ARM® Cordio Profiles

ARM-EPM-115883 1.0

App Framework API

Confidential

ARM® Cordio App Framework API

Reference Manual

Release Information

The following changes have been made to this book:

Document History

Date	Issue	Confidentiality	Change
25 September 2015	-	Confidential	First Wicentric release for 1.1 as 2011-0020
1 March 2016	A	Confidential	First ARM release for 1.1
24 August 2016	A	Confidential	AUSPEX # / API update

Proprietary Notice

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

Contents

ARM® Cordio Profiles 1

1 Preface 11

1.1 About this book 11

1.1.1 Intended audience 11

1.1.2 Using this book 11

1.1.3 Terms and abbreviations 11

1.1.4 Conventions 12

1.1.5 Additional reading 13

1.2 Feedback 13

1.2.1 Feedback on content 13

2 Introduction 14

2.1 Overview 14

2.2 Modules 14

3 Main Interface 15

3.1 Constants and Data Types 15

3.1.1 Discoverable/connectable mode 15

3.1.2 Advertising and scan data storage locations 15

3.1.3 Service discovery and configuration client status 15

3.1.5 Actions for incoming requests 16

3.1.7 appAdvCfg_t 16

3.1.8 appExtAdvCfg_t 16

3.1.9 appSlaveCfg_t 16

3.1.10 appMasterCfg_t 17

3.1.11 appSecCfg_t 17

3.1.12 appUpdateCfg_t 17

3.1.13 appReqActCfg_t 18
3.1.14 appDiscCfg_t 18
3.1.15 appCfg_t 18
3.1.16 appDevInfo_t 18
3.2 Global Variables 19
3.2.1 pAppAdvCfg 19
3.2.2 pAppExtAdvCfg 19
3.2.3 pAppSlaveCfg 19
3.2.4 pAppMasterCfg 19
3.2.5 pAppSecCfg 19
3.2.6 pAppUpdateCfg 19
3.2.7 pAppDiscCfg 19
3.2.8 pAppCfg 19
3.2.9 pAppMasterReqActCfg 19
3.2.10 pAppSlaveReqActCfg 19
3.3 Initialization Functions 19
3.3.1 AppSlaveInit() 19
3.3.2 AppMasterInit() 20
3.4 Advertising Functions 20
3.4.1 AppAdvSetData() 20
3.4.2 AppAdvStart() 20
3.4.3 AppAdvStop() 20
3.4.4 AppAdvSetAdValue() 21
3.4.5 AppSlaveIsAdvertising() 21
1.3 Advertising Extensions 21
3.4.6 AppExtAdvSetData() 22
3.4.7 AppExtAdvStart() 22
3.4.8 AppExtAdvStop() 22

3.4.9 AppExtAdvSetAdValue() 22
3.5 Scanning Functions 23
3.5.1 AppScanStart() 23
3.5.2 AppScanStop() 23
3.5.3 *AppScanGetResult() 23
3.5.4 AppScanGetNumResults() 24
3.6 Connection and Security Functions 24
3.6.1 AppConnClose() 24
3.6.2 AppConnIsOpen() 24
3.6.3 AppHandlePasskey() 24
3.6.4 AppSetBondable() 25
3.6.5 AppSlaveSecurityReq() 25
3.6.6 AppConnAccept() 25
3.6.7 AppExtConnAccept() 25
3.6.8 AppMasterSecurityReq() 26
3.7 Discovery Functions 26
3.7.1 AppDiscInit() 26
3.7.2 AppDiscRegister() 26
3.7.3 AppDiscSetHdlList() 26
3.7.4 AppDiscComplete() 26
3.7.5 AppDiscFindService() 27
3.7.6 AppDiscConfigure() 27
3.7.7 AppDiscServiceChanged() 28
3.7.8 AppDiscProcDmMsg() 28
3.7.9 AppDiscProcAttMsg() 28
3.8 Message Processing Functions 28
3.8.1 AppSlaveProcDmMsg() 28
3.8.2 AppSlaveSecProcDmMsg() 28

3.8.3 AppMasterProcDmMsg() 29

3.8.4 AppMasterSecProcDmMsg() 29

3.8.5 AppServerConnCback() 29

3.9 Callback Interface 29

*3.9.1 (*appDiscCback_t)() 29*

4 DB Interface 30

4.1 Constants and Data Types 30

4.1.1 appDbHdl_t 30

4.1.2 APP_DB_HDL_NONE 30

4.2 Functions 30

4.2.1 AppDbInit() 30

4.2.2 AppDbNewRecord() 30

4.2.3 AppDbDeleteRecord() 30

4.2.4 AppDbValidateRecord() 31

4.2.5 AppDbCheckValidRecord() 31

4.2.6 AppDbCheckBonded() 31

4.2.7 AppDbDeleteAllRecords() 31

4.2.8 AppDbFindByAddr() 31

4.2.9 AppDbFindByLtkReq() 32

4.2.10 AppDbGetHdl() 32

*4.2.11 *AppDbGetKey() 32*

4.2.12 AppDbSetKey() 32

*4.2.13 *AppDbGetCccTbl() 33*

4.2.14 AppDbSetCccTblValue() 33

4.2.15 AppDbGetDiscStatus() 33

4.2.16 AppDbSetDiscStatus() 33

4.2.17 AppDbGetHdlList() 34

4.2.18 AppDbSetHdlList() 34

*4.2.19 *AppDbGetDevName() 34*

4.2.20 AppDbSetDevName() 34

5 UI Interface 34

5.1 Constants and Data Types 35

5.1.1 UI event enumeration 35

5.1.2 Button press enumeration 35

5.1.3 LED values 36

5.1.4 Sound tone values 36

5.1.5 appUiSound_t 37

5.1.6 appUiLed_t 37

5.2 Functions 37

5.2.1 AppUiAction() 37

5.2.2 AppUiDisplayPasskey() 37

5.2.3 AppUiDisplayRssi() 38

5.2.4 AppUiBtnRegister() 38

5.2.5 AppUiSoundPlay() 38

5.2.6 AppUiSoundStop() 38

5.2.7 AppUiLedStart() 38

5.2.8 AppUiLedStop() 39

5.3 Callback Interface 39

*5.3.1 (*appUiBtnCback_t)() 39*

6 HW Interface 39

6.1 Constants and Data Types 39

6.1.1 appHrm_t 39

6.1.2 appDateTime_t 39

6.1.3 appBpm_t 40

6.1.4 appWsm_t 40

6.1.5 appTm_t 41

6.1.6 appPlxCm_t 41

6.1.7 appPlxScm_t 41

6.2 Functions 42

6.2.1 AppHwBattRead() 42

6.2.2 AppHwHrmRead() 42

6.2.3 AppHwBpmRead() 42

6.2.4 AppHwWsmRead() 43

6.2.5 AppHwTmRead() 43

6.2.6 AppHwTmSetUnits () 43

6.2.7 AppHwPlxcmRead() 44

6.2.8 AppHwPlxscmRead() 44

1 Preface

This preface introduces the Cordio Application Framework API Reference Manual.

1.1 About this book

This document describes the Cordio Application Framework API and lists the API functions and their parameters.

1.1.1 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.

1.1.2 Using this book

This book is organized into the following chapters:

- **Introduction**
  Read this for an overview of the API.
- **Main Interface**
  Read this for a description of the main interface.
- **DB Interface**
  Read this for a description of the device database interface.
- **UI Interface**
  Read this for a description of the UI interface API functions.
- **HW Interface**
  Read this for a description of hardware interface API functions.

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
AE	Advertising Extensions
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
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.

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:

- The title.
- The number, ARM-EPM-115157.
- 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 of the App Framework subsystem. The App Framework is a service layer for applications that simplifies application development.

2.1 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.

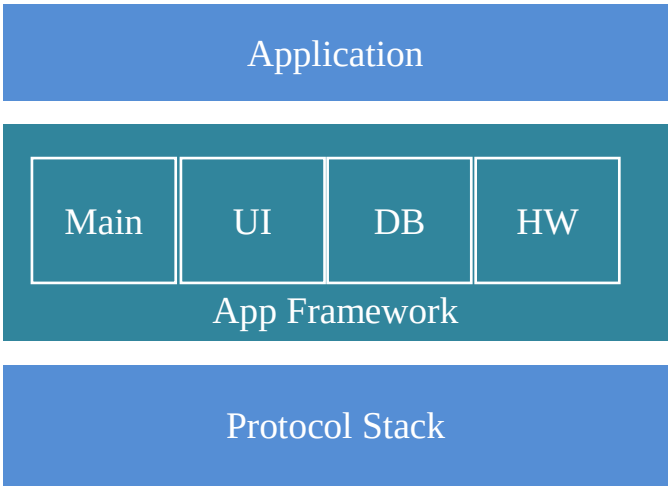

Figure 1. App Framework software system diagram.

2.2 Modules

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

Table 1 API modules

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.

3 Main Interface

3.1 Constants and Data Types

3.1.1 Discoverable/connectable mode

Discoverable/connectable mode used by function `AppAdvStart()`.

Table 2 Discoverable/connectable mode

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

3.1.2 Advertising and scan data storage locations

Advertising and scan data storage locations.

Table 3 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.

3.1.3 Service discovery and configuration client status

Service discovery and configuration client status.

3.1.4 Table 4 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.

3.1.5 Actions for incoming requests

Actions for incoming requests.

3.1.6 Table 5 Actions for incoming requests

Name	Description
APP_ACT_ACCEPT	Accept incoming request.
APP_ACT_REJECT	Reject incoming request.
APP_ACT_NONE	Do nothing—app will handle incoming request.

3.1.7 appAdvCfg_t

Configurable parameters for advertising.

Table 6 appAdvCfg_t

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

3.1.8 appExtAdvCfg_t

Configurable parameters for extended advertising.

Table 7 appExtAdvCfg_t

Type	Name	Description
uint16_t	advDuration[]	Advertising durations in ms.
uint16_t	advInterval[]	Advertising intervals 0.625 ms units.
uint8_t	maxEaEvents[]	Maximum number of extended advertising events Controller will send prior to terminating extended advertising.
bool_t	useLegacyPdu[]	Whether to use legacy advertising PDUs with extended advertising. If set to TRUE then length of advertising data cannot exceed 31octets.

3.1.9 appSlaveCfg_t

Configurable parameters for slave.

Table 8 appSlaveCfg_t

Type	Name	Description
uint8_t	connMax	Maximum connections.

3.1.10 appMasterCfg_t

Configurable parameters for master.

Table 9 appMasterCfg_t

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).

3.1.11 appSecCfg_t

Configurable parameters for security.

Table 10 appSecCfg_t

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.

3.1.12 appUpdateCfg_t

Configurable parameters for connection parameter update.

Table 11 appUpdateCfg_t

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.

3.1.13 appReqActCfg_t

Configurable parameters for incoming request actions.

Table 12 appReqActCfg_t

Type	Name	Description
uint8_t	remConnParamReqAct	Action for the remote connection parameter request.

3.1.14 appDiscCfg_t

Configurable parameters for slave.

Table 13 appDiscCfg_t

Type	Name	Description
bool_t	connMax	TRUE to wait for a secure connection before initiating discovery.

3.1.15 appCfg_t

Configurable parameters for application.

Table 14 appCfg _t

Type	Name	Description
bool_t	abortDisc	TRUE to abort service discovery if service not found.
bool_t	disconnect	TRUE to disconnect if ATT transaction times out.

3.1.16 appDevInfo_t

Device information data type.

Table 15 appDevInfo_t

Type	Name	Description
bdAddr_t	addr	Peer device address.
uint8_t	addrType	Peer address type.
uint8_t	directAddrType	Type of address directed advertisement is addressed

		to.
bdAddr_t	directAddr	Address directed advertisement is addressed to.

3.2 Global Variables

3.2.1 pAppAdvCfg

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

3.2.2 pAppExtAdvCfg

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

3.2.3 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.

3.2.4 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.

3.2.5 pAppSecCfg

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

3.2.6 pAppUpdateCfg

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

3.2.7 pAppDiscCfg

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

3.2.8 pAppCfg

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

3.2.9 pAppMasterReqActCfg

This is a pointer to the master incoming request actions used by the application. The application must set this variable during system initialization.

3.2.10 pAppSlaveReqActCfg

This is a pointer to the master incoming request actions used by the application. The application must set this variable during system initialization.

3.3 Initialization Functions

3.3.1 AppSlaveInit()

Initialize the App Framework for operation as a Bluetooth LE slave.

Syntax:

```
void AppSlaveInit(voidDesc)
```

This function is generally called once during system initialization before any other App Framework API functions are called.

3.3.2 AppMasterInit()

Initialize the App Framework for operation as a Bluetooth LE master.

Syntax:

```
void AppMasterInit(void)
```

This function is generally called once during system initialization before any other App Framework API functions are called.

3.4 Advertising Functions

3.4.1 AppAdvSetData()

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.

Syntax:

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

Where:

- `location`: Data location. See 3.1.2.
- `len`: Length of the data. Maximum length is 31 bytes.
- `pData`: Pointer to the data.

3.4.2 AppAdvStart()

Start advertising using the parameters for the given mode.

Syntax:

```
void AppAdvStart(uint8_t mode)
```

Where:

- `mode`: Discoverable/connectable mode. 3.1.1.

3.4.3 AppAdvStop()

Stop advertising.

Syntax:

```
void AppAdvStop(void)
```

The device will no longer be connectable or discoverable.

3.4.4 AppAdvSetAdValue()

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.

Syntax:

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

Where:

- 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.

3.4.5 AppSlaveIsAdvertising()

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

Syntax:

```
bool_t AppSlaveIsAdvertising(void)
```

Return TRUE if device is advertising, FALSE otherwise.

1.3 Advertising Extensions

To enable Advertising Extensions (AE) within an application:

- Add the following files to the project
    - app_slave_ae.c (for slave role)
    - dm_adv_ae.c (for advertising role)
    - dm_conn_master_ae.c (for central role - Master)
    - dm_conn_slave_ae.c (for peripheral role - Slave)
    - dm_scan_ae.c (for scanning role)
    - hci_cmd_ae.c (for HCI command interface)

- Configure AE in wsfOsInit() through proper DM initialization by:

Table 16 DM AE Initialization API

For	Replace	With
Advertising Role	DmAdvInit()	DmExtAdvInit()

Scanning Role	DmScanInit()	DmExtScanInit()
Peripheral Role	DmConnSlaveInit()	DmExtConnSlaveInit()
Central Role	DmConnMasterInit()	DmExtConnMasterInit()

3.4.6 AppExtAdvSetData()

Set extended advertising 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.

Syntax:

```
void AppExtAdvSetData(uint8_t advHandle, uint8_t location, uint16_t len, uint8_t
        *pData, uint16_t bufLen)
```

Where:

- `advHandle`: Advertising handle.
- `location`: Data location. See 3.1.2.
- `len`: Length of the data. Maximum length is 31 bytes.
- `pData`: Pointer to the data.
- `bufLen`: Length of the data buffer maintained by Application. Minimum length is 31 bytes.

3.4.7 AppExtAdvStart()

Start extended advertising using the parameters for the given mode.

Syntax:

```
void AppExtAdvStart(uint8_t numSets, uint8_t *pAdvHandles, uint8_t mode)
```

Where:

`numSets`: Number of advertising sets.
`pAdvHandles`: Advertising handles array.
`mode`: Discoverable/connectable mode. 3.1.1.

3.4.8 AppExtAdvStop()

Stop extended advertising. If the number of sets is set to 0 then all advertising sets are disabled.

Syntax:

```
void AppExtAdvStop(uint8_t numSets, uint8_t *pAdvHandles)
```

Where:

`numSets`: Number of advertising sets.
`pAdvHandles`: Advertising handles array.

3.4.9 AppExtAdvSetAdValue()

Set the value of an advertising data element in the extended 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.

Syntax:

```
bool_t AppExtAdvSetAdValue(uint8_t advHandle, uint8_t location, uint8_t adType,
          uint8_t len, uint8_t *pValue)
```

Where:

- advHandle: Advertising handle.
- 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.

3.5 Scanning Functions

3.5.1 AppScanStart()

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

Syntax:

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

Where:

- mode: Discoverability mode. See the *Device Manager API Reference Manual*.
- scanType: Scan type. See *Device Manager API Reference Manual*.
- duration: The scan duration, in milliseconds. If set to zero, scanning will continue until AppScanStop() is called.

3.5.2 AppScanStop()

This function is called to stop scanning.

Syntax:

```
void AppScanStop(void)
```

3.5.3 *AppScanGetResult()

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

Syntax:

```
appDevInfo_t *AppScanGetResult(uint8_t idx))
```

Where:

- `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.

3.5.4 AppScanGetNumResults()

Get the number of stored scan results.

Syntax:

```
void)
```

Where:

```
int8_t AppScanGetNumResults(void)
```

3.6 Connection and Security Functions

3.6.1 AppConnClose()

Close a connection with the given connection identifier.

Syntax:

```
void AppConnClose(dmConnId_t connId)
```

Where:

- `connId`: Connection identifier. See *Device Manager API Reference Manual*.

3.6.2 AppConnIsOpen()

Check if a connection is open.

Syntax:

```
dmConnId_t AppConnIsOpen(void)
```

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.

3.6.3 AppHandlePasskey()

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.

Syntax:

```
void AppHandlePasskey(dmSecAuthReqIndEvt_t *pAuthReq)
```

Where:

- `pAuthReq`: DM authentication requested event structure. See *Device Manager API Reference Manual*.

3.6.4 AppSetBondable()

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.

Syntax:

```
void AppSetBondable(bool_t bondable)
```

Where:

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

3.6.5 AppSlaveSecurityReq()

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.

Syntax:

```
void AppSlaveSecurityReq(dmConnId_t connId)
```

Where:

- `connId`: Connection identifier. See *Device Manager API Reference Manual*.

3.6.6 AppConnAccept()

Accept a connection to a peer device with the given address.

Syntax:

```
void AppConnAccept(uint8_t advType, uint8_t addrType, uint8_t *pAddr)
```

Where:

- `advType`: Advertising type.
- `addrType`: Address type.
- `pAddr`: Peer device address.

3.6.7 AppExtConnAccept()

Accept a connection to a peer device with the given address using a given advertising set.

Syntax:

```
void AppExtConnAccept(uint8_t advHandle, uint8_t advType, uint8_t addrType,
        uint8_t *pAddr)
```

Where:

- `advHandle`: Advertising handle.
- `advType`: Advertising type.
- `addrType`: Address type.
- `pAddr`: Peer device address.

3.6.8 AppMasterSecurityReq()

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.

Syntax:

```
void AppMasterSecurityReq(dmConnId_t connId)
```

Where:

- `connId`: Connection identifier. See *Device Manager API Reference Manual*.

3.7 Discovery Functions

3.7.1 AppDiscInit()

Initialize app framework discovery.

Syntax:

```
void AppDiscInit(void)
```

This function is generally called once during system initialization before any other App Framework API functions are called.

3.7.2 AppDiscRegister()

Register a callback function to service discovery status.

Syntax:

```
void AppDiscRegister(appDiscCback_t cback)
```

Where:

- `cback`: Application service discovery callback function.

3.7.3 AppDiscSetHdlList()

Set the discovery cached handle list for a given connection.

Syntax:

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

Where:

- `connId`: Connection identifier. See *Device Manager API Reference Manual*.
- `listLen`: Length of characteristic and handle lists.
- `pHdlList`: Characteristic handle list.

3.7.4 AppDiscComplete()

Syntax:

```
void AppDiscComplete(dmConnId_t connId, uint8_t status)
```

Where:

- `connId`: Connection identifier. See *Device Manager API Reference Manual*.
- `status`: Service or configuration status. See 3.1.3.

3.7.5 AppDiscFindService()

Perform service and characteristic discovery for a given service.

Syntax:

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

Where:

- `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.

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`.

3.7.6 AppDiscConfigure()

Configure characteristics for discovered services.

Syntax:

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

Where:

- `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.

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`.

3.7.7 AppDiscServiceChanged()

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.

Syntax:

```
void AppDiscServiceChanged(attEvt_t *pMsg)
```

Where:

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

3.7.8 AppDiscProcDmMsg()

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

Syntax:

```
void AppDiscProcDmMsg(dmEvt_t *pMsg)
```

Where:

- pMsg: Pointer to DM callback event message. See *Device Manager API Reference Manual*.

3.7.9 AppDiscProcAttMsg()

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

Syntax:

```
void AppDiscProcDmMsg(dmEvt_t *pMsg)
```

Where:

- pMsg: Pointer to ATT callback event message. See *Attribute Protocol API Reference Manual*.

3.8 Message Processing Functions

3.8.1 AppSlaveProcDmMsg()

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

Syntax:

```
void AppSlaveProcDmMsg(dmEvt_t *pMsg)
```

Where:

- pMsg: Pointer to DM callback event message. See *Device Manager API Reference Manual*.

3.8.2 AppSlaveSecProcDmMsg()

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

Syntax:

```
void AppSlaveSecProcDmMsg(dmEvt_t *pMsg)
```

Where:

- pMsg: Pointer to DM callback event message. See *Device Manager API Reference Manual*.

3.8.3 AppMasterProcDmMsg()

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

Syntax:

```
void AppMasterProcDmMsg(dmEvt_t *pMsg)
```

Where:

- pMsg: Pointer to DM callback event message. See *Device Manager API Reference Manual*.

3.8.4 AppMasterSecProcDmMsg()

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

Syntax:

```
void AppMasterSecProcDmMsg(dmEvt_t *pMsg)
```

Where:

- pMsg: Pointer to DM callback event message. See *Device Manager API Reference Manual*.

3.8.5 AppServerConnCback()

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.

Syntax:

```
void AppMasterSecProcDmMsg(dmEvt_t *pMsg)
```

Where:

- pDmEvt: Pointer to DM callback event message. See *Device Manager API Reference Manual*].

3.9 Callback Interface

3.9.1 (*appDiscCback_t)()

Service discovery and configuration callback.

Syntax:

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

Where:

- `connId`: Connection identifier. See *Device Manager API Reference Manual*.
- `status`: Service or configuration status.

4 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.

4.1 Constants and Data Types

4.1.1 appDbHdl_t

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

4.1.2 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.

4.2 Functions

4.2.1 AppDbInit()

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

4.2.2 AppDbNewRecord()

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

Syntax:

```
appDbHdl_t AppDbNewRecord(uint8_t addrType, uint8_t *pAddr))
```

Where:

- `addrType`: Address type. See *Device Manager API Reference Manual*.
- `pAddr`: Peer device address.

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

The function returns the database record handle.

4.2.3 AppDbDeleteRecord()

Delete a new device database record. This function is called if bonding fails or if the application desired to remove a bond.

Syntax:

```
void AppDbDeleteRecord(appDbHdl_t hdl)
```

Where:

- `hdl`: Database record handle.

4.2.4 AppDbValidateRecord()

Validate a new device database record. This function is called when pairing is successful and the devices are bonded.

Syntax:

```
void AppDbValidateRecord(appDbHdl_t hdl, uint8_t keyMask)
```

Where:

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

4.2.5 AppDbCheckValidRecord()

Check if a record has been validated. If it has not, delete it. This function is typically called when the connection is closed.

Syntax:

```
void AppDbCheckValidRecord(appDbHdl_t hdl))
```

Where:

- `hdl`: Database record handle.

4.2.6 AppDbCheckBonded()

Check if there is a stored bond with any device.

Syntax:

```
bool_t AppDbCheckBonded(void)
```

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

4.2.7 AppDbDeleteAllRecords()

Delete all database records.

Syntax:

```
void AppDbDeleteAllRecords((void)
```

4.2.8 AppDbFindByAddr()

Find a device database record by peer address.

Syntax:

```
appDbHdl_t AppDbFindByAddr(uint8_t addrType, uint8_t *pAddr)
```

Where:

- `addrType`: Address type. See *Device Manager API Reference Manual*.
- `pAddr`: Peer device address.

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

4.2.9 AppDbFindByLtkReq()

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.

Syntax:

```
appDbHdl_t AppDbFindByLtkReq(uint16_t encDiversifier, uint8_t *pRandNum)
```

Where:

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

4.2.10 AppDbGetHdl()

Get the device database record handle associated with an open connection.

Syntax:

```
appDbHdl_t AppDbGetHdl(dmConnId_t connId)
```

Where:

- `connId`: Connection identifier. See *Device Manager API Reference Manual*.

This function returns the database record handle or APP_DB_HDL_NONE.

4.2.11 *AppDbGetKey()

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

Syntax:

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

Where:

- `hdl`: Database record handle.
- `type`: Type of key to get. See *Device Manager API Reference Manual*.
- `pSecLevel`: If the key is valid, returns the security level of the key. See *Device Manager API Reference Manual*.

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

4.2.12 AppDbSetKey()

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

Syntax:

```
void AppDbSetKey(appDbHdl_t hdl, dmSecKeyIndEvt_t *pKey)
```

Where:

- `hdl`: Database record handle.
- `pKey`: Key data. See *Device Manager API Reference Manual*.

4.2.13 *AppDbGetCccTbl()

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

Syntax:

```
uint16_t *AppDbGetCccTbl(appDbHdl_t hdl)
```

Where:

- `hdl`: Database record handle.

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

4.2.14 AppDbSetCccTblValue()

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.

Syntax:

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

Where:

- `hdl`: Database record handle.
- `idx`: Table index. See *Attribute Protocol API Reference Manual*.
- `value`: Client characteristic configuration value. See *Attribute Protocol API Reference Manual*.

4.2.15 AppDbGetDiscStatus()

Get the discovery status.

Syntax:

```
uint8_t AppDbGetDiscStatus(appDbHdl_t hdl)
```

Where:

- `hdl`: Database record handle.

This function returns the discovery status.

4.2.16 AppDbSetDiscStatus()

Set the discovery status.

Syntax:

```
void AppDbSetDiscStatus(appDbHdl_t hdl, uint8_t status)
```

Where:

- `hdl`: Database record handle.
- `status`: The discovery status. See 3.1.3.

4.2.17 AppDbGetHdlList()

Get the cached handle list.

Syntax:

```
uint16_t *AppDbGetHdlList(appDbHdl_t hdl)
```

Where:

- `hdl`: Database record handle.

This function returns a pointer to the handle list.

4.2.18 AppDbSetHdlList()

Set the discovery status.

Syntax:

```
void AppDbSetHdlList(appDbHdl_t hdl, uint16_t *pHdlList)
```

Where:

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

4.2.19 *AppDbGetDevName()

Get the device name.

Syntax:

```
char *AppDbGetDevName(uint8_t *pLen)
```

Where:

- `pLen`: Returned device name length.

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

4.2.20 AppDbSetDevName()

Set the device name.

Syntax:

```
void AppDbSetDevName(uint8_t len, char *pStr)
```

Where:

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

5 UI Interface

The UI interface provides the application with simple user interface abstractions for button press

handling, sounds, display, and other user feedback.

5.1 Constants and Data Types

5.1.1 UI event enumeration

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

Table 17 UI event enumeration

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.

5.1.2 Button press enumeration

Button press enumeration.

Table 18 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.

5.1.3 LED values

LED values.

Table 19 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.

5.1.4 Sound tone values

Sound tone values.

Table 20 Sound tone Vvalues

Name	Description
APP_UI_SOUND_WRAP	Sound tone value for wrap/repeat.

5.1.5 appUiSound_t

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

Table 21 appUiSound_t

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

5.1.6 appUiLed_t

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

Table 22 appUiLed_t

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

5.2 Functions

5.2.1 AppUiAction()

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.

Syntax:

```
void AppUiAction(uint8_t event)
```

Where:

- event: User interface event value. See 5.1.1.

5.2.2 AppUiDisplayPasskey()

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

Syntax:

```
void AppUiDisplayPasskey(uint32_t passkey)
```

Where:

- `passkey`: Passkey to display.

5.2.3 AppUiDisplayRssi()

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

Syntax:

```
void AppUiDisplayRssi(int8_t rssi)
```

Where:

- `rssi`: RSSI value to display.

5.2.4 AppUiBtnRegister()

Register a callback function to receive button presses.

Syntax:

```
void AppUiBtnRegister(appUiBtnCback_t cback)
```

Where:

- `cback`: Application button callback function.

5.2.5 AppUiSoundPlay()

Play a sound.

Syntax:

```
void AppUiSoundPlay(const appUiSound_t *pSound)
```

Where:

- `pSound`: Pointer to sound tone/duration array. See 5.1.3.

5.2.6 AppUiSoundStop()

Stop the sound that is currently playing.

Syntax:

```
void AppUiSoundStop(void)
```

5.2.7 AppUiLedStart()

Start LED blinking.

Syntax:

```
void AppUiLedStart(const appUiLed_t *pLed)
```

Where:

- `pLed`: Pointer to LED data structure. See 5.1.6.

5.2.8 AppUiLedStop()

Stop LED blinking.

Syntax:

```
void AppUiLedStop(void)
```

5.3 Callback Interface

5.3.1 (*appUiBtnCback_t)()

This callback function sends button events to the application.

Syntax:

```
void (*appUiBtnCback_t)(uint8_t btn)
```

Where:

- `btn`: Button press event. See 5.1.2.

6 HW Interface

The HW interface provides an abstraction layer for hardware sensors.

6.1 Constants and Data Types

6.1.1 appHrm_t

Heart rate measurement structure.

Table 23 appHrm_t

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.

6.1.2 appDateTime_t

Date and time structure.

Table 24 appDateTime_t

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.

6.1.3 appBpm_t

Blood pressure measurement structure.

Table 25 appBpm_t

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.

6.1.4 appWsm_t

Weight scale measurement structure.

Table 26 appWsm_t

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

6.1.5 appTm_t

Temperature measurement structure.

Table 27 appTm_t

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

6.1.6 appPlxCm_t

Pulse oximeter continuous measurement structure.

Table 28 appTm_t

Type	Name	Description
uint8_t	flags	Flags
uint16_t	spo2	SpO2PR-Spot-Check - SpO2
uint16_t	pulseRate	SpO2PR-Spot-Check - Pulse Rate
uint16_t	spo2Fast	SpO2PR-Spot-Check Fast - SpO2
uint16_t	pulseRateFast	SpO2PR-Spot-Check Fast - Pulse Rate
uint16_t	spo2Slow	SpO2PR-Spot-Check Slow - SpO2
uint16_t	pulseRateSlow	SpO2PR-Spot-Check Slow - Pulse Rate
uint16_t	measStatus	Measurement Status
uint32_t	sensorStatus	Device and Sensor Status
uint16_t	pulseAmpIndex	Pulse Amplitude Index

6.1.7 appPlxScm_t

Pulse oximeter spot check measurement structure.

Table 29 appTm_t

Type	Name	Description
uint8_t	flags	Flags
uint16_t	spo2	SpO2PR-Spot-Check - SpO2

uint16_t	pulseRate	SpO2PR-Spot-Check - Pulse Rate
appDateTime_t	timestamp	Timestamp
uint16_t	measStatus	Measurement Status
uint32_t	sensorStatus	Device and Sensor Status
uint16_t	pulseAmpIndex	Pulse Amplitude Index
uint8_t	flags	Flags

6.2 Functions

6.2.1 AppHwBattRead()

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

Syntax:

```
void AppHwBattRead(uint8_t *pLevel)
```

Where:

- pLevel: Battery level return value.

6.2.2 AppHwHrmRead()

Perform a heart rate measurement.

Syntax:

```
void AppHwHrmRead(appHrm_t *pHrm)
```

Where:

- pHrm: Heart rate measurement return value.

Return the heart rate along with any RR interval data.

6.2.3 AppHwBpmRead()

Perform a blood pressure measurement.

Syntax:

```
void AppHwBpmRead(bool_t intermed, appBpm_t *pBpm)
```

Where:

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

Return the measurement data.

6.2.4 AppHwWsmRead()

Perform a weight scale measurement.

Syntax:

```
void AppHwWsmRead(appWsm_t *pWsm)
```

Where:

- `pWsm`: Weight scale measurement return value.

Return the measurement data.

6.2.5 AppHwTmRead()

Perform a temperature measurement.

Syntax:

```
void AppHwTmRead(bool_t intermed, appWsm_t *pWsm)
```

Where:

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

Return the measurement data.

6.2.6 AppHwTmSetUnits ()

Set the temperature measurement units.

Syntax:

```
void AppHwTmSetUnits (uint8_t units)
```

Where:

- `units`: `CH_TM_FLAG_UNITS_C` or `CH_TM_FLAG_UNITS_F`.

6.2.7 AppHwPlxcmRead()

Perform a pulse oximeter continuous measurement.

Syntax:

```
void AppHwPlxcmRead(appPlxCm_t *pPlxcm)
```

Where:

- `pPlxcm`: Pulse oximeter measurement return value.

Return the measurement data.

6.2.8 AppHwPlxscmRead()

Perform a pulse oximeter spot check measurement.

Syntax:

```
void AppHwPlxscmRead(appPlxScm_t *pPlxscm)
```

Where:

- `pPlxscm`: Pulse oximeter measurement return value.

Return the measurement data.