Profile and Service API

Document 2012-0021
Revision 1.3
September 30, 2015

Table of Contents

1 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.

1.1 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)
- Device Information Service (DIS)
- Glucose Service (GLS)
- Heart Rate Service (HRS)
- Health Thermometer Service (HTS)
- HID Service (HIDS)
- Immediate Alert Service (IAS)
- Link Loss Service (LLS)
- TX Power Service (TPS)
- Weight Scale Service (WSS)

The following standard profiles are supported:

- Alert Notification Profile (ANP)
- Blood Pressure Profile (BLP)
- 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)
- Time Profile (TIP)
- Weight Scale Profile (WSP)

2 Service API

Services are divided into separate modules as follows:

Service	Interface file

Battery Service	svc_batt.h
Blood Pressure Service	svc_bps.h
Device Information Service	svc_dis.h
Glucose Service	svc_gls.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
GAP Service GATT Service	svc_core.h
Wicentric Proprietary Service	svc_wp.h
Weight Scale Service	svc_wss.h

2.1 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.

2.1.1 void SvcAddGroup(void)

Add the attribute group for the service to the attribute database. This function is typically called once at system startup to set up and initialize a service.

2.1.2 void SvcRemoveGroup(void)

Remove the attribute group for the service from the attribute database. The service will no longer be available to peer devices.

2.1.3 void SvcCbackRegister(attsReadCback_t readCback, attsWriteCback_t writeCback)

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.

- **readCback**: Read callback. See [2].
- **writeCback**: Write callback. See [2].

3 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**.

3.1 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.

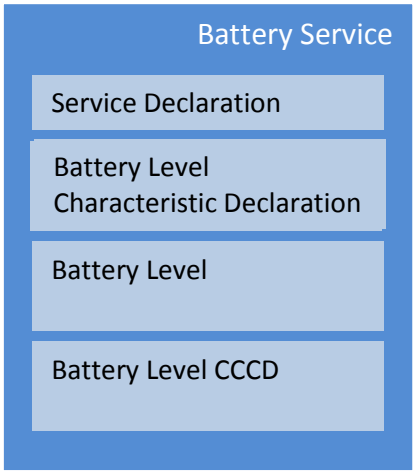

Figure 1. Battery service attributes.

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).

3.2 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 overalap 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
};
```

3.3 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.

```
/*! 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**.

3.4 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.

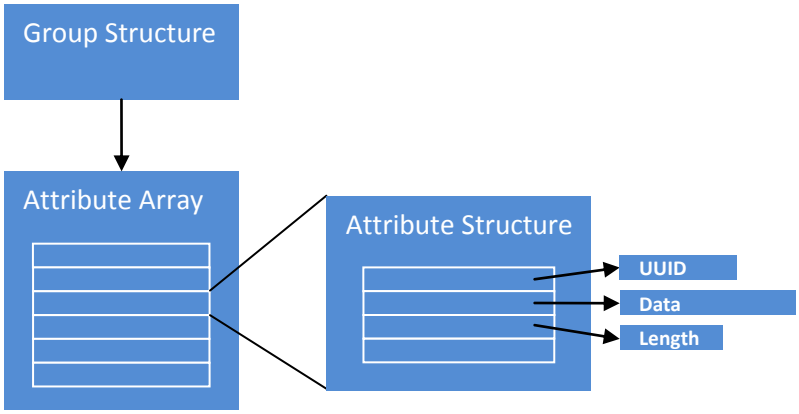

Figure 2. Service data structures.

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.

3.4.1 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 */
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.

3.4.2 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,
    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.

3.4.3 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
};
```

4 Profile API

4.1 Alert Notification Profile Client

4.1.1 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.

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

4.1.2 void AnpcAnsDiscover(dmConnId_t connId, uint16_t *pHdlList)

Perform service and characteristic discovery for Alert Notification service. 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.

- **connId**: Connection identifier. See [1].
- **pHdlList**: Characteristic handle list.

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

Send a command to the alert notification control point.

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

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

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

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

Returns ATT_SUCCESS if handle is found, ATT_ERR_NOT_FOUND otherwise.

4.2 Battery Service Server

4.2.1 basCfg_t

Battery server configurable 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.

4.2.2 void BasInit(wsfHandlerId_t handlerId, basCfg_t *pCfg)

Initialize the battery service server.

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

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

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

- **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.

4.2.4 void BasMeasBattStop(void)

Stop periodic battery level measurement.

4.2.5 void BasProcMsg(wsfMsgHdr_t *pMsg)

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

- **pMsg**: Event message.

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

Send the battery level to the peer device.

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

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

ATTS read callback for battery service used to read the battery level. Use this function as a parameter to SvcBattCbackRegister().

4.3 Blood Pressure Profile Client

4.3.1 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.

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.

4.3.2 void BlpcBpsDiscover(dmConnId_t connId, uint16_t *pHdlList)

Perform service and characteristic discovery for Blood Pressure service. 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.

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

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

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

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

Returns ATT_SUCCESS if handle is found, ATT_ERR_NOT_FOUND otherwise.

4.4 Blood Pressure Profile Sensor

4.4.1 blpsCfg_t

Blood pressure sensor configurable parameters.

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

4.4.2 void BlpsInit(wsfHandlerId_t handlerId, blpsCfg_t *pCfg)

Initialize the Blood Pressure profile sensor.

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

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

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

- **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.

4.4.4 void BlpsMeasStop(void)

Stop periodic blood pressure measurement.

4.4.5 void BlpsMeasComplete(dmConnId_t connId, uint8_t bpmCccIdx)

Blood pressure measurement complete.

- **connId**: DM connection identifier.
- **bpmCccIdx**: Index of blood pressure measurement CCC descriptor in CCC descriptor handle table.

4.4.6 void BlpsProcMsg(wsfMsgHdr_t *pMsg)

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

- **pMsg**: Event message.

4.4.7 void BlpsSetBpmFlags(uint8_t flags)

Set the blood pressure measurement flags.

- **flags**: Blood pressure measurement flags.

4.4.8 void BlpsSetIcpFlags(uint8_t flags)

Set the intermediate cuff pressure flags.

- **flags**: Intermediate cuff pressure flags.

4.5 Device Information Service Client

4.5.1 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.

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

4.5.2 void DisDiscover(dmConnId_t connId, uint16_t *pHdlList)

Perform service and characteristic discovery for DIS service. 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.

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

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

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

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

Returns ATT_SUCCESS if handle is found, ATT_ERR_NOT_FOUND otherwise.

4.6 Find Me Profile Locator

4.6.1 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.

Name	Description
FMPL_IAS_AL_HDL_IDX	Alert level .
FMPL_IAS_HDL_LIST_LEN	Handle list length.

4.6.2 void FmplIasDiscover(dmConnId_t connId, uint16_t *pHdlList)

Perform service and characteristic discovery for Immediate Alert service. 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.

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

4.6.3 void FmplSendAlert(dmConnId_t connId, uint16_t handle, uint8_t alert)

Send an immediate alert to the peer device.

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

4.7 GATT Client

4.7.1 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.

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.

4.7.2 void GattDiscover(dmConnId_t connId, uint16_t *pHdlList)

Perform service and characteristic discovery for GATT service. 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.

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

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

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

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

Returns ATT_SUCCESS if handle is found, ATT_ERR_NOT_FOUND otherwise.

4.8 Glucose Profile Client

4.8.1 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.

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.

4.8.2 glpcFilter_t

Glucose service RACP filter type.

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

4.8.3 void GlpcGlsDiscover(dmConnId_t connId, uint16_t *pHdlList)

Perform service and characteristic discovery for Glucose service. 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.

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

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

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

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

Return ATT_SUCCESS if handle is found, ATT_ERR_NOT_FOUND otherwise.

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

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

- **connId**: Connection identifier.
- **handle**: Attribute handle.
- **opcode**: Command opcode.
- **oper**: Command operator or 0 if no operator required.
- **pFliter**: Command filter parameters or NULL of no parameters required.

4.8.6 void GlpcGlsSetLastSeqNum(uint16_t seqNum)

Set the last received glucose measurement sequence number.

- **seqNum**: Glucose measurement sequence number.

4.8.7 uint16_t GlpcGlsGetLastSeqNum(void)

Return the last received glucose measurement sequence number.

4.9 Glucose Profile Sensor

4.9.1 void GlpsInit(void)

Initialize the Glucose profile sensor.

4.9.2 void GlpsProcMsg(wsfMsgHdr_t *pMsg)

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

- **pMsg**: Event message.

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

ATTS write callback for glucose service record access control point. Use this function as a parameter to SvcGlsCbackRegister().

4.9.4 void GlpsSetFeature(uint16_t feature)

Set the supported features of the glucose sensor.

- **feature**: Feature bitmask.

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

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

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

4.10 Heart Rate Profile Client

4.10.1 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.

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.

4.10.2 void HrpcHrsDiscover(dmConnId_t connId, uint16_t *pHdlList)

Perform service and characteristic discovery for Heart Rate service. 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.

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

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

Send a command to the heart rate control point.

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

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

Process a value received in an ATT read response, notification, or indication message. Parameter pHdlList must point to an array of length HRPC_HRS_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.

- **connId**: Connection identifier.

- **pMsg**: ATT callback message.

4.11 Heart Rate Profile Sensor

4.11.1 hrpsCfg_t

Heart rate service configurable parameters.

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

4.11.2 void HrpsInit(wsfHandlerId_t handlerId, hrpsCfg_t *pCfg)

Initialize the Heart Rate profile sensor.

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

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

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

- **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.

4.11.4 void HrpsMeasStop(void)

Stop periodic heart rate measurement.

4.11.5 void HrpsProcMsg(wsfMsgHdr_t *pMsg)

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

- **pMsg**: Event message.

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

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

4.11.7 void HrpsSetFlags(uint8_t flags)

Set the heart rate measurement flags.

- **flags**: Heart rate measurement flags.

4.12 Health Thermometer Profile Client

4.12.1 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.

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.

4.12.2 void HtpcHtsDiscover(dmConnId_t connId, uint16_t *pHdlList)

Perform service and characteristic discovery for Health Thermometer service. 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.

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

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

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

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

Returns ATT_SUCCESS if handle is found, ATT_ERR_NOT_FOUND otherwise.

4.13 Health Thermometer Profile Sensor

4.13.1 htpsCfg_t

Health Thermometer profile sensor configurable parameters.

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

4.13.2 void HtpsInit(wsfHandlerId_t handlerId, htpsCfg_t *pCfg)

Initialize the Health Thermometer profile sensor.

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

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

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

- **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.

4.13.4 void HtpsMeasStop(void)

Stop periodic temperature measurement.

4.13.5 void HtpsMeasComplete(dmConnId_t connId, uint8_t tmCccIdx)

Temperature measurement complete.

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

4.13.6 void HtpsProcMsg(wsfMsgHdr_t *pMsg)

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

- **pMsg**: Event message.

4.13.7 void HtpsSetTmFlags(uint8_t flags)

Set the temperature measurement flags.

- **flags**: Temperature measurement flags.

4.13.8 void HtpsSetItFlags(uint8_t flags)

Set the intermediate temperature flags.

- **flags**: Intermediate temperature flags.

4.14 Phone Alert Status Profile Client

4.14.1 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.

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.

4.14.2 void PaspcPassDiscover(dmConnId_t connId, uint16_t *pHdlList)

Perform service and characteristic discovery for Phone Alert Status service. 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.

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

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

Send a command to the ringer control point.

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

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

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

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

Returns ATT_SUCCESS if handle is found, ATT_ERR_NOT_FOUND otherwise.

4.15 Time Profile Client

4.15.1 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.

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.

4.15.2 void TipcCtsDiscover(dmConnId_t connId, uint16_t *pHdlList)

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 characteristics and descriptors will be set in pHdlList.

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

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

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.

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

Return ATT_SUCCESS if handle is found, ATT_ERR_NOT_FOUND otherwise.

4.16 Weight Scale Profile Client

4.16.1 void WspcWssDiscover(dmConnId_t connId, uint16_t *pHdlList)

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.

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

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

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

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

Return ATT_SUCCESS if handle is found, ATT_ERR_NOT_FOUND otherwise.

4.17 Weight Scale Profile Sensor

4.17.1 void WspsMeasComplete(dmConnId_t connId, uint8_t wsmCccIdx)

Weight scale measurement complete.

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

4.17.2 void WspsSetWsmFlags(uint8_t flags)

Set the weight scale measurement flags.

- **flags**: Weight scale measurement flags.

4.18 Wicentric Proprietary Profile Client

4.18.1 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.

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.

4.18.2 WpcP1Discover(dmConnId_t connId, uint16_t *pHdlList)

Perform service and characteristic discovery for Wicentric 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.

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

4.19 HID Device Profile

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

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

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

- **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.

4.19.2 void HidSetProtocolMode(uint8_t protocolMode)

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

- **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.

4.19.3 uint8_t HidGetProtocolMode()

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

4.19.4 uint8_t 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.

4.19.5 void HidInit(const hidConfig_t *pConfig)

The HidInit function is used to initialize the HID profile.

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

4.19.6 Callback Functions

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

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

- **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.

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

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

- **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.

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

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

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

5 References

1. Wicentric, “Device Manager API”, 2009-0008.
2. Wicentric, “Attribute Protocol API”, 2009-0010.

6 Definitions

ATT	Attribute protocol software subsystem
CCC or CCCD	Client characteristic configuration (descriptor)
DM	Device Manager software subsystem
HCI	Host Controller Interface
LE	(Bluetooth) Low Energy
LTK	Long Term Key
WSF	Wicentric Software Foundation software service and porting layer