

 免費電子書

學習

bluetooth

Free unaffiliated eBook created from
Stack Overflow contributors.

#bluetooth

.....	1
1:	2
.....	2
Examples.....	2
.....	2
.....	2
2: WindowsLE	4
.....	4
Examples.....	4
.....	4
LE.....	4
LE.....	4
LERSI.....	5
3:	7
.....	7
Examples.....	7
readValue.....	7
writeValue.....	7
4: L2CAP	9
Examples.....	9
CBluez.....	9
5: TIBLE	11
Examples.....	11
BLE.....	11
.....	11
CCS.....	11
.....	15
.....	15
simple_gatt_profile.c.....	15
simple_peripheral.c.....	20
.....	24

SPI.....24

I/O.....26

.....29

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [bluetooth](#)

It is an unofficial and free bluetooth ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official bluetooth.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1:

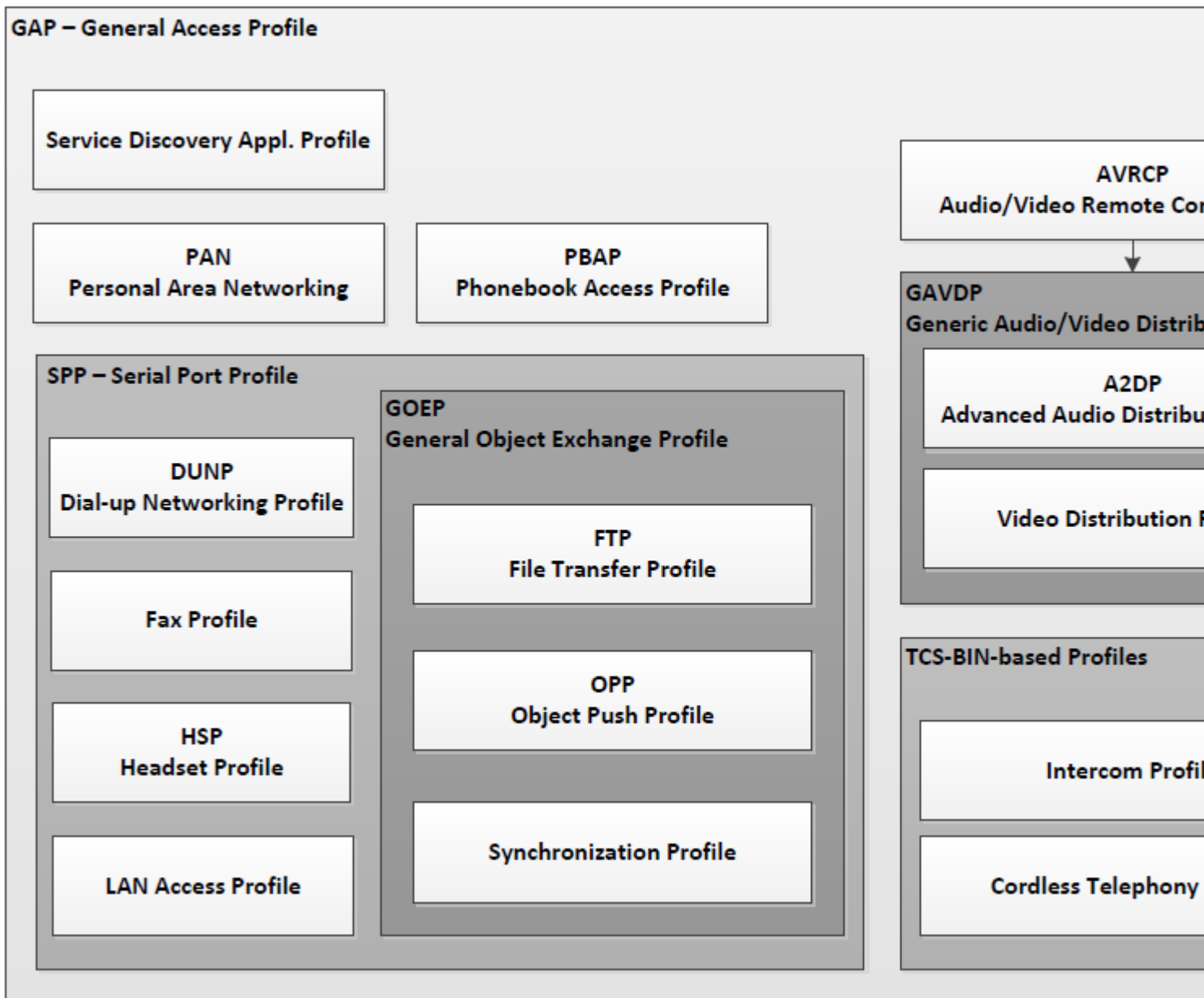
◦ 2090SIGIEEE 802.15.1◦ Ad-hoc◦

Harald BlauzahnBlauzahn“”◦ Jaap Haartsen◦ ◦

ISM2.402 GHz2.480 GHz◦ WifiISM◦

Examples

-
- ◦



BT SIG◦ ◦

A2DP -

A2DPACL。 。 。

GAVDPGAP

AVRCP - /

/AVRCP//。 AV / C/AVCTP。 A / V。 。 A / V。

GAP

HFP -

。 。 。 。 。

SPPGAP

HSP -

。 PDA。 。

SPPGAP

PBAP -

PBAP。 -。 “”“SIM”。 。 vCard。 。 。 /。

GOEPSPPGAP

<https://riptutorial.com/zh-TW/bluetooth/topic/4846/>

2: WindowsLE

- - LE。
- - LE。
- - LE。

- Windows 10。 Windows 10LE。
- Windows 10LE。

Examples

WindowsPackage.appxmanifestBluetooth。

1. Package.appxmanifest
2. Capabilities
3. Bluetooth

LE

Windows 10。 0xFFFEHello World。

```
BluetoothLEAdvertisementPublisher publisher = new BluetoothLEAdvertisementPublisher();

// Add custom data to the advertisement
var manufacturerData = new BluetoothLEManufacturerData();
manufacturerData.CompanyId = 0xFFFE;

var writer = new DataWriter();
writer.WriteString("Hello World");

// Make sure that the buffer length can fit within an advertisement payload (~20 bytes).
// Otherwise you will get an exception.
manufacturerData.Data = writer.DetachBuffer();

// Add the manufacturer data to the advertisement publisher:
publisher.Advertisement.ManufacturerData.Add(manufacturerData);

publisher.Start();
```

。

LE

。

```
BluetoothLEAdvertisementWatcher watcher = new BluetoothLEAdvertisementWatcher();

// Use active listening if you want to receive Scan Response packets as well
// this will have a greater power cost.
watcher.ScanningMode = BluetoothLEScanningMode.Active;
```

```
// Register a listener, this will be called whenever the watcher sees an advertisement.
watcher.Received += OnAdvertisementReceived;

watcher.Start();
```

- **0xFFFE Hello World** ◦ *LE Windows* ◦

```
var manufacturerData = new BluetoothLEManufacturerData();
manufacturerData.CompanyId = 0xFFFE;

// Make sure that the buffer length can fit within an advertisement payload (~20 bytes).
// Otherwise you will get an exception.
var writer = new DataWriter();
writer.WriteString("Hello World");
manufacturerData.Data = writer.DetachBuffer();

watcher.AdvertisementFilter.Advertisement.ManufacturerData.Add(manufacturerData);
```

- **0-128** ◦

```
// Set the in-range threshold to -70dBm. This means advertisements with RSSI >= -70dBm
// will start to be considered "in-range" (callbacks will start in this range).
watcher.SignalStrengthFilter.InRangeThresholdInDBm = -70;

// Set the out-of-range threshold to -75dBm (give some buffer). Used in conjunction
// with OutOfRangeTimeout to determine when an advertisement is no longer
// considered "in-range".
watcher.SignalStrengthFilter.OutOfRangeThresholdInDBm = -75;

// Set the out-of-range timeout to be 2 seconds. Used in conjunction with
// OutOfRangeThresholdInDBm to determine when an advertisement is no longer
// considered "in-range"
watcher.SignalStrengthFilter.OutOfRangeTimeout = TimeSpan.FromMilliseconds(2000);
```

```
watcher.Received += OnAdvertisementReceived;
watcher.Stopped += OnAdvertisementWatcherStopped;

private async void OnAdvertisementReceived(BluetoothLEAdvertisementWatcher watcher,
BluetoothLEAdvertisementReceivedEventArgs eventArgs)
{
    // Do whatever you want with the advertisement

    // The received signal strength indicator (RSSI)
    Int16 rssi = eventArgs.RawSignalStrengthInDBm;
}

private async void OnAdvertisementWatcherStopped(BluetoothLEAdvertisementWatcher watcher,
BluetoothLEAdvertisementWatcherStoppedEventArgs eventArgs)
{
    // Watcher was stopped
}
```

-

LERSSI

LEEventArgsRSSI

```
private async void OnAdvertisementReceived(BluetoothLEAdvertisementWatcher watcher,
BluetoothLEAdvertisementReceivedEventArgs eventArgs)
{
    // The received signal strength indicator (RSSI)
    Int16 rssi = eventArgs.RawSignalStrengthInDBm;
}
```

◦ ◦

“”◦ 0-50 DBm-50-90-90◦ ◦

WindowsLE <https://riptutorial.com/zh-TW/bluetooth/topic/5553/windowsle>

3:

- <https://developers.google.com/web/updates/2015/07/interact-with-ble-devices-on-the-web>
- <https://googlechrome.github.io/samples/web-bluetooth/index.html>

Examples

readValue

```
function onClick() {

  navigator.bluetooth.requestDevice({filters: [{services: ['battery_service']}]})
  .then(device => {
    // Connecting to GATT Server...
    return device.gatt.connect();
  })
  .then(server => {
    // Getting Battery Service...
    return server.getPrimaryService('battery_service');
  })
  .then(service => {
    // Getting Battery Level Characteristic...
    return service.getCharacteristic('battery_level');
  })
  .then(characteristic => {
    // Reading Battery Level...
    return characteristic.readValue();
  })
  .then(value => {
    let batteryLevel = value.getUint8(0);
    console.log('> Battery Level is ' + batteryLevel + '%');
  })
  .catch(error => {
    console.log('Argh! ' + error);
  });
}
```

writeValue

```
function onClick() {

  navigator.bluetooth.requestDevice({filters: [{services: ['heart_rate']}]})
  .then(device => {
    // Connecting to GATT Server...
    return device.gatt.connect();
  })
  .then(server => {
    // Getting Heart Rate Service...
    return server.getPrimaryService('heart_rate');
  })
  .then(service => {
    // Getting Heart Rate Control Point Characteristic...
    return service.getCharacteristic('heart_rate_control_point');
  })
}
```

```
.then(characteristic => {
  // Writing 1 is the signal to reset energy expended.
  let resetEnergyExpended = new Uint8Array([1]);
  return characteristic.writeValue(resetEnergyExpended);
})
.then(_ => {
  console.log('> Energy expended has been reset.');
```

<https://riptutorial.com/zh-TW/bluetooth/topic/4936/>

4: L2CAP

Examples

CBluez

```
int get_l2cap_connection () {
```

◦

```
int ssock = 0;
int csock = 0;
int reuse_addr = 1;
struct sockaddr_l2 src_addr;
struct bt_security bt_sec;
int result = 0;
```

◦ PF_BLUETOOTH SOCK_SEQPACKET TCPL2CAP BTPROTO_L2CAP ◦

```
ssock = socket(PF_BLUETOOTH, SOCK_SEQPACKET, BTPROTO_L2CAP);
```

```
if (ssock < 0) {
    perror("Opening L2CAP socket failed");
    return -1;
}
```

◦ bluetooth.h BDADDR_ANY ◦ bcopy ◦ ID ◦

```
memset(&src_addr, 0, sizeof(src_addr));
bcopy(&src_addr.l2_bdaddr, BDADDR_ANY);
src_addr.l2_family = AF_BLUETOOTH;
src_addr.l2_bdaddr_type = BDADDR_LE_PUBLIC;
src_addr.l2_cid = htobs(CID_ATT);
```

SO_REUSEADDR bind

```
setsockopt(ssock, SOL_SOCKET, SO_REUSEADDR, &reuse_addr, sizeof(reuse_addr));
```

◦ ◦

```
result = bind(ssock, (struct sockaddr*) &src_addr, sizeof(src_addr));
if (result < 0) {
    perror("Binding L2CAP socket failed");
    return -1;
}
```

◦ MEDIUM ◦

```

memset(&bt_sec, 0, sizeof(bt_sec));
bt_sec.level = BT_SECURITY_MEDIUM;
result = setsockopt(ssock, SOL_BLUETOOTH, BT_SECURITY, &bt_sec, sizeof(bt_sec));
if (result != 0) {
    perror("Setting L2CAP security level failed");
    return -1;
}

```

ssock。 ◦ listen。

```

result = listen(ssock, 10);
if (result < 0) {
    perror("Listening on L2CAP socket failed");
    return -1;
}

```

◦ acceptpeer_addr。 csock/。

```

memset(peer_addr, 0, sizeof(*peer_addr));
socklen_t addrlen = sizeof(*peer_addr);
csock = accept(ssock, (struct sockaddr*)peer_addr, &addrlen);
if (csock < 0) {
    perror("Accepting connection on L2CAP socket failed");
    return -1;
}

```

◦ batostr。

```

printf("Accepted connection from %s", batostr(&peer_addr->l2_bdaddr));

```

◦ csock。

```

close(ssock);
return csock;
}

```

L2CAP <https://riptutorial.com/zh-TW/bluetooth/topic/6558/l2cap>

5: TIBLE

Examples

BLE

TICC26XXSoCBLEMCU。MCUTI API。◦◦

BLE“Hello World”MCUBLEBLEPC。◦

◦

1. BLE

TIBLE-STACK-2-2-0。 “C\ ti”。

2. IDE -

- ARMIAR Embedded Workbench。 30。
- TICode Composer StudioCCS。 TIIDE。 CCS V6.1.3

3. TIXDS100 USBJTAG。

CCS

Simple Peripheral ProfileBLE-Stack。 CCS。

1. CCS。 - >。 “”“Code Compose Studio - > CCS Projects”“”。



Getting Started

Import

Select

Imports existing CCS

Select an import source

type filter text

- General
 - Archive File
 - Existing Project
 - File System
 - Preferences
- C/C++
- Code Composer
 - Build Variables
 - CCS Projects
 - Legacy CCSv3
- Energia
- Git
- Install
- Remote Systems
- Run/Debug

2. 'C:\ti\simplelink\ble_sdk_2_02_00_31\examples\cc2650em\simple_peripheral\ccs'

◦ ◦ “” ◦ ◦

◦ ◦ “” ◦ ◦

 Import CCS Eclipse Projects

Select CCS Projects to Import

<https://riptutorial.com/zh-TW/home> Select a directory to search for existing CCS Eclipse projects.

Simple Peripheral Profile

- simple_peripheral_cc2650em_app
- simple_peripheral_cc2650em_stack

Import CCS Eclipse Projects

'cc2650em'TI cc2650. _stackTIBEL-Stack-2-2-0. ◦ _appBLE.◦

Select CCS Projects to Import

'Project-> Build All'。 'compress_dwarf'

- “Propoerties”。
- “Build-> ARM Linker”“Edit Flags”。
- '--compress_dwarf = off'。

“Run-> debug”MCU。

BLE。 /PROFILES / simple_gatt_profile.c.hApplication / simple_peripheral.c.h

simple_gatt_profile.c

。 UUID0xFFF05。 simple_gatt_profile.c。 。

		UUID		
simplePeripheralChar1	1	0xFFF1	1	
simplePeripheralChar2	1	0xFFF2	2	
simplePeripheralChar3	1	0xFFF3	3	
simplePeripheralChar4	1	0xFFF4	4	
simplePeripheralChar5		0xFFF5	5	

。 MCUsimplePeripheralChar4。

。

```
/*
 * Profile Attributes - Table
 */

static gattAttribute_t simpleProfileAttrTbl[SERVAPP_NUM_ATTR_SUPPORTED] =
{
    // Simple Profile Service
    {
        { ATT_BT_UUID_SIZE, primaryServiceUUID }, /* type */
        GATT_PERMIT_READ, /* permissions */
        0, /* handle */
        (uint8 *)&simpleProfileService /* pValue */
    },

    // Characteristic 1 Declaration
    {
        { ATT_BT_UUID_SIZE, characterUUID },
        GATT_PERMIT_READ,
        0,
        &simpleProfileChar1Props
    },
}
```

```

// Characteristic Value 1
{
  { ATT_UUID_SIZE, simpleProfileChar1UUID },
  GATT_PERMIT_READ | GATT_PERMIT_WRITE,
  0,
  &simpleProfileChar1
},

// Characteristic 1 User Description
{
  { ATT_BT_UUID_SIZE, charUserDescUUID },
  GATT_PERMIT_READ,
  0,
  simpleProfileChar1UserDesc
},
...
};

```

“primaryServiceUUID”UUID0xFFFD。 。 。 BLE。

```

// Register GATT attribute list and CBs with GATT Server App
status = GATTServApp_RegisterService( simpleProfileAttrTbl,
                                     GATT_NUM_ATTRS( simpleProfileAttrTbl ),
                                     GATT_MAX_ENCRYPT_KEY_SIZE,
                                     &simpleProfileCBs );

```

“”。

```

/*****
 * PROFILE CALLBACKS
 */

// Simple Profile Service Callbacks
// Note: When an operation on a characteristic requires authorization and
// pfnAuthorizeAttrCB is not defined for that characteristic's service, the
// Stack will report a status of ATT_ERR_UNLIKELY to the client. When an
// operation on a characteristic requires authorization the Stack will call
// pfnAuthorizeAttrCB to check a client's authorization prior to calling
// pfnReadAttrCB or pfnWriteAttrCB, so no checks for authorization need to be
// made within these functions.
CONST gattServiceCBs_t simpleProfileCBs =
{
  simpleProfile_ReadAttrCB, // Read callback function pointer
  simpleProfile_WriteAttrCB, // Write callback function pointer
  NULL // Authorization callback function pointer
};

```

simpleProfile_ReadAttrCB。 simpleProfile_WriteAttrCB。 。

。

```

/*****
 * @fn          simpleProfile_ReadAttrCB
 *
 * @brief       Read an attribute.
 *

```

```

* @param      connHandle - connection message was received on
* @param      pAttr - pointer to attribute
* @param      pValue - pointer to data to be read
* @param      pLen - length of data to be read
* @param      offset - offset of the first octet to be read
* @param      maxLen - maximum length of data to be read
* @param      method - type of read message
*
* @return     SUCCESS, blePending or Failure
*/
static bStatus_t simpleProfile_ReadAttrCB(uint16_t connHandle,
                                          gattAttribute_t *pAttr,
                                          uint8_t *pValue, uint16_t *pLen,
                                          uint16_t offset, uint16_t maxLen,
                                          uint8_t method)
{
    bStatus_t status = SUCCESS;

    // If attribute permissions require authorization to read, return error
    if ( gattPermitAuthorRead( pAttr->permissions ) )
    {
        // Insufficient authorization
        return ( ATT_ERR_INSUFFICIENT_AUTHOR );
    }

    // Make sure it's not a blob operation (no attributes in the profile are long)
    if ( offset > 0 )
    {
        return ( ATT_ERR_ATTR_NOT_LONG );
    }

    uint16_t uuid = 0;
    if ( pAttr->type.len == ATT_UUID_SIZE )
        // 128-bit UUID
        uuid = BUILD_UINT16( pAttr->type.uuid[12], pAttr->type.uuid[13]);
    else
        uuid = BUILD_UINT16( pAttr->type.uuid[0], pAttr->type.uuid[1]);

    switch ( uuid )
    {
        // No need for "GATT_SERVICE_UUID" or "GATT_CLIENT_CHAR_CFG_UUID" cases;
        // gattserverapp handles those reads

        // characteristics 1 and 2 have read permissions
        // characteristic 3 does not have read permissions; therefore it is not
        // included here
        // characteristic 4 does not have read permissions, but because it
        // can be sent as a notification, it is included here
        case SIMPLEPROFILE_CHAR2_UUID:
            *pLen = SIMPLEPROFILE_CHAR2_LEN;
            VOID memcpy( pValue, pAttr->pValue, SIMPLEPROFILE_CHAR2_LEN );
            break;

        case SIMPLEPROFILE_CHAR1_UUID:
            *pLen = SIMPLEPROFILE_CHAR1_LEN;
            VOID memcpy( pValue, pAttr->pValue, SIMPLEPROFILE_CHAR1_LEN );
            break;

        case SIMPLEPROFILE_CHAR4_UUID:
            *pLen = SIMPLEPROFILE_CHAR4_LEN;
            VOID memcpy( pValue, pAttr->pValue, SIMPLEPROFILE_CHAR4_LEN );

```

```

        break;

    case SIMPLEPROFILE_CHAR5_UUID:
        *pLen = SIMPLEPROFILE_CHAR5_LEN;
        VOID memcpy( pValue, pAttr->pValue, SIMPLEPROFILE_CHAR5_LEN );
        break;

    default:
        // Should never get here! (characteristics 3 and 4 do not have read permissions)
        *pLen = 0;
        status = ATT_ERR_ATTR_NOT_FOUND;
        break;
}

return ( status );
}

```

◦ 7. ◦ 'ifoffset> 0'blob. blob. UUID. UUID16128. 16UUID128UUIDPC. 128UUID16UUID.

```

uint16 uuid = 0;
if ( pAttr->type.len == ATT_UUID_SIZE )
    // 128-bit UUID
    uuid = BUILD_UINT16( pAttr->type.uuid[12], pAttr->type.uuid[13]);
else
    uuid = BUILD_UINT16( pAttr->type.uuid[0], pAttr->type.uuid[1]);

```

UUID. 'pValue'.

```

switch ( uuid )
{
    case SIMPLEPROFILE_CHAR1_UUID:
        *pLen = SIMPLEPROFILE_CHAR1_LEN;
        VOID memcpy( pValue, pAttr->pValue, SIMPLEPROFILE_CHAR1_LEN );
        break;

    case SIMPLEPROFILE_CHAR2_UUID:
        *pLen = SIMPLEPROFILE_CHAR2_LEN;
        VOID memcpy( pValue, pAttr->pValue, SIMPLEPROFILE_CHAR2_LEN );
        break;

    case SIMPLEPROFILE_CHAR4_UUID:
        *pLen = SIMPLEPROFILE_CHAR4_LEN;
        VOID memcpy( pValue, pAttr->pValue, SIMPLEPROFILE_CHAR4_LEN );
        break;

    case SIMPLEPROFILE_CHAR5_UUID:
        *pLen = SIMPLEPROFILE_CHAR5_LEN;
        VOID memcpy( pValue, pAttr->pValue, SIMPLEPROFILE_CHAR5_LEN );
        break;

    default:
        *pLen = 0;
        status = ATT_ERR_ATTR_NOT_FOUND;
        break;
}

```

UUIDGATT_CLIENT_CHAR_CFG_UUID. ◦ API GATTServApp_ProcessCCCWriteReqBLE.

```

case GATT_CLIENT_CHAR_CFG_UUID:
    status = GATTServApp_ProcessCCCWriteReq( connHandle, pAttr, pValue, len,
                                              offset, GATT_CLIENT_CFG_NOTIFY |
GATT_CLIENT_CFG_INDICATE ); // allow client to request notification or indication features
    break;

```

MCU。 。 。

```

// If a charactersitic value changed then callback function to notify application of change
if ( ( notifyApp != 0xFF ) && simpleProfile_AppCBs && simpleProfile_AppCBs->
    pfnSimpleProfileChange )
{
    simpleProfile_AppCBs->pfnSimpleProfileChange( notifyApp );
}

```

BLEAPI GATTServApp_ProcessCharCfg。 APISimpleProfile_SetParameter。

```

/*****
 * @fn      SimpleProfile_SetParameter
 *
 * @brief   Set a Simple Profile parameter.
 *
 * @param   param - Profile parameter ID
 * @param   len - length of data to write
 * @param   value - pointer to data to write. This is dependent on
 *               the parameter ID and WILL be cast to the appropriate
 *               data type (example: data type of uint16 will be cast to
 *               uint16 pointer).
 *
 * @return  bStatus_t
 */
bStatus_t SimpleProfile_SetParameter( uint8 param, uint8 len, void *value )
{
    bStatus_t ret = SUCCESS;
    switch ( param )
    {
        case SIMPLEPROFILE_CHAR2:
            if ( len == SIMPLEPROFILE_CHAR2_LEN )
            {
                VOID memcpy( simpleProfileChar2, value, SIMPLEPROFILE_CHAR2_LEN );
            }
            else
            {
                ret = bleInvalidRange;
            }
            break;

        case SIMPLEPROFILE_CHAR3:
            if ( len == sizeof ( uint8 ) )
            {
                simpleProfileChar3 = *((uint8*)value);
            }
            else
            {
                ret = bleInvalidRange;
            }
            break;
    }
}

```

```

case SIMPLEPROFILE_CHAR1:
    if ( len == SIMPLEPROFILE_CHAR1_LEN )
    {
        VOID memcpy( simpleProfileChar1, value, SIMPLEPROFILE_CHAR1_LEN );
    }
    else
    {
        ret = bleInvalidRange;
    }
    break;

case SIMPLEPROFILE_CHAR4:
    if ( len == SIMPLEPROFILE_CHAR4_LEN )
    {
        //simpleProfileChar4 = *((uint8*)value);
        VOID memcpy( simpleProfileChar4, value, SIMPLEPROFILE_CHAR4_LEN );
        // See if Notification has been enabled
        GATTServApp_ProcessCharCfg( simpleProfileChar4Config, simpleProfileChar4, FALSE,
            simpleProfileAttrTbl, GATT_NUM_ATTRS(
simpleProfileAttrTbl ),
                                INVALID_TASK_ID, simpleProfile_ReadAttrCB );
    }
    else
    {
        ret = bleInvalidRange;
    }
    break;

case SIMPLEPROFILE_CHAR5:
    if ( len == SIMPLEPROFILE_CHAR5_LEN )
    {
        VOID memcpy( simpleProfileChar5, value, SIMPLEPROFILE_CHAR5_LEN );
    }
    else
    {
        ret = bleInvalidRange;
    }
    break;

default:
    ret = INVALIDPARAMETER;
    break;
}

return ( ret );
}

```

SIMPLEPROFILE_CHAR4SimpleProfile_SetParameter。

PROFILES / simple_gatt_profile.c.hBLE。

simple_peripheral.c

TIBLEOS。MCU。simple_peripheral.c。。

。

```

// Advertising interval when device is discoverable (units of 625us, 160=100ms)
#define DEFAULT_ADVERTISING_INTERVAL          160

// Limited discoverable mode advertises for 30.72s, and then stops
// General discoverable mode advertises indefinitely
#define DEFAULT_DISCOVERABLE_MODE            GAP_ADTYPE_FLAGS_GENERAL

// Minimum connection interval (units of 1.25ms, 80=100ms) if automatic
// parameter update request is enabled
#define DEFAULT_DESIRED_MIN_CONN_INTERVAL    80

// Maximum connection interval (units of 1.25ms, 800=1000ms) if automatic
// parameter update request is enabled
#define DEFAULT_DESIRED_MAX_CONN_INTERVAL    400

// Slave latency to use if automatic parameter update request is enabled
#define DEFAULT_DESIRED_SLAVE_LATENCY        0

// Supervision timeout value (units of 10ms, 1000=10s) if automatic parameter
// update request is enabled
#define DEFAULT_DESIRED_CONN_TIMEOUT         1000

// Whether to enable automatic parameter update request when a connection is
// formed
#define DEFAULT_ENABLE_UPDATE_REQUEST        TRUE

// Connection Pause Peripheral time value (in seconds)
#define DEFAULT_CONN_PAUSE_PERIPHERAL        6

// How often to perform periodic event (in msec)
#define SBP_PERIODIC_EVT_PERIOD              1000

```

DEFAULT_DESIRED_MIN_CONN_INTERVAL DEFAULT_DESIRED_MAX_CONN_INTERVAL
 DEFAULT_DESIRED_SLAVE_LATENCY。。

DEFAULT_DESIRED_CONN_TIMEOUT。 DEFAULT_ENABLE_UPDATE_REQUEST。。

SBP_PERIODIC_EVT_PERIOD。。

SimpleBLEPeripheral_init。

```

// Create one-shot clocks for internal periodic events.
Util_constructClock(&periodicClock, SimpleBLEPeripheral_clockHandler,
                    SBP_PERIODIC_EVT_PERIOD, 0, false, SBP_PERIODIC_EVT);

```

SBP_PERIODIC_EVT_PERIOD。 SBP_PERIODIC_EVT SimpleBLEPeripheral_clockHandler。

```
Util_startClock(&periodicClock);
```

Util_startClock GAPROLE_CONNECTED SimpleBLEPeripheral_processStateChangeEvt。

。

```

/*****
 * @fn      SimpleBLEPeripheral_clockHandler

```



```

*
* @brief   Handler function for clock timeouts.
*
* @param   arg - event type
*
* @return  None.
*/
static void SimpleBLEPeripheral_clockHandler(UArg arg)
{
    // Store the event.
    events |= arg;

    // Wake up the application.
    Semaphore_post(sem);
}

```

OS。 。 BLEAPI。 **BLEAPI**。 。 simpleBLEPeripheral_taskFxn。

```

/*****
* @fn      SimpleBLEPeripheral_taskFxn
*
* @brief   Application task entry point for the Simple BLE Peripheral.
*
* @param   a0, a1 - not used.
*
* @return  None.
*/
static void SimpleBLEPeripheral_taskFxn(UArg a0, UArg a1)
{
    // Initialize application
    SimpleBLEPeripheral_init();

    // Application main loop
    for (;;)
    {
        // Waits for a signal to the semaphore associated with the calling thread.
        // Note that the semaphore associated with a thread is signaled when a
        // message is queued to the message receive queue of the thread or when
        // ICall_signal() function is called onto the semaphore.
        ICall_Errno errno = ICall_wait(ICALL_TIMEOUT_FOREVER);

        if (errno == ICALL_ERRNO_SUCCESS)
        {
            ICall_EntityID dest;
            ICall_ServiceEnum src;
            ICall_HciExtEvt *pMsg = NULL;

            if (ICall_fetchServiceMsg(&src, &dest,
                                     (void **) &pMsg) == ICALL_ERRNO_SUCCESS)
            {
                uint8 safeToDealloc = TRUE;

                if ((src == ICALL_SERVICE_CLASS_BLE) && (dest == selfEntity))
                {
                    ICall_Stack_Event *pEvt = (ICall_Stack_Event *)pMsg;

                    // Check for BLE stack events first
                    if (pEvt->signature == 0xffff)
                    {
                        if (pEvt->event_flag & SBP_CONN_EVT_END_EVT)

```

```

        {
            // Try to retransmit pending ATT Response (if any)
            SimpleBLEPeripheral_sendAttRsp();
        }
    }
    else
    {
        // Process inter-task message
        safeToDealloc = SimpleBLEPeripheral_processStackMsg((ICall_Hdr *)pMsg);
    }
}

if (pMsg && safeToDealloc)
{
    ICall_freeMsg(pMsg);
}
}

// If RTOS queue is not empty, process app message.
while (!Queue_empty(appMsgQueue))
{
    sbpEvt_t *pMsg = (sbpEvt_t *)Util_dequeueMsg(appMsgQueue);
    if (pMsg)
    {
        // Process message.
        SimpleBLEPeripheral_processAppMsg(pMsg);

        // Free the space from the message.
        ICall_free(pMsg);
    }
}

if (events & SBP_PERIODIC_EVT)
{
    events &= ~SBP_PERIODIC_EVT;

    Util_startClock(&periodicClock);

    // Perform periodic application task
    SimpleBLEPeripheral_performPeriodicTask();
}
}
}

```

◦ ◦ ◦ SBP_PERIODIC_EVT SimpleBLEPeripheral_performPeriodicTask;

```

/*****
 * @fn      SimpleBLEPeripheral_performPeriodicTask
 *
 * @brief   Perform a periodic application task. This function gets called
 *          every five seconds (SBP_PERIODIC_EVT_PERIOD). In this example,
 *          the value of the third characteristic in the SimpleGATTProfile
 *          service is retrieved from the profile, and then copied into the
 *          value of the fourth characteristic.
 *
 * @param   None.
 *
 * @return  None.
 *****/

```

```

*/
static void SimpleBLEPeripheral_performPeriodicTask(void)
{
    uint8_t newValue[SIMPLEPROFILE_CHAR4_LEN];
    // user codes to do specific work like reading the temperature
    // .....
    SimpleProfile_SetParameter(SIMPLEPROFILE_CHAR4, SIMPLEPROFILE_CHAR4_LEN,
                               newValue);
}

```

UART。 SimpleProfile_SetParameterAPI。 BLE。 。

。

```

static void SimpleBLEPeripheral_charValueChangeCB(uint8_t paramID)
{
    SimpleBLEPeripheral_enqueueMsg(SBP_CHAR_CHANGE_EVT, paramID);
}

```

。

```

static void SimpleBLEPeripheral_processCharValueChangeEvt(uint8_t paramID)
{
    uint8_t newValue[SIMPLEPROFILE_CHAR1_LEN];

    switch(paramID)
    {
        case SIMPLEPROFILE_CHAR1:
            SimpleProfile_GetParameter(SIMPLEPROFILE_CHAR1, &newValue[0]);
            ProcessUserCmd(newValue[0], NULL);
            break;

        case SIMPLEPROFILE_CHAR3:
            break;

        default:
            // should not reach here!
            break;
    }
}

```

SIMPLEPROFILE_CHAR1SimpleProfile_GetParameter。

simple_peripheral.c。 。

BLEMCUGPIO。 GPIOADC。 TICC2640 MCU31GPIO。

CC2640ADCUARTSSPISSII2C。 TIBLE。 。

SPI

TIBLEAPI;API。

SPI

- <ti / drivers / SPI.h> - API
- <ti / drivers / spi / SPICC26XXDMA.h> - CC2640API
- <ti / drivers / dma / UDMACC26XX.h> - SPIuDMA

TI BLESPICC26XXDMA.h

SPIcsbp_spi.c。 。 --SPI_Handle。 --SPI_ParamsSPI。

```
#include <ti/drivers/SPI.h>
#include <ti/drivers/spi/SPICC26XXDMA.h>
#include <ti/drivers/dma/UDMACC26XX.h>

static void sbp_spiInit();

static SPI_Handle spiHandle;
static SPI_Params spiParams;

void sbp_spiInit(){
    SPI_init();
    SPI_Params_init(&spiParams);
    spiParams.mode                = SPI_MASTER;
    spiParams.transferMode       = SPI_MODE_CALLBACK;
    spiParams.transferCallbackFxn = sbp_spiCallback;
    spiParams.bitRate             = 800000;
    spiParams.frameFormat        = SPI_POLO_PHA0;
    spiHandle = SPI_open(CC2650DK_7ID_SPI0, &spiParams);
}
```

SPI_Handle。 API SPI_init。 SPI_Params_initspiParamsSPI_Params。 。 SPI800kbps
sbp_spiCallback。

SPI_openSPISPI。 SPI_openSPIID。 CC2640SPIID01。 SPI。

```
/*!
 * @def    CC2650DK_7ID_SPIName
 * @brief  Enum of SPI names on the CC2650 dev board
 */
typedef enum CC2650DK_7ID_SPIName {
    CC2650DK_7ID_SPI0 = 0,
    CC2650DK_7ID_SPI1,
    CC2650DK_7ID_SPICOUNT
} CC2650DK_7ID_SPIName;
```

SPI_HandleSPI。 - SPI_TransactionSPI。

```
/*!
 * @brief
 * A ::SPI_Transaction data structure is used with SPI_transfer(). It indicates
 * how many ::SPI_FrameFormat frames are sent and received from the buffers
 * pointed to txBuf and rxBuf.
 * The arg variable is a user-definable argument which gets passed to the
 * ::SPI_CallbackFxn when the SPI driver is in ::SPI_MODE_CALLBACK.
 */
```

```

typedef struct SPI_Transaction {
    /* User input (write-only) fields */
    size_t    count;        /*!< Number of frames for this transaction */
    void      *txBuf;       /*!< void * to a buffer with data to be transmitted */
    void      *rxBuf;       /*!< void * to a buffer to receive data */
    void      *arg;         /*!< Argument to be passed to the callback function */

    /* User output (read-only) fields */
    SPI_Status status;      /*!< Status code set by SPI_transfer */

    /* Driver-use only fields */
} SPI_Transaction;

```

SPI'txBuf'count'. SPI_transferspiHandlespiTransSPI。

```

static SPI_Transaction spiTrans;
bool sbp_spiTransfer(uint8_t len, uint8_t * txBuf, uint8_t rxBuf, uint8_t * args)
{
    spiTrans.count = len;
    spiTrans.txBuf = txBuf;
    spiTrans.rxBuf = rxBuf;
    spiTrans.arg   = args;

    return SPI_transfer(spiHandle, &spiTrans);
}

```

SPI'rxBuf'。

。 。 **API**。

```

void sbp_spiCallback(SPI_Handle handle, SPI_Transaction * transaction){
    uint8_t * args = (uint8_t *)transaction->arg;

    // may want to disable the interrupt first
    key = Hwi_disable();
    if(transaction->status == SPI_TRANSFER_COMPLETED){
        // do something here for successful transaction...
    }
    Hwi_restore(key);
}

```

I/O

SPI。 APISPIPICC26XXDMA_ObjectSPICC26XXDMA_HWAttrsV1SPI_Config。 'board.c'。

```

/* SPI objects */
SPICC26XXDMA_Object spiCC26XXDMAObjects[CC2650DK_7ID_SPICOUNT];

/* SPI configuration structure, describing which pins are to be used */
const SPICC26XXDMA_HWAttrsV1 spiCC26XXDMAHWAttrs[CC2650DK_7ID_SPICOUNT] = {
    {
        .baseAddr      = SSI0_BASE,
        .intNum        = INT_SSI0_COMB,
        .intPriority    = ~0,
        .swiPriority    = 0,
    }
}

```

```

        .powerMngrId      = PowerCC26XX_PERIPH_SSI0,
        .defaultTxBufValue = 0,
        .rxChannelBitMask = 1<<UDMA_CHAN_SSI0_RX,
        .txChannelBitMask = 1<<UDMA_CHAN_SSI0_TX,
        .mosiPin          = ADC_MOSI_0,
        .misoPin          = ADC_MISO_0,
        .clkPin           = ADC_SCK_0,
        .csnPin           = ADC_CSN_0
    },
    {
        .baseAddr         = SSI1_BASE,
        .intNum           = INT_SSI1_COMB,
        .intPriority       = ~0,
        .swiPriority       = 0,
        .powerMngrId      = PowerCC26XX_PERIPH_SSI1,
        .defaultTxBufValue = 0,
        .rxChannelBitMask = 1<<UDMA_CHAN_SSI1_RX,
        .txChannelBitMask = 1<<UDMA_CHAN_SSI1_TX,
        .mosiPin          = ADC_MOSI_1,
        .misoPin          = ADC_MISO_1,
        .clkPin           = ADC_SCK_1,
        .csnPin           = ADC_CSN_1
    }
};

/* SPI configuration structure */
const SPI_Config SPI_config[] = {
    {
        .fxnTablePtr = &SPICC26XXDMA_fxnTable,
        .object      = &spiCC26XXDMAObjects[0],
        .hwAttrs     = &spiCC26XXDMAHWAttrs[0]
    },
    {
        .fxnTablePtr = &SPICC26XXDMA_fxnTable,
        .object      = &spiCC26XXDMAObjects[1],
        .hwAttrs     = &spiCC26XXDMAHWAttrs[1]
    },
    {NULL, NULL, NULL}
};

```

SPI_ConfigSPI. fxnTablePtrobjecthwAttrs. 'fxnTablePtr'API.

“”。 “”。

'hwAttrs'SPIIOSPI.'hwAttrs'。 IO。 **CC26XX MCU**IOIO。

'board.h'IO。

```

#define ADC_CSN_1          IOID_1
#define ADC_SCK_1          IOID_2
#define ADC_MISO_1        IOID_3
#define ADC_MOSI_1        IOID_4
#define ADC_CSN_0          IOID_5
#define ADC_SCK_0          IOID_6
#define ADC_MISO_0        IOID_7
#define ADC_MOSI_0        IOID_8

```

SPI。

TIBLE <https://riptutorial.com/zh-TW/bluetooth/topic/7058/tible>

S. No		Contributors
1		Community , Jon Carlstedt , Lasse Meyer
2	WindowsLE	Carter
3		François Beaufort
4	L2CAP	Lasse Meyer
5	TIBLE	RamenChef , Zefu Dai