

5.15.3 HtpsMeasStart()

Start periodic temperature measurement. This function starts a timer to perform periodic measurements.

Syntax:

```
void HtpsMeasStart(dmConnId_t connId, uint8_t timerEvt, uint8_t itCccIdx)
```

Where:

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

- connId: DM connection identifier.
- timerEvt: WSF event designated by the application for the timer.
- itCccIdx: Index of intermediate temperature CCC descriptor in CCC descriptor handle table.

5.15.4 HtpsMeasStop()

Stop periodic temperature measurement.

Syntax:

```
void HtpsMeasStop(void)
```

5.15.5 HtpsMeasComplete()

Temperature measurement complete.

Syntax:

```
void HtpsMeasComplete(dmConnId_t connId, uint8_t tmCccIdx)
```

Where:

- connId: DM connection identifier.
- tmCccIdx: Index of temperature measurement CCC descriptor in CCC descriptor handle table.

5.15.6 HtpsProcMsg()

This function is called by the application when the periodic measurement timer expires.

Syntax:

```
void HtpsProcMsg(wsfMsgHdr_t *pMsg)
```

Where:

- pMsg: Event message.

5.15.7 HtpsSetTmFlags()

Set the temperature measurement flags.

Syntax:

```
void HtpsSetTmFlags(uint8_t flags)
```

Where:

- flags: Temperature measurement flags.

5.15.8 HtpsSetItFlags()

Set the intermediate temperature flags.

Syntax:

```
void HtpsSetItFlags(uint8_t flags)
```

Where:

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

- flags: Intermediate temperature flags.

5.16 Phone Alert Status Profile Client

5.16.1 Handle Index Enumeration

Enumeration of handle indexes of characteristics to be discovered. These indexes give the location of their respective characteristic handles in the handle list returned from service and characteristic discovery.

Table 16 Phone alert handle index enumeration

Name	Description
PASPC_PASS_AS_HDL_IDX	Alert status.
PASPC_PASS_AS_CCC_HDL_IDX	Alert status CCC descriptor.
PASPC_PASS_RS_HDL_IDX	Ringer setting.
PASPC_PASS_RS_CCC_HDL_IDX	Ringer setting CCC descriptor.
PASPC_PASS_RCP_HDL_IDX	Ringer control point.
PASPC_PASS_HDL_LIST_LEN	Handle list length.

5.16.2 PaspcPassDiscover()

Perform service and characteristic discovery for Phone Alert Status service.

Syntax:

```
void PaspcPassDiscover(dmConnId_t connId, uint16_t *pHdlList)
```

Where:

- connId: Connection identifier.
- pHdlList: Characteristic handle list.

Parameter pHdlList must point to an array of length PASPC_PASS_HDL_LIST_LEN. If discovery is successful the handles of discovered characteristics and descriptors will be set in pHdlList.

5.16.3 PaspcPassControl()

Send a command to the ringer control point.

Syntax:

```
void PaspcPassControl(dmConnId_t connId, uint16_t handle, uint8_t command)
```

Where:

- connId: Connection identifier.
- handle: Attribute handle.
- command: Control point command.

5.16.4 PaspcPassValueUpdate()

Process a value received in an ATT read response, notification, or indication message.

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

Syntax:

```
uint8_t PaspcPassValueUpdate(uint16_t *pHdlList, attEvt_t *pMsg)
```

Where:

- pHdlList: Characteristic handle list.
- pMsg: ATT callback message.

Parameter pHdlList must point to an array of length PASPC_PASS_HDL_LIST_LEN. If the attribute handle of the message matches a handle in the handle list the value is processed, otherwise it is ignored.

Returns ATT_SUCCESS if handle is found, ATT_ERR_NOT_FOUND otherwise.

5.17 Pulse Oximeter Profile Client

5.17.1 Handle Index Enumeration

Enumeration of handle indexes of characteristics to be discovered. These indexes give the location of their respective characteristic handles in the handle list returned from service and characteristic discovery.

Table 17 Glusose profile handle index enumeration

Name	Description
PLXPC_PLXS_PLXSC_HDL_IDX	Pulse oximeter spot check measurement.
PLXPC_PLXS_PLXSC_CCC_HDL_IDX	Pulse oximeter spot check measurement CCC descriptor.
PLXPC_PLXS_PLXC_HDL_IDX	Pulse oximeter continuous measurement.
PLXPC_PLXS_PLXC_CCC_HDL_IDX	Pulse oximeter continuous measurement CCC descriptor.
PLXPC_PLXS_PLXF_HDL_IDX	Pulse oximeter features.
PLXPC_PLXS_RACP_HDL_IDX	Record access control point.
PLXPC_PLXS_RACP_CCC_HDL_IDX	Record access control point CCC descriptor.
PLXPC_PLXS_HDL_LIST_LEN	Handle list length.

5.17.2 PlxpcPlxsDiscover()

Perform service and characteristic discovery for Pulse oximeter service.

Syntax:

```
void PlxpcPlxsDiscover(dmConnId_t connId, uint16_t *pHdlList)
```

Where:

- connId: Connection identifier.
- pHdlList: Characteristic handle list.

Parameter pHdlList must point to an array of length PLXPC_PLXS_HDL_LIST_LEN. If discovery is successful the handles of discovered characteristics and descriptors will be set in pHdlList.

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

5.17.3 PlxpcPlxsValueUpdate()

Process a value received in an ATT read response, notification, or indication message.

Syntax:

```
uint8_t PlxpcPlxsValueUpdate(uint16_t *pHdlList, attEvt_t *pMsg)
```

Where:

- `pHdlList`: Characteristic handle list.
- `pMsg`: ATT callback message.

Parameter `pHdlList` must point to an array of length `PLXPC_PLXS_HDL_LIST_LEN`. If the ATT handle of the message matches a handle in the handle list the value is processed, otherwise it is ignored.

Return `ATT_SUCCESS` if handle is found, `ATT_ERR_NOT_FOUND` otherwise.

5.17.4 PlxpcPlxsRacpSend()

Send a command to the pulse oximeter service record access control point.

Syntax:

```
void PlxpcPlxsRacpSend(dmConnId_t connId, uint16_t handle, uint8_t opcode,
         uint8_t oper)
```

Where:

- `connId`: Connection identifier.
- `handle`: Attribute handle.
- `opcode`: Command opcode.
- `oper`: Command operator or 0 if no operator required.

5.18 Pulse Oximeter Profile Sensor

5.18.1 plxpsCfg_t

Pulse oximeter sensor configurable parameters.

Table 18 Pulse oximeter profile parameters

Type	Name	Description
`wsfTimerTicks_t`	`period`	Measurement timer expiration period in ms.

5.18.2 PlxpsInit()

Initialize the Pulse Oximeter profile sensor.

Syntax:

```
void PlxpsInit(wsfHandlerId_t handlerId, plxpsCfg_t *pCfg)
```

Where:

- `handerId`: WSF handler ID of the application using this service.

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

- pCfg: Configurable parameters.

5.18.3 PlxpsProcMsg()

This function is called by the application when a message that requires processing by the pulse oximeter profile sensor is received.

Syntax:

```
void PlxpsProcMsg(wsfMsgHdr_t *pMsg)
```

Where:

- pMsg: Event message.

5.18.4 PlxpsBtn()

Handle a button press.

Syntax:

```
void PlxpsBtn(dmConnId_t connId, uint8_t btn)
```

Where:

- connId: connection identifier.
- btn: Button press.

5.18.5 PlxpsWriteCback()

ATTS write callback for pulse oximeter service record access control point.

Syntax:

```
uint8_t PlxpsWriteCback(dmConnId_t connId, uint16_t handle, uint8_t operation,
          uint16_t offset, uint16_t len, uint8_t *pValue, attsAttr_t *pAttr)
```

Use this function as a parameter to SvcPlxsCbackRegister().

5.18.6 PlxpsSetFeature()

Set the supported features of the pulse oximeter sensor.

Syntax:

```
void PlxpsSetFeature(uint16_t feature, uint16_t measStatus, uint32_t
          sensorStatus)
```

Where:

- feature: Feature bitmask.
- measStatus: Measurement status.
- sensorStatus: Sensor status.

5.18.7 PlxpsSetCccIdx()

Set the CCCD index used by the application for pulse oximeter service characteristics.

Syntax:

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

```
void PlxpsSetCccIdx(uint8_t plxmCccIdx, uint8_t plxmcCccIdx, uint8_t racpCccIdx)
```

Where:

- `plxmCccIdx`: Pulse Oximeter measurement CCCD index.
- `plxmcCccIdx`: Pulse Oximeter measurement context CCCD index.
- `racpCccIdx`: Record access control point CCCD index.

5.18.8 PlxpsMeasStart()

Start periodic pulse oximeter measurement. This function starts a timer to perform periodic measurements.

Syntax:

```
void PlxpsMeasStart(dmConnId_t connId, uint8_t timerEvt, uint8_t plxmCccIdx)
```

Where:

- `connId`: DM connection identifier.
- `timerEvt`: WSF event designated by the application for the timer.
- `plxmCccIdx`: Index of the pulse oximeter measurement CCC descriptor in CCC descriptor handle table.

5.18.9 PlxpsMeasStop()

Stop periodic pulse oximiter measurement.

Syntax:

```
void PlxpsMeasStop(void)
```

5.19 Runners Speed and Cadence Profile Sensor

5.19.1 RscpsSetParameter()

Set a running speed measurement parameter.

Syntax:

```
void RscpsSetParameter(uint8_t type, uint32_t value)
```

Where:

- `type`: Parameter identifier.
- `value`: Measurement value.

5.19.2 RscpsSetFeatures()

Set the features attribute.

Syntax:

```
void RscpsSetFeatures(uint16_t features)
```

Where:

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

- features: Features bitmask.

5.19.3 RscpsSendSpeedMeasurement()

Notifies the collector of a Running Speed and Cadence Measurement.

Syntax:

```
void RscpsSendSpeedMeasurement(dmConnId_t connId)
```

Where:

- connId: Features bitmask.

5.20 Scan Parameter Profile Server

5.20.1 ScppsRegisterCback()

Called to register an application scan interval window callback function.

Syntax:

```
void ScppsRegisterCback(ScppsAppCback_t *cback)
```

5.20.2 ScppsAttsWriteCback()

Called to register an application scan interval window callback function.

Syntax:

```
uint8_t ScppsAttsWriteCback(dmConnId_t connId, uint16_t handle, uint8_t
          operation, uint16_t offset, uint16_t len, uint8_t *pValue, attsAttr_t
          *pAttr)
```

5.21 Time Profile Client

5.21.1 Handle Index Enumeration

Enumeration of handle indexes of characteristics to be discovered. These indexes give the location of their respective characteristic handles in the handle list returned from service and characteristic discovery.

Table 19 Time profile handle index enumeration

Name	Description
TIPC_CTS_CT_HDL_IDX	Current time.
TIPC_CTS_CT_CCC_HDL_IDX	Current time client characteristic configuration descriptor.
TIPC_CTS_LTI_HDL_IDX	Local time information.
TIPC_CTS_RTI_HDL_IDX	Reference time information.
TIPC_CTS_HDL_LIST_LEN	Handle list length.

5.21.2 TipcCtsDiscover()

Perform service and characteristic discovery for Current Time service. Parameter pHdlList must point to an array of length TIPC_CTS_HDL_LIST_LEN. If discovery is successful the handles of discovered

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

characteristics and descriptors will be set in pHdlList.

Syntax:

```
void TipcCtsDiscover(dmConnId_t connId, uint16_t *pHdlList)
```

Where:

- connId: Connection identifier.
- pHdlList: Characteristic handle list.

5.21.3 TipcCtsValueUpdate()

Process a value received in an ATT read response, notification, or indication message. Parameter pHdlList must point to an array of length TIPC_CTS_HDL_LIST_LEN. If the attribute handle of the message matches a handle in the handle list the value is processed, otherwise it is ignored.

Syntax:

```
uint8_t TipcCtsValueUpdate(uint16_t *pHdlList, attEvt_t *pMsg)
```

Where:

- Where:pHdlList: Characteristic handle list.
- pMsg: ATT callback message.

Return ATT_SUCCESS if handle is found, ATT_ERR_NOT_FOUND otherwise.

5.22 Weight Scale Profile Client

5.22.1 WspcWssDiscover()

Perform service and characteristic discovery for Weight Scale service. Parameter pHdlList must point to an array of length WSPC_WSS_HDL_LIST_LEN. If discovery is successful the handles of discovered characteristics and descriptors will be set in pHdlList.

Syntax:

```
void WspcWssDiscover(dmConnId_t connId, uint16_t *pHdlList)
```

- Where:connId: Connection identifier.
- pHdlList: Characteristic handle list.

5.22.2 WspcWssValueUpdate()

Process a value received in an ATT read response, notification, or indication message.

Syntax:

```
uint8_t WspcWssValueUpdate(uint16_t *pHdlList, attEvt_t *pMsg)
```

Where:

- pHdlList: Characteristic handle list.
- pMsg: ATT callback message.

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

Parameter pHdlList must point to an array of length WSPC_WSS_HDL_LIST_LEN. If the ATT handle of the message matches a handle in the handle list the value is processed, otherwise it is ignored.

Return ATT_SUCCESS if handle is found, ATT_ERR_NOT_FOUND otherwise.

5.23 Weight Scale Profile Sensor

5.23.1 WspsMeasComplete()

Weight scale measurement complete.

Syntax:

```
void WspsMeasComplete(dmConnId_t connId, uint8_t wsmCccIdx)
```

Where:

- connId: Connection identifier.
- wsmCccIdx: Index of weight scale measurement CCC descriptor in CCC descriptor handle table.

5.23.2 WspsSetWsmFlags()

Set the weight scale measurement flags.

Syntax:

```
void WspsSetWsmFlags(uint8_t flags)
```

Where:

- flags: Weight scale measurement flags.

5.24 Cordio Proprietary Profile Client

5.24.1 Handle Index Enumeration

Enumeration of handle indexes of characteristics to be discovered. These indexes give the location of their respective characteristic handles in the handle list returned from service and characteristic discovery.

Table 20 Handle index enumeration

Name	Description
WPC_P1_DAT_HDL_IDX	Proprietary data characteristic.
WPC_P1_NA_CCC_HDL_IDX	Proprietary data client characteristic configuration descriptor.
WPC_P1_HDL_LIST_LEN	Handle list length.

5.24.2 WpcP1Discover()

Perform service and characteristic discovery for Cordio proprietary service P1. Parameter pHdlList must point to an array of length WPC_P1_HDL_LIST_LEN. If discovery is successful the handles of discovered characteristics and descriptors will be set in pHdlList.

Syntax:

Copyright © 2011-2016 ARM. All rights reserved.

Confidential

```
void WpcP1Discover(dmConnId_t connId, uint16_t *pHdlList)
```

Where:

- `connId`: Connection identifier.
- `pHdlList`: Characteristic handle list

Copyright © 2011-2016 ARM. All rights reserved.

Confidential