wicentric

App Framework API

Document 2011-0020
Revision 1.4
September 25, 2015

Table of Contents

1 Introduction

This document describes the API of the App Framework subsystem. The App Framework is a service layer for applications that simplifies application development.

1.1 Overview

The App Framework performs many operations common to Bluetooth LE embedded applications, such as:

- Application-level device, connection, and security management.
- Simple user interface abstractions for button press handling, sounds, display, and other user feedback.
- An abstracted device database for storing bonding data and other device parameters.

The relationship between the App Framework, the application, and the protocol stack is shown in Figure 1.

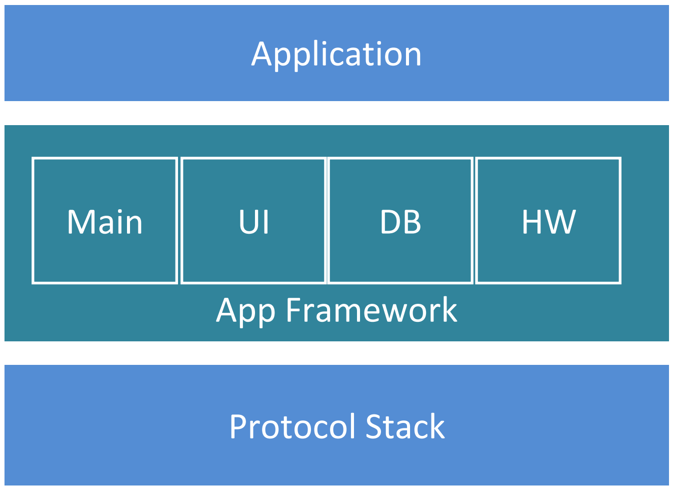

Figure 1. App Framework software system diagram.

1.2 Modules

The App Framework consists of several modules, each with their own API interface file.

Module	Interface file	Description
Main	app_api.h	Device, connection, and security management.
UI	app_ui.h	User interface abstraction.
DB	app_db.h	Device database.
HW	app_hw.h	Hardware sensor interface abstraction.

The Main module is designed to be platform-independent while the UI and DB modules are designed with platform-independent APIs and platform-specific implementations.

2 Main Interface

2.1 Constants and Data Types

2.1.1 Discoverable/Connectable Mode

Discoverable/connectable mode used by function AppAdvStart().

Name	Description
APP_MODE_CONNECTABLE	Connectable mode.
APP_MODE_DISCOVERABLE	Discoverable mode.
APP_MODE_AUTO_INIT	Automatically configure mode based on bonding info.

2.1.2 Advertising and Scan Data Storage Locations

Advertising and scan data storage locations.

Name	Description
APP_ADV_DATA_CONNECTABLE	Advertising data for connectable mode.
APP_SCAN_DATA_CONNECTABLE	Scan data for connectable mode.
APP_ADV_DATA_DISCOVERABLE	Advertising data for discoverable mode.
APP_SCAN_DATA_DISCOVERABLE	Scan data for discoverable mode.

2.1.3 Service Discovery and Configuration Client Status

Service discovery and configuration client status.

Name	Description
APP_DISC_INIT	No discovery or configuration complete.
APP_DISC_SEC_REQUIRED	Security required to complete configuration.
APP_DISC_START	Service discovery started.
APP_DISC_CMPL	Service discovery complete.
APP_DISC_FAILED	Service discovery failed.
APP_DISC_CFG_START	Service configuration started.
APP_DISC_CFG_CONN_START	Configuration for connection setup started.
APP_DISC_CFG_CMPL	Service configuration complete.

2.1.4 appSlaveCfg_t

Configurable parameters for slave.

Type	Name	Description
uint16_t	advDuration[]	Advertising durations in ms.
uint16_t	advInterval[]	Advertising intervals 0.625 ms units.

2.1.5 appMasterCfg_t

Configurable parameters for master.

Type	Name	Description
uint16_t	scanInterval	The scan interval, in 0.625 ms units.
uint16_t	scanWindow	The scan window, in 0.625 ms units. Must be less than or equal to scan interval.
uint16_t	scanDuration	The scan duration in ms. Set to zero to scan until stopped.
uint8_t	discMode	The GAP discovery mode (general, limited, or none).
uint8_t	scanType	The scan type (active or passive).

2.1.6 appSecCfg_t

Configurable parameters for security.

Type	Name	Description
uint8_t	auth	Authentication and bonding flags.
uint8_t	iKeyDist	Initiator key distribution flags.
uint8_t	rKeyDist	Responder key distribution flags.
bool_t	oob	TRUE if out-of-band pairing data is present.
bool_t	initiateSec	TRUE to initiate security upon connection.

2.1.7 appUpdateCfg_t

Configurable parameters for connection parameter update.

Type	Name	Description
wsfTimerTicks_t	idlePeriod	Connection idle period in ms before attempting connection parameter update; set to zero to disable.
uint16_t	connIntervalMin	Minimum connection interval in 1.25ms units.
uint16_t	connIntervalMax	Maximum connection interval in 1.25ms units.
uint16_t	connLatency	Connection latency.
uint16_t	supTimeout	Supervision timeout in 10ms units.
uint8_t	maxAttempts	Number of update attempts before giving up.

2.1.8 appDevInfo_t

Device information data type.

Type	Name	Description
bdAddr_t	addr	Peer device address.
uint8_t	addrType	Peer address type.

2.2 Global Variables

2.2.1 pAppSlaveCfg

This is a pointer to the slave configurable parameters used by the application. If slave mode is used, the application must set this variable during system initialization.

2.2.2 pAppMasterCfg

This is a pointer to the master configurable parameters used by the application. If master mode is used, the application must set this variable during system initialization.

2.2.3 pAppSecCfg

This is a pointer to the security-related configurable parameters used by the application. The application must set this variable during system initialization.

2.2.4 pAppUpdateCfg

This is a pointer to the connection parameter update parameters used by the application. The application must set this variable during system initialization.

2.3 Initialization Functions

2.3.1 void AppSlaveInit(void)

Initialize the App Framework for operation as a Bluetooth LE slave. This function is generally called once during system initialization before any other App Framework API functions are called.

2.3.2 void AppMasterInit(void)

Initialize the App Framework for operation as a Bluetooth LE master. This function is generally called once during system initialization before any other App Framework API functions are called.

2.4 Advertising Functions

2.4.1 void AppAdvSetData(uint8_t location, uint8_t len, uint8_t *pData)

Set advertising or scan data. Separate advertising and scan data can be set for connectable and discoverable modes. The application must allocate and maintain the memory pointed to by pData while the device is advertising.

- **location**: Data location. See 2.1.2.
- **len**: Length of the data. Maximum length is 31 bytes.
- **pData**: Pointer to the data.

2.4.2 void AppAdvStart(uint8_t mode)

Start advertising using the parameters for the given mode.

- **mode**: Discoverable/connectable mode. 2.1.1.

2.4.3 void AppAdvStop(void)

Stop advertising. The device will no longer be connectable or discoverable.

2.4.4 bool_t AppAdvSetAdValue(uint8_t location, uint8_t adType, uint8_t len, uint8_t *pValue)

Set the value of an advertising data element in the advertising or scan response data. If the element already exists in the data then it is replaced with the new value. If the element does not exist in the data it is appended to it, space permitting.

There is special handling for the device name (AD type DM_ADV_TYPE_LOCAL_NAME). If the name can only fit in the data if it is shortened, the name is shortened and the AD type is changed to DM_ADV_TYPE_SHORT_NAME.

- **location:** Data location.
- **adType**: Advertising data element type.
- **len**: Length of the value. Maximum length is 29 bytes.
- **pValue**: Pointer to the value.

Return TRUE if the element was successfully added to the data, FALSE otherwise.

2.4.5 void AppSetAdvType(uint8_t advType)

Set the advertising type, which can be DM_ADV_CONN_UNDIRECT, DM_ADV_DISC_UNDIRECT, or DM_ADV_NONCONN_UNDIRECT.

- **advType**: Advertising type.

2.5 Scanning Functions

2.5.1 void AppScanStart(uint8_t mode, uint8_t scanType, uint16_t duration)

This function is called to start scanning. A scan is performed using the given discoverability mode, scan type, and duration.

- **mode**: Discoverability mode. See [1].
- **scanType**: Scan type. See [1].
- **duration**: The scan duration, in milliseconds. If set to zero, scanning will continue until AppScanStop() is called.

2.5.2 void AppScanStop(void)

This function is called to stop scanning.

2.5.3 appDevInfo_t *AppScanGetResult(uint8_t idx)

Get a stored scan result from the scan result list. The first result is at index zero.

- **idx**: Index of result in scan result list.

This function returns a pointer to the scan result device information or NULL if the index contains no result.

2.5.4 uint8_t AppScanGetNumResults(void)
Get the number of stored scan results.

2.6 Connection and Security Functions

2.6.1 void AppConnClose(dmConnId_t connId)
Close a connection with the given connection identifier.

- **connId**: Connection identifier. See [1].

2.6.2 dmConnId_t AppConnIsOpen(void)
Check if a connection is open. This function returns the connection identifier of the open connection. If operating as a master with multiple simultaneous connections, the returned connection identifier is for the first open connection found.

2.6.3 void AppHandlePasskey(dmSecAuthReqIndEvt_t *pAuthReq)
Handle a passkey request during pairing. If the passkey is to be displayed, a random passkey is generated and displayed. If the passkey is to be entered the user is prompted to enter the passkey.

- **pAuthReq**: DM authentication requested event structure. See [1].

2.6.4 void AppSetBondable(bool_t bondable)
Set the bondable mode of the device. When a device is in bondable mode it can pair with a peer device and store the keys exchanged during pairing.

- **bondable**: TRUE to set device to bondable, FALSE to set to non-bondable.

2.6.5 void AppSlaveSecurityReq(dmConnId_t connId)
Initiate a request for security as a slave device. This function will send a message to the master peer device requesting security. The master device should either initiate encryption or pairing.

- **connId**: Connection identifier. See [1].

2.6.6 void AppMasterSecurityReq(dmConnId_t connId)
Initiate security as a master device. If there is a stored encryption key for the peer device this function will initiate encryption, otherwise it will initiate pairing.

- **connId**: Connection identifier. See [1].

2.7 Discovery Functions

2.7.1 void AppDiscInit(void)
Initialize app framework discovery. This function is generally called once during system initialization before any other App Framework API functions are called.

2.7.2 void AppDiscRegister(appDiscCback_t cback)

Register a callback function to service discovery status.

- **cback**: Application service discovery callback function.

2.7.3 void AppDiscSetHdlList(dmConnId_t connId, uint8_t hdlListLen, uint16_t *pHdlList)

Set the discovery cached handle list for a given connection.

- **connId**: Connection identifier. See [1].
- **listLen**: Length of characteristic and handle lists.
- **pHdlList**: Characteristic handle list.

2.7.4 void AppDiscComplete(dmConnId_t connId, uint8_t status)

- **connId**: Connection identifier. See [1].
- **status**: Service or configuration status. See 2.1.3.

2.7.5 void AppDiscFindService(dmConnId_t connId, uint8_t uuidLen, uint8_t *pUuid, uint8_t listLen, attcDiscChar_t **pCharList, uint16_t *pHdlList)

Perform service and characteristic discovery for a given service. Parameter pUuid points to the UUID of the service to discover. Parameter pCharList contains the list of characteristics and descriptors to discover. Parameter pHdlList points to memory allocated by the application for storing the handles of discovered characteristics and descriptors. Handles are stored at the same index in pHdlList as the index of their respective characteristics in pCharList.

- **connId**: Connection identifier.
- **uuidLen**: Length of service UUID (2 or 16).
- **pUuid**: Pointer to service UUID.
- **listLen**: Length of characteristic and handle lists.
- **pCharList**: Characterisic list for discovery.
- **pHdlList**: Characteristic handle list.

2.7.6 void AppDiscConfigure(dmConnId_t connId, uint8_t status, uint8_t cfgListLen, attcDiscCfg_t *pCfgList, uint8_t hdlListLen, uint16_t *pHdlList)

Configure characteristics for discovered services. Parameter pCfgList points to a list of characteristic information used to read or write a set of characteristics. Parameter pHdlList contains the handles of the characteristics. Each entry in pCfgList contains a handle index that maps to the position of the characteristic's handle in pHdlList.

- **connId**: Connection identifier.
- **status**: Set to APP_DISC_CFG_START if configuration is being performed after service discovery or APP_DISC_CFG_CONN_START if configuration is being performed on connection setup.
- **cfgListLen**: Length of characteristic configuration list.
- **pCfgList**: Characteristic configuration list.
- **hdlListLen**: Length of characteristic handle list.

- **pHdlList**: Characteristic handle list.

2.7.7 AppDiscServiceChanged(attEvt_t *pMsg)

Perform the GATT service changed procedure. This function is called when an indication is received containing the GATT service changed characteristic. This function may initialize the discovery state and initiate service discovery and configuration.

- **pMsg**: Pointer to ATT callback event message containing received indication.

2.7.8 void AppDiscProcDmMsg(dmEvt_t *pMsg)

Process discovery-related DM messages. This function should be called from the application's event handler.

- **pMsg**: Pointer to DM callback event message. See [1].

2.7.9 void AppDiscProcAttMsg(attEvt_t *pMsg)

Process discovery-related ATT messages. This function should be called from the application's event handler.

- **pMsg**: Pointer to ATT callback event message. See [2].

2.8 Message Processing Functions

2.8.1 void AppSlaveProcDmMsg(dmEvt_t *pMsg)

Process connection-related DM messages for a slave. This function should be called from the application's event handler.

- **pMsg**: Pointer to DM callback event message. See [1].

2.8.2 void AppSlaveSecProcDmMsg(dmEvt_t *pMsg)

Process security-related DM messages for a slave. This function should be called from the application's event handler.

- **pMsg**: Pointer to DM callback event message. See [1].

2.8.3 void AppMasterProcDmMsg(dmEvt_t *pMsg)

Process connection-related DM messages for a master. This function should be called from the application's event handler.

- **pMsg**: Pointer to DM callback event message. See [1].

2.8.4 void AppMasterSecProcDmMsg(dmEvt_t *pMsg)

Process security-related DM messages for a master. This function should be called from the application's event handler.

- **pMsg**: Pointer to DM callback event message. See [1].

2.8.5 void AppServerConnCback(dmEvt_t *pDmEvt)

ATT connection callback for app framework. This function is used when the application is operating as an ATT server and it uses notifications or indications. This function can be called by the application's ATT connection callback or it can be installed as the ATT connection callback.

- **pDmEvt**: Pointer to DM callback event message. See [1].

2.9 Callback Interface

2.9.1 void (*appDiscCback_t)(dmConnId_t connId, uint8_t status)

Service discovery and configuration callback.

- **connId**: Connection identifier. See [1].
- **status**: Service or configuration status.

3 DB Interface

The DB interface provides an abstracted device database for storing bonding data and other device parameters. The DB interface is used internally by the App Framework t manage bonding data and client characteristic configuration descriptors. The interface can also be used by the application.

3.1 Constants and Data Types

3.1.1 appDbHdl_t

Device database record handle type. Each record in the device database is accessed via a unique handle.

3.1.2 APP_DB_HDL_NONE

No device database record handle. This special value for the record handle is typically used to indicate an error or that no record was found.

3.2 Functions

3.2.1 void AppDbInit(void)

Initialize the device database. This function is typically called once at system startup.

3.2.2 appDbHdl_t AppDbNewRecord(uint8_t addrType, uint8_t *pAddr)

Create a new device database record. This function is typically called when bonding begins.

- **addrType**: Address type. See [1].
- **pAddr**: Peer device address.

This function returns the database record handle of the new record.

3.2.3 void AppDbDeleteRecord(appDbHdl_t hdl)
Delete a new device database record. This function is called if bonding fails or if the application desired to remove a bond.

- **hdl**: Database record handle.

3.2.4 void AppDbValidateRecord(appDbHdl_t hdl, uint8_t keyMask)
Validate a new device database record. This function is called when pairing is successful and the devices are bonded.

- **hdl**: Database record handle.
- **keyMask**: Bitmask of keys to validate.

3.2.5 void AppDbCheckValidRecord(appDbHdl_t hdl)
Check if a record has been validated. If it has not, delete it. This function is typically called when the connection is closed.

- **hdl**: Database record handle.

3.2.6 bool_t AppDbCheckBonded(void)
Check if there is a stored bond with any device.

This function returns TRUE if a bonded device is found, FALSE otherwise.

3.2.7 void AppDbDeleteAllRecords(void)
Delete all database records.

3.2.8 appDbHdl_t AppDbFindByAddr(uint8_t addrType, uint8_t *pAddr)
Find a device database record by peer address.

- **addrType**: Address type. See [1].
- **pAddr**: Peer device address.

This function returns the database record handle or APP_DB_HDL_NONE if not found.

3.2.9 appDbHdl_t AppDbFindByLtkReq(uint16_t encDiversifier, uint8_t *pRandNum)
Find a device database record from data in an LTK request. The App Framework calls this function when operating as a slave device and the master requests to enable encryption with the LTK.

3.2.10 appDbHdl_t AppDbGetHdl(dmConnId_t connId)
Get the device database record handle associated with an open connection.

- **connId**: Connection identifier. See [1].

This function returns the database record handle or APP_DB_HDL_NONE.

3.2.11 dmSecKey_t *AppDbGetKey(appDbHdl_t hdl, uint8_t type, uint8_t *pSecLevel)

Get a key from a device database record. The App Framework calls this function to retrieve the LTK when encryption is enabled.

- **hdl**: Database record handle.
- **type**: Type of key to get. See [1].
- **pSecLeve**l: If the key is valid, returns the security level of the key. See [1].

This function returns a pointer to the key if the key is valid or NULL if not valid.

3.2.12 void AppDbSetKey(appDbHdl_t hdl, dmSecKeyIndEvt_t *pKey)

Set a key in a device database record. The App Framework calls this function to store a key received during pairing.

- **hdl**: Database record handle.
- **pKey**: Key data. See [1].

3.2.13 uint16_t *AppDbGetCccTbl(appDbHdl_t hdl)

Get the client characteristic configuration descriptor table. This table contains a peer device's stored settings for indications and notifications.

- **hdl**: Database record handle.

This function returns a pointer to client characteristic configuration descriptor table.

3.2.14 void AppDbSetCccTblValue(appDbHdl_t hdl, uint16_t idx, uint16_t value)

Set a value in the client characteristic configuration table. This function is typically called from the application's ATT client characteristic configuration callback to store a new value when it is written by the peer device.

- **hdl**: Database record handle.
- **idx**: Table index. See [2].
- **value**: Client characteristic configuration value. See [2].

3.2.15 uint8_t AppDbGetDiscStatus(appDbHdl_t hdl)

Get the discovery status.

- **hdl**: Database record handle.

This function returns the discovery status.

3.2.16 void AppDbSetDiscStatus(appDbHdl_t hdl, uint8_t status)

Set the discovery status.

- **hdl**: Database record handle.
- **status**: The discovery status. See 2.1.3.

3.2.17 uint16_t *AppDbGetHdlList(appDbHdl_t hdl)
Get the cached handle list.

- **hdl**: Database record handle.

This function returns a pointer to the handle list.

3.2.18 void AppDbSetHdlList(appDbHdl_t hdl, uint16_t *pHdlList)
Set the discovery status.

- **hdl**: Database record handle.
- **pHdlList**: Pointer to handle list.

3.2.19 char *AppDbGetDevName(uint8_t *pLen)
Get the device name.

- **pLen**: Returned device name length.

Returns a pointer to a UTF-8 string containing the device name or NULL if not set.

3.2.20 void AppDbSetDevName(uint8_t len, char *pStr)
Set the device name.

- **len**: Device name length.
- **pStr**: UTF-8 string containing the device name.

4 UI Interface
The UI interface provides the application with simple user interface abstractions for button press handling, sounds, display, and other user feedback.

4.1 Constants and Data Types

4.1.1 UI Event Enumeration
The following UI event enumeration values are used by function AppUiAction().

Name	Description
APP_UI_NONE	No event.
APP_UI_RESET_CMPL	Reset complete.
APP_UI_DISCOVERABLE	Enter discoverable mode.
APP_UI_ADV_START	Advertising started.
APP_UI_ADV_STOP	Advertising stopped.
APP_UI_SCAN_START	Scanning started.
APP_UI_SCAN_STOP	Scanning stopped.
APP_UI_SCAN_REPORT	Scan data received from peer device.
APP_UI_CONN_OPEN	Connection opened.

APP_UI_CONN_CLOSE	Connection closed.
APP_UI_SEC_PAIR_CMPL	Pairing completed successfully.
APP_UI_SEC_PAIR_FAIL	Pairing failed or other security failure.
APP_UI_SEC_ENCRYPT	Connection encrypted.
APP_UI_SEC_ENCRYPT_FAIL	Encryption failed.
APP_UI_PASSKEY_PROMPT	Prompt user to enter passkey.
APP_UI_ALERT_CANCEL	Cancel a low or high alert.
APP_UI_ALERT_LOW	Low alert.
APP_UI_ALERT_HIGH	High alert.

4.1.2 Button Press Enumeration

Button press enumeration.

Name	**Description**
APP_UI_BTN_NONE	No button press.
APP_UI_BTN_1_DOWN	Button 1 down press.
APP_UI_BTN_1_SHORT	Button 1 short press.
APP_UI_BTN_1_MED	Button 1 medium press.
APP_UI_BTN_1_LONG	Button 1 long press.
APP_UI_BTN_1_EX_LONG	Button 1 extra long press.
APP_UI_BTN_2_DOWN	Button 2 down press.
APP_UI_BTN_2_SHORT	Button 2 short press.
APP_UI_BTN_2_MED	Button 2 medium press.
APP_UI_BTN_2_LONG	Button 2 long press.
APP_UI_BTN_2_EX_LONG	Button 2 extra long press.

4.1.3 LED Values

LED values.

Name	**Description**
APP_UI_LED_NONE	No LED.
APP_UI_LED_1	LED 1.
APP_UI_LED_2	LED 2.
APP_UI_LED_3	LED 3.
APP_UI_LED_4	LED 4.
APP_UI_LED_WRAP	Wrap to beginning of sequence.

4.1.4 Sound Tone Values

Sound tone values.

Name	**Description**
APP_UI_SOUND_WRAP	Sound tone value for wrap/repeat.

4.1.5 appUiSound_t

This structure is used to create sounds played by function AppUiSoundPlay().

Type	Name	Description
uint16_t	tone	Sound tone in Hz. Use 0 for silence.
uint16_t	duration	Sound duration in milliseconds.

4.1.6 appUiLed_t

This structure is used to create LED flash patterns used with function AppUiLedStart().

Type	Name	Description
uint8_t	led	LED to control.
uint8_t	state	On or off.
uint16_t	duration	Duration in milliseconds.

4.2 Functions

4.2.1 void AppUiAction(uint8_t event)

Perform a user interface action based on the event value passed to the function. The implementation of this function will perform a particular action, such as playing a sound or blinking an LED.

- **event**: User interface event value. See 4.1.1.

4.2.2 void AppUiDisplayPasskey(uint32_t passkey)

Display a passkey. This function is only applicable to devices that can display the six-digit numeric passkey value.

- **passkey**: Passkey to display.

4.2.3 void AppUiDisplayRssi(int8_t rssi)

Display an RSSI value. This function is only applicable to devices that can be in a connection.

- **rssi**: RSSI value to display.

4.2.4 void AppUiBtnRegister(appUiBtnCback_t cback)

Register a callback function to receive button presses.

- **cback**: Application button callback function.

4.2.5 void AppUiSoundPlay(const appUiSound_t *pSound)

Play a sound.

- **pSound**: Pointer to sound tone/duration array. See 4.1.3.

4.2.6 void AppUiSoundStop(void)
Stop the sound that is currently playing.

4.2.7 void AppUiLedStart(const appUiLed_t *pLed)
Start LED blinking.

- **pLed**: Pointer to LED data structure. See 4.1.6.

4.2.8 void AppUiLedStop(void)
Stop LED blinking.

4.3 Callback Interface

4.3.1 void (*appUiBtnCback_t)(uint8_t btn)
This callback function sends button events to the application.

- **btn**: Button press event. See 4.1.2.

5 HW Interface
The HW interface provides an abstraction layer for hardware sensors.

5.1 Constants and Data Types

5.1.1 appHrm_t
Heart rate measurement structure.

Type	**Name**	**Description**
uint16_t	*pRrInterval	Array of RR intervals.
uint8_t	numIntervals	Length of RR interval array.
uint16_t	energyExp	Energy expended value.
uint8_t	heartRate	Heart rate.
uint8_t	flags	Heart rate measurement flags.

5.1.2 appDateTime_t
Date and time structure.

Type	**Name**	**Description**
uint16_t	year	Year.
uint8_t	month	Month.
uint8_t	day	Day.
uint8_t	hour	Hour.
uint8_t	min	Minutes.
uint8_t	sec	Seconds.

5.1.3 appBpm_t

Blood pressure measurement structure.

Type	Name	Description
appDateTime_t	timestamp	Date-time.
uint16_t	systolic	Systolic pressure.
uint16_t	diastolic	Diastolic pressure.
uint16_t	map	Mean arterial pressure.
uint16_t	pulseRate	Pulse rate.
uint16_t	measStatus	Measurement status.
uint8_t	flags	Flags.
uint8_t	userId	User ID.

5.1.4 appWsm_t

Weight scale measurement structure.

Type	Name	Description
appDateTime_t	timestamp	Date-time.
uint32_t	weight	Weight.
uint8_t	flags	Weight measurement flags.

5.1.5 appTm_t

Temperature measurement structure.

Type	Name	Description
appDateTime_t	timestamp	Date-time.
uint32_t	temperature	Temperature.
uint8_t	flags	Flags.
uint8_t	tempType	Temperature type.

5.2 Functions

5.2.1 void AppHwBattRead(uint8_t *pLevel)

Read the battery level. The battery level value returned in pLevel is the percentage of remaining battery capacity (0-100%).

- **pLevel**: Battery level return value.

5.2.2 void AppHwHrmRead(appHrm_t *pHrm)

Perform a heart rate measurement. Return the heart rate along with any RR interval data.

- **pHrm**: Heart rate measurement return value.

5.2.3 void AppHwBpmRead(bool_t intermed, appBpm_t *pBpm)

Perform a blood pressure measurement. Return the measurement data.

- **intermed**: TRUE if this is an intermediate measurement.
- **pBpm**: Blood pressure measurement return value.

5.2.4 void AppHwWsmRead(appWsm_t *pWsm)

Perform a weight scale measurement. Return the measurement data.

- **pWsm**: Weight scale measurement return value.

5.2.5 void AppHwTmRead(bool_t intermed, appWsm_t *pWsm)

Perform a temperature measurement. Return the measurement data.

- **intermed**: TRUE if this is an intermediate measurement.
- **pTm**: Temperature measurement return value.

5.2.6 void AppHwTmSetUnits (uint8_t units)

Set the temperature measurement units.

- **units**: CH_TM_FLAG_UNITS_C or CH_TM_FLAG_UNITS_F.

6 References

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

7 Definitions

ATT	Attribute protocol software subsystem
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