硬體週邊Tips

From DigiBuk

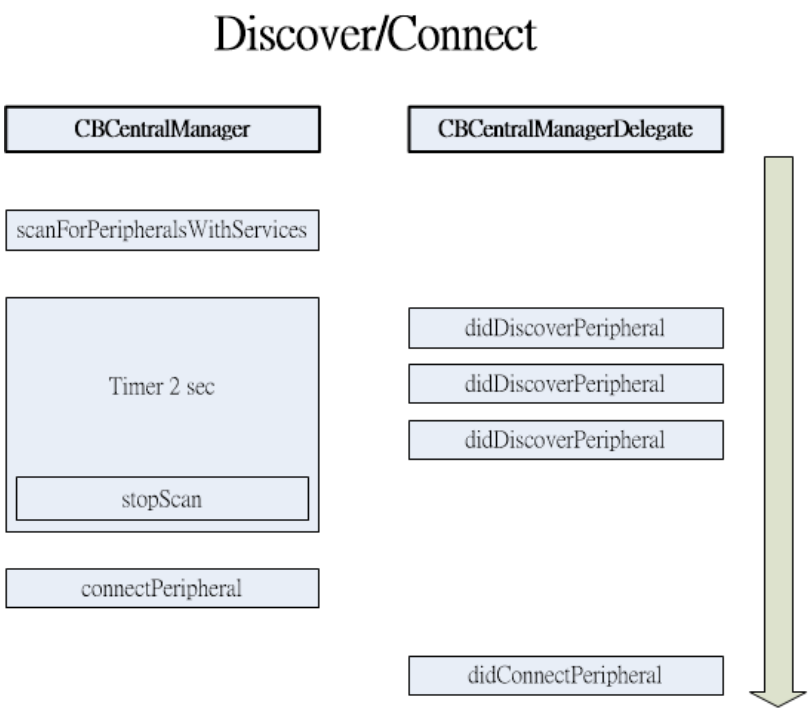
藍牙

BLE(CoreBluetooth)

Bluetooth 4.0之後就將通訊模式分為高速及低速種類，目前將低速低耗能簡稱為BLE (http://en.wikipedia.org/wiki/Bluetooth_low_energy) ，可以連接一些量測型的感測器類型像：心跳計、血壓…等，這使得iDevice不再是利用MFi認證過的Dock才能與iDevice連接，增加APP型態的多元。

如果想要跟BLE週邊連接，iOS提供了CoreBluetooth framework來與週邊連接，整個程式中分為Discover、Connect、Explore、Interact，下面將會以從連線至BLE讀取到資料為原則來介紹。

Discover/Connect



依照箭頭方向由上而下為順序來依序完成Discover與Connect流程，下面會針對此流程加上範例來介紹

CBCentralManager

要使用CoreBluetooth就要先了解一下CBCentralManager，這個Object掌控整個BLE的管理，一開始要先對CBCentralManager來做個初始化

```
CBCentralManager *CM = [[CBCentralManager alloc] initWithDelegate:self queue:nil];
```

centralManagerDidUpdateState

在初始化時指定Delegate為self，並在.h內加上Delegate宣告：

```
@interface TestCoreBluetooth : NSObject<CBCentralManagerDelegate> {
:
:
:
}
```

加完宣告後就需要完成centralManagerDidUpdateState這個Delegate，

```
-(void)centralManagerDidUpdateState:(CBCentralManager*)cManager
{
```

```
    NSMutableString* nsmstring=[NSMutableString stringWithString:@"UpdateState:"];
    BOOL isWork=FALSE;
    switch (cManager.state) {
        case CBCentralManagerStateUnknown:
            [nsmstring appendString:@"Unknown\n"];
            break;
        case CBCentralManagerStateUnsupported:
            [nsmstring appendString:@"Unsupported\n"];
            break;
        case CBCentralManagerStateUnauthorized:
            [nsmstring appendString:@"Unauthorized\n"];
            break;
        case CBCentralManagerStateResetting:
            [nsmstring appendString:@"Resetting\n"];
            break;
        case CBCentralManagerStatePoweredOff:
            [nsmstring appendString:@"PoweredOff\n"];
            if (connectedPeripheral!=NULL){
                [CM cancelPeripheralConnection:connectedPeripheral];
            }
            break;
        case CBCentralManagerStatePoweredOn:
            [nsmstring appendString:@"PoweredOn\n"];
            isWork=TRUE;
            break;
        default:
            [nsmstring appendString:@"none\n"];
            break;
    }
    NSLog(@"%@",nsmstring);
    [delegate didUpdateState:isWork message:nsmstring getStatus:cManager.state];
}
```

依照centralManagerDidUpdateState來結果來判斷iDevice是否支援BLE，畢竟BLE是在iphone 4s、New iPad之後才有的，可以根據此項來決定APP的功能或其他提示使用者的動作。

scanForPeripheralsWithServices

確定周邊支援BLE且運作正常後，我們就要來開啟BLE搜尋功能來尋找BLE的週邊，當週邊接收到搜尋功能的廣播時就會在一定時間內回覆，在此我們可以設定一個2秒的Timer來停止scan

```
CBCentralManager *CM = [[CBCentralManager alloc] initWithDelegate:self queue:nil];
[CM scanForPeripheralsWithServices:nil options:options];
[NSTimer scheduledTimerWithTimeInterval:2.0f target:self selector:@selector(scanTimeout:) userInfo:nil repeats:NO];
```

設定2秒後觸發執行scanTimeout method，再將scanForPeripheralsWithServices的值設為nil，代表搜尋的Service type不受限制，當你只是要搜尋特定的對向時可以將它的UUID填入，像：

```
        NSArray *uuidArray= [NSArray arrayWithObjects:[CBUUID UUIDWithString:@"180D"], nil];
        [CM scanForPeripheralsWithServices:uuidArray options:options];
```

其中「UUIDWithString:@"180D"」的180D就是*Heart Rate* Service type，如果有指定Service type，則結果就會將週邊有*Heart Rate*一一列出來，想要知道這類的Service Type可以到Bluetooth官網 (http://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx) 查詢。

一互了解Service type是哪一種類型時就可以來做對應的流程及資料的解析，也可以製作出符合標準週邊的APP。

didDiscoverPeripheral

didDiscoverPeripheral這是屬於Delegate，所以要按照它預設的宣告將要處理的過程寫在裡面，格式如下：

```
-(void)centralManager:(CBCentralManager *)central didDiscoverPeripheral:(CBPeripheral *)peripheral advertisementData:(NSDictionary *)advertisementData R
{

//處理過程

}
```

advertisementData會報告可以連線的週邊內容，像這樣：

```
adverisement:{
    kCBAdvDataLocalName = "INFOS 4090v35.05";
    kCBAdvDataServiceUUIDs =     (
        "Unknown (<fff0>)"
    );
    kCBAdvDataTxPowerLevel = 0;
}
```

RSSI是訊號的強度，是以NSNumber Object存在，整個範例可以是這樣子：

```
-(void)centralManager:(CBCentralManager *)central didDiscoverPeripheral:(CBPeripheral *)peripheral advertisementData:(NSDictionary *)advertisementData R
```

```
{

    NSMutableString* nsmstring=[NSMutableString stringWithString:@"\n"];
    [nsmstring appendString:@"Peripheral Info:"];
    [nsmstring appendFormat:@"NAME: %@\n",peripheral.name];
    [nsmstring appendFormat:@"RSSI: %@\n",RSSI];

    if (peripheral.isConnected){
        [nsmstring appendString:@"isConnected: connected"];
    }else{
        [nsmstring appendString:@"isConnected: disconnected"];
    }
    NSLog(@"adverisement:%@",advertisementData);
    [nsmstring appendFormat:@"adverisement:%@",advertisementData];
    [nsmstring appendString:@"didDiscoverPeripheral\n"];
    NSLog(@"%@",nsmstring);

}
```

結果輸出：

```
2013-02-25 14:43:17.243 gw-health-01[141:907]
Peripheral Info:NAME: INFOS 4090v35.05
RSSI: -69
isConnected: disconnected
adverisement:{
    kCBAdvDataServiceUUIDs =     (
        "Unknown (<fff0>)"
    );
}
```

發現可連線的BLE週邊就會不斷的執行*didDiscoverPeripheral* 這個Delegate，所以要利用這個狀況將每次發現BLE週邊存入Array，再提供給USER選擇，或是從中可以去判斷某個特別的週邊是否存在而決定要不要連線。

stopScan

執行*scanForPeripheralsWithServices* 掃描周邊時設定一個2秒的Timer，當時間到時就停止scan，一般來說2秒內無反應就可以當作是沒有其他週邊回應，承上面scanForPeripheralsWithServices， 將stopScan寫在scanTimeout裡面：

```
- (void) scanTimeout:(NSTimer*)timer
{
    if (CM!=NULL){
                [CM stopScan];
    }else{
        NSLog(@"CM is Null!");
    }
    NSLog(@"scanTimeout");
}
```

connectPeripheral

利用didDiscoverPeripheral所得到的BLE週邊列表讓User選擇要連線的BLE，再將 CBPeripheral傳入connectPeripheral進行連線，格式：

```
      [CBCentralManager connectPeripheral:CBPeripheral* options:NSDictionary*]
```

在此將它包裝成一個connect Method，

```
- (void) connect:(CBPeripheral*)peripheral
{

       if (![peripheral isConnected]) {
                [CM connectPeripheral:peripheral options:nil];
       connectedPeripheral=peripheral;
       }

}
```

option傳入nil，connectPeripheral傳入Method connect的值。

didConnectPeripheral

執行connectPeripheral之後並連線成功後就會引發*didConnectPeripheral*的Delegate：

```
-(void)centralManager:(CBCentralManager *)central didConnectPeripheral:(CBPeripheral *)peripheral
{

```

```
:
:
}
```

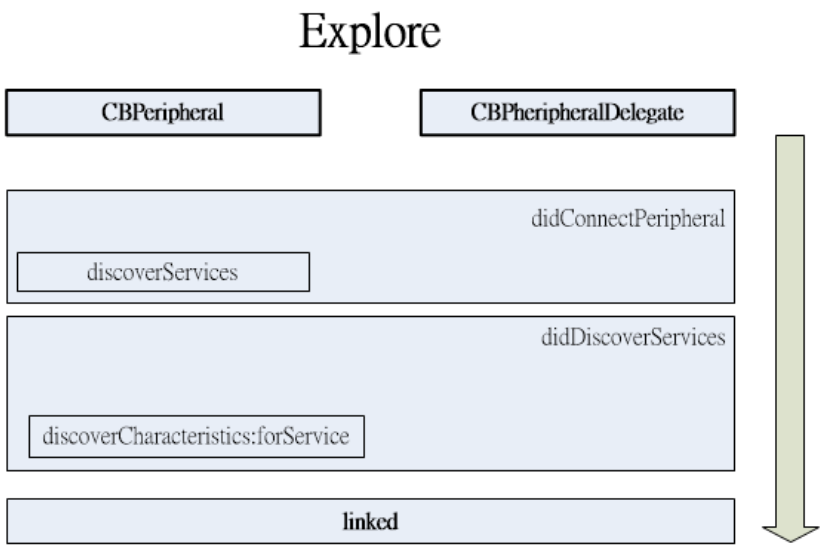
在這裡有個重點，當連線成功後引發Delegate時，就必需要針對其*CBPeripheral*來馬上進行*discoverServices*的動作，去了解週邊提供什麼樣的Services

```
-(void)centralManager:(CBCentralManager *)central didConnectPeripheral:(CBPeripheral *)peripheral
{
    NSLog(@"Connect To Peripheral with name: %@\nwith UUID:%@\n",peripheral.name,CFUUIDCreateString(NULL, peripheral.UUID));
    peripheral.delegate=self;
    [peripheral discoverServices:nil];//一定要執行"discoverService"功能去尋找可用的Service

}
```

執行discoverServices之後又會引發另一個Delegate叫「didDiscoverServices」，接下來則會在Explore中介紹。

Explore



Discover/Connect 中使用CBCentralManager進行連線/搜尋BLE周邊的功能，連線之後需要靠的是CBPeripherall來傳送/接收資料。

CBPeripheral

使用CBPeripheral使需要設定Delegate才可以進行連線的動作，加上之前加入的CBCentralManagerDelegate，CODE就變成：

```
@interface DYCoreBluetooth : NSObject<CBCentralManagerDelegate, CBPeripheralDelegate> {

:
:
:

}
```

之後連線的重點全都是在Delegate的互動，查看Service Type或是有什麼樣的Services可以提供。

didConnectPeripheral

前面有稍為介紹didConnectPeripheral，這是在連線成功後就會引發的Delegate，但一定要在這裡執行一些Method才可以順利的引發另一個CBPeripheral的Delegate去查看有什麼樣的Services

```
-(void)centralManager:(CBCentralManager *)central didConnectPeripheral:(CBPeripheral *)peripheral
{
    NSLog(@"Connect To Peripheral with name: %@\nwith UUID:%@\n",peripheral.name,CFUUIDCreateString(NULL, peripheral.UUID));

    peripheral.delegate=self;
    [peripheral discoverServices:nil];//一定要執行"discoverService"功能去尋找可用的Service
```

```
}
```

例子中已經將「 peripheral.delegate=self 」，所以接下來進行 peripheral的任何動做引發的Delegate都在這個Object中，執行「discoverServices] Method，讓它去尋找Services，一找到Services就又會引發「didDiscoverServices 」Delegate，這樣我們就可以了解有什麼Services。

didDiscoverServices

從這裡開始就是最關鍵

```
- (void)peripheral:(CBPeripheral *)peripheral didDiscoverServices:(NSError *)error {
   NSLog(@"didDiscoverServices:\n");
   if( peripheral.UUID == NULL  ) return; // zach ios6 added
   if (!error) {
       NSLog(@"====%@\n",peripheral.name);
       NSLog(@"========== %d of service for UUID %@ ==========\n",peripheral.services.count,CFUUIDCreateString(NULL,peripheral.UUID));

       for (CBService *p in peripheral.services){
           NSLog(@"Service found with UUID: %@\n", p.UUID);
           [peripheral discoverCharacteristics:nil forService:p];
       }

   }
   else {
       NSLog(@"Service discovery was unsuccessfull !\n");
   }

}
```

peripheral.services.count 會知道有多少個Services，在每個Servces中還會有Characteristics需要了解，所以會針對每個Service來執行 *peripheral discoverCharacteristics: forService:*去了解每個Service下有多少個Characteristics提供傳送/接收的溝通，在執行*discoverCharacteristics*時也引發了*didDiscoverCharacteristicsForService* Delegate，最後再由*didDiscoverCharacteristicsForService*真正的判斷什麼樣的Service什麼樣的Characteristics再進行處理之後收到的資料，例如： 發現**2A37**的Characteristic，就要進行註冊通知，到時候BLE週邊發訊息過來才會在收到當下得到資料。

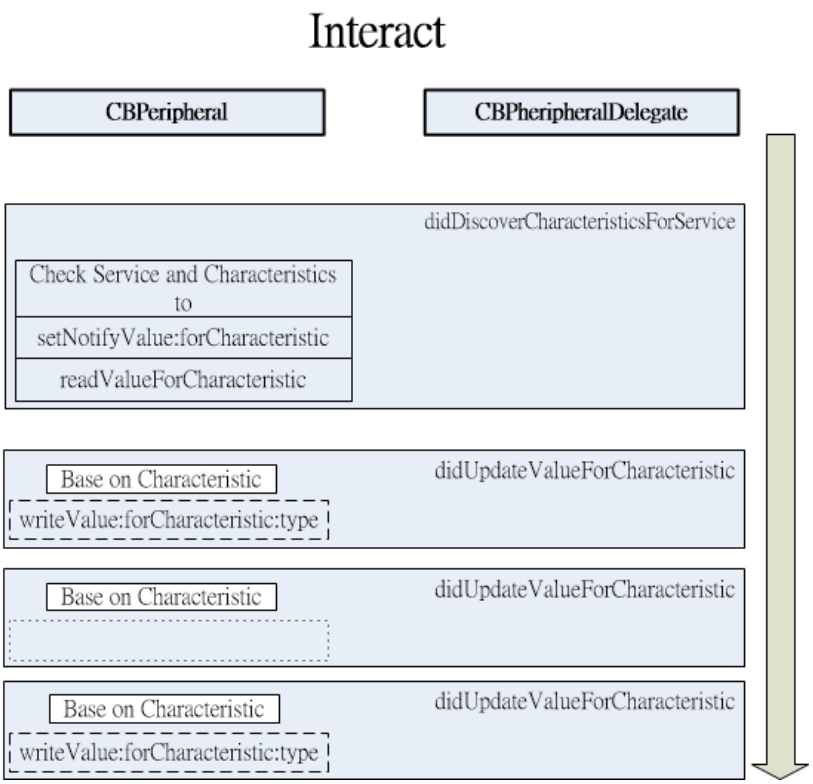
didDiscoverCharacteristicsForService

整個最關鍵的地方就是這個Delegate，程式架構如下：

```
-(void)peripheral:(CBPeripheral *)peripheral didDiscoverCharacteristicsForService:(CBService *)service error:(NSError *)error
{
:
:
:

}
```

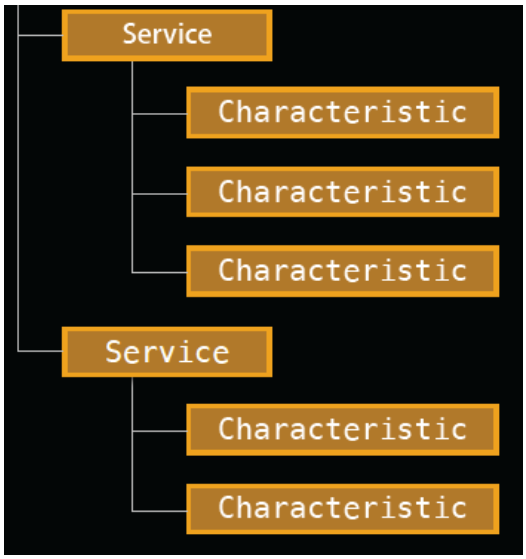
Interact



最後完成*didDiscoverCharacteristicsForService*之後，整個連線過程就算是完成，之後的*didUpdateValueForCharacteristic*是整個資料接收的主要Delegate，經過接收到的資料引發Delegate進行即時處理就可以取得BLE週邊的訊息，再使用*writeValue*的Method寫入資料到BLE週邊，整個BLE的最基本使用方式就大致上完成。

didDiscoverCharacteristicsForService

由Apple提供的資料擷取某部分來了解架構，等下程式就是利用這架構去一一尋訪所有的*CharacteristicsForService*



每樣Servic下都會有很多的Characteristics，Characteristics是提供資料傳遞的重點，它會有個UUID編號，再由這 個編號去Bluetooth (http://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx) 官方查表得到是哪種資料格式，再將資料解開加以使用。

真正的例子：

```
-(void)peripheral:(CBPeripheral *)peripheral didDiscoverCharacteristicsForService:(CBService *)service error:(NSError *)error
{

    CBService *s = [peripheral.services objectAtIndex:(peripheral.services.count - 1)];
    NSLog(@"=========== Service UUID %s ===========\n",[self CBUUIDToString:service.UUID]);
    if (!error) {
        NSLog(@"=========== %d Characteristics of service ",service.characteristics.count);

        for(CBCharacteristic *c in service.characteristics){
```

```
          NSLog(@" %s \n",[ self CBUUIDToString:c.UUID]);
        //  CBService *s = [peripheral.services objectAtIndex:(peripheral.services.count - 1)];
          if(service.UUID == NULL || s.UUID == NULL) return; // zach ios6 added


          //Register notification
          if ([service.UUID isEqual:[CBUUID UUIDWithString:@"180D"]])

          {
              if ([c.UUID isEqual:[CBUUID UUIDWithString:@"2A37"]])
              {
                  [self notification:service.UUID characteristicUUID:c.UUID peripheral:peripheral on:YES];
                  NSLog(@"registered notification 2A37");
              }
              if ([c.UUID isEqual:[CBUUID UUIDWithString:@"2A38"]])
              {
                  [self notification:service.UUID characteristicUUID:c.UUID peripheral:peripheral on:YES];
                  NSLog(@"registered notification 2A38");
              }
              if ([c.UUID isEqual:[CBUUID UUIDWithString:@"2A39"]])
              {
                  [self notification:service.UUID characteristicUUID:c.UUID peripheral:peripheral on:YES];
                  NSLog(@"registered notification 2A39");
              }

          }

      }
      NSLog(@"=== Finished set notification ===\n");


  }
  else {
      NSLog(@"Characteristic discorvery unsuccessfull !\n");

  }
  if([self compareCBUUID:service.UUID UUID2:s.UUID]) {//利用此來確定整個流程都結束後才能設定通知
      [delegate didConnected:peripheral error:error];
      NSLog(@"=== Finished discovering characteristics ===\n");
      //全部服務都讀取完畢時才能使用！

  }

}
```

例子是以Heart Rate(180D) (http://developer.bluetooth.org/gatt/services/Pages/ServiceViewer.aspx?u=org.bluetooth.service.heart_rate.xml)

Heart Rate	org.bluetooth.service.heart_rate	0x180D	Adopted

Heart Rate來說，0x2A37 (http://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicViewer.aspx?u=org.bluetooth.characteristic.heart_rate_measurement.xml) 可以得到心跳的數據，所以針對此項可以註冊通知來得到每次的心跳數據更新。

*notification*是我自行撰寫的Method，可以從Service UUID及Characteristic UUID來找到Service與Characteristic的Object Point，才可以符合註冊通知的格式

```
  [(CBPeripheral *)p setNotifyValue:(BOOL) forCharacteristic:CBCharacteristic *)]
```

將Characteristic的Point傳入並設定*setNotifyValue:on*就完成註冊通知，之後如果有更新資料時就會引發*didUpdateValueForCharacteristic* Delegate，再進行資料處理。

- notification

notification可以直接copy下程式碼重覆使用相當方便。

```
-(void) notification:(CBUUID *) serviceUUID characteristicUUID:(CBUUID *)characteristicUUID peripheral:(CBPeripheral *)p on:(BOOL)on {

  CBService *service = [self getServiceFromUUID:serviceUUID p:p];
  if (!service) {
      if (p.UUID == NULL) return; // zach ios6 addedche
      NSLog(@"Could not find service with UUID on peripheral with UUID \n");
      return;
  }
  CBCharacteristic *characteristic = [self getCharacteristicFromUUID:characteristicUUID service:service];
  if (!characteristic) {
      if (p.UUID == NULL) return; // zach ios6 added
      NSLog(@"Could not find characteristic with UUID  on service with UUID  on peripheral with UUID\n");
      return;
  }
  [p setNotifyValue:on forCharacteristic:characteristic];

}


-(CBService *) getServiceFromUUID:(CBUUID *)UUID p:(CBPeripheral *)p {

  for (CBService* s in p.services){
      if ([self compareCBUUID:s.UUID UUID2:UUID]) return s;
  }
```

```
    return nil; //Service not found on this peripheral
}

-(CBCharacteristic *) getCharacteristicFromUUID:(CBUUID *)UUID service:(CBService*)service {

    for (CBCharacteristic* c in service.characteristics){
        if ([self compareCBUUID:c.UUID UUID2:UUID]) return c;
    }
    return nil; //Characteristic not found on this service
}
```

didUpdateValueForCharacteristic

*didUpdateValueForCharacteristic*在連線完成後顯的非常重要，範例中有比對2個UUID為2A37 (http://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicViewer.aspx?u=org.bluetooth.characteristic.body_sensor_location.xml) 與2A38 (http://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicViewer.aspx?u=org.bluetooth.characteristic.heart_rate_measurement.xml)

```
-(void)peripheral:(CBPeripheral *)peripheral didUpdateValueForCharacteristic:(CBCharacteristic *)characteristic error:(NSError *)error
{

    if ([characteristic.UUID isEqual:[CBUUID UUIDWithString:@"2A37"]])
    {
        if( (characteristic.value)  || !error )
        {

        }
    }

    if ([characteristic.UUID isEqual:[CBUUID UUIDWithString:@"2A38"]])
    {
        //set refresh int
        uint8_t val = 1;
        NSData* valData = [NSData dataWithBytes:(void*)&val length:sizeof(val)];
        [peripheral writeValue:valData forCharacteristic:characteristic type:CBCharacteristicWriteWithResponse];
    }

}
```

針對這兩個UUID成立時做對應的工作，這裡以2A38來解說一下。

- 2A38

從程式中的2A38來說，當更新資料為2A38時，程式將直接寫入 *1* ，也就是在下表中可以了解到，*1*所代表的就是*Chest*

Home > GATT Specifications > Characteristics > CharacteristicViewer

The material contained on this page is informative only. Authoritative compliance information is contained in the applicable Bluetooth specification.

Name: Body Sensor Location

Type: org.bluetooth.characteristic.body_sensor_location

Assigned Number: 0x2A38

Value Fields

Names	Field Requirement	Format	Minimum Value	Maximum Value	Additional Information
Body Sensor Location	Mandatory	8bit	N/A	N/A	Enumerations

Key	Value
0	Other
1	Chest
2	Wrist
3	Finger
4	Hand
5	Ear Lobe
6	Foot
7 - 255	Reserved for future use

意思是告訴心跳感測器量測的位置是在胸部的部分。

- 注意事項

整個*didUpdateValueForCharacteristic*在處理時請注意資料格式的解釋，往往是因為格式解釋錯誤才會得到不正確的資料。

ReConnect

相機

GPS

電池

晃動

距離

Retrieved from "http://mediweb.35g.tw:9000/~web/wiki/index.php?title=硬體週邊Tips&oldid=353"
