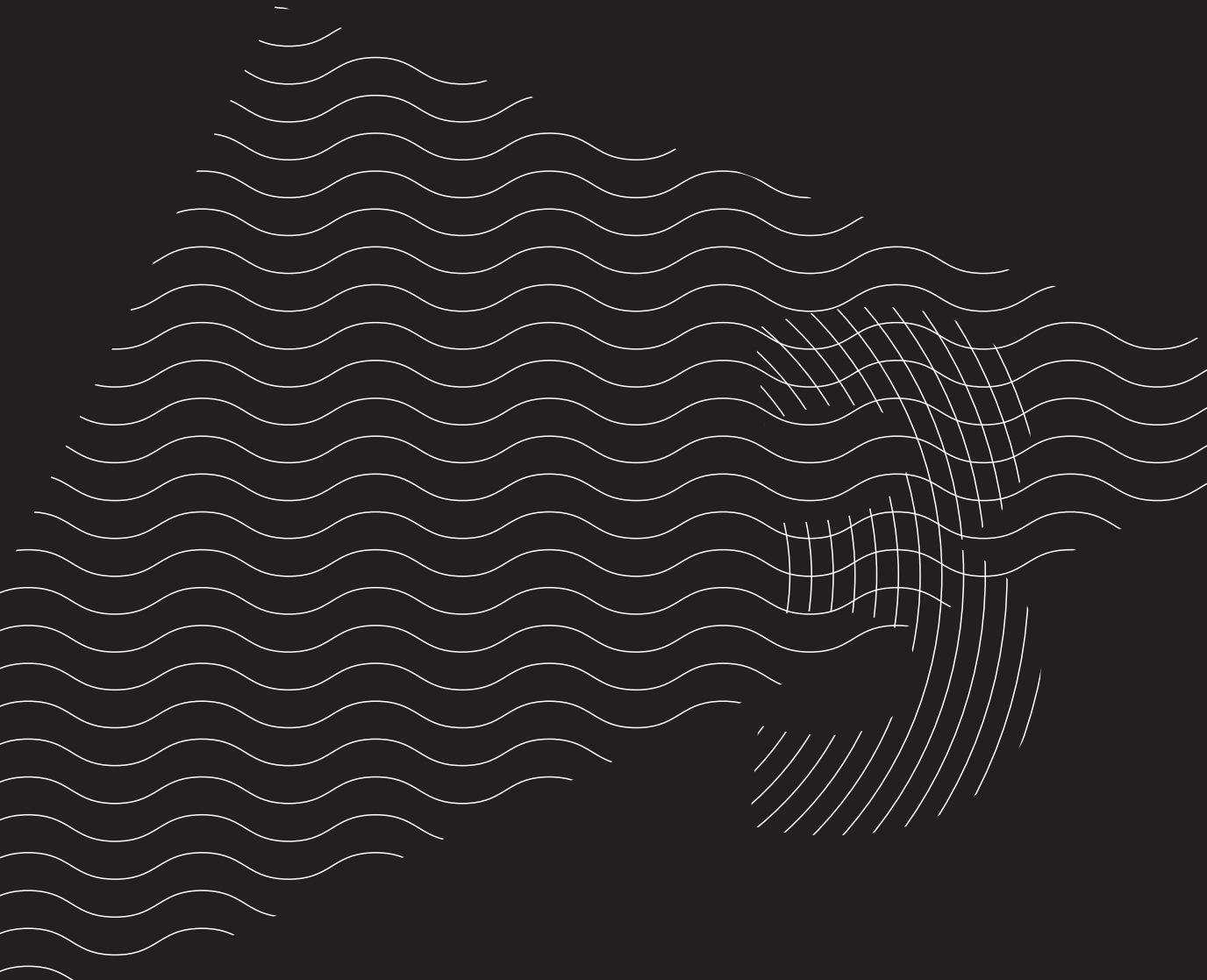


Part1. BLE 編 | 3 章

BLE を理解する

本章では、BLE の規格をわかりやすく解説していきます。
iOS デベロッパーとしてどう仕様を読み解けばいいかも随所に明記していますので、
実装に役立てることができます。

(松村礼央)



BLE (Bluetooth Low Energy) はBluetooth ver.4.0において新しく組み込まれた無線規格です。その「低電力性」と「サービス多様性」が読者であるiOS開発者の方々にとって魅力であること、そして従来のBluetooth ver.3.0の規格と異なるものであることは、1章で解説したとおりです。本章ではiOSアプリケーションからBLEを扱うにあたり、知識と用語について解説するとともにその仕組みについて概説します。

本章の対象とするBluetoothのバージョンおよび定義

バージョン

本章で解説の対象とするBluetoothのバージョンはver.4.1とし、Core Specification ver.4.1、Supplement to the Bluetooth Core Specification ver.4.0に従うものとします。なお、2015年1月の執筆時点で、Bluetooth ver.4.2が公開されており、最新の情報はBluetooth Technology Special Interest Groupのサイトで確認することができます。^{※1}

Bluetooth固有の用語について

本章で紹介する用語のうち、Core Specificationsで扱うBluetooth固有の英語の用語については、そのまま英語にて表記を行います。他方、固有でない英語の用語についてはカタカナもしくは日本語訳で記述しています。

パケットサイズの単位「octet」

本章では、BluetoothのCore Specificationsの表記に則り、パケットサイズの単位として**octet** (= 8 bit) 表記で解説します。8bitは通常「byte」で表現されることが多いのですが、処理系によっては8bitではない場合があります。標準化が十分とはいえません。そのため、通信分野ではその曖昧さを除外する目的で、8bit = 1octetであると標準化されたoctetを単位として利用する傾向があり、Bluetooth Core Specificationもこれに則っています。

対象外の項目

本章では、SMP (Security Manager Protocol) およびHCI (Host Controller Interface) に関しては解説の対象外とします。SMPについてはペアリング、セキュリティ関連の項目が該当し、HCIについてはBLEに用いる半導体やシステム構成などの項目が該当します。SMPおよびHCIについては、『Bluetooth Low Energy: The Developer's Handbook』(Robin Heydon 著)の一読を推奨します。

※1 <https://www.bluetooth.org/ja-jp/specification/adopted-specifications>

3-1. BLEの概要

BLEの仕組みの要となるのは、**GATT (Generic Attribute Profile)** と呼ばれるプロファイルです（「プロファイル」については3-2節で解説します。ここでは、とりあえず**機能や振る舞い**であると理解していただければ問題ありません）。

1章では、BLEを「さまざまな食材を適度な価格で売買できる市場」に、通信でやり取りされるデータは「食材」に、そのデータをやり取りするために消費した消費電力は文字どおり「代金（コスト）」にたとえました。GATTを同様に表現すると「食材の陳列するブースと店員」です。食材の並べ方、分類の仕方、そして、そこでの売買の仕方、これらの機能を市場において提供するのがGATTの役割となります。

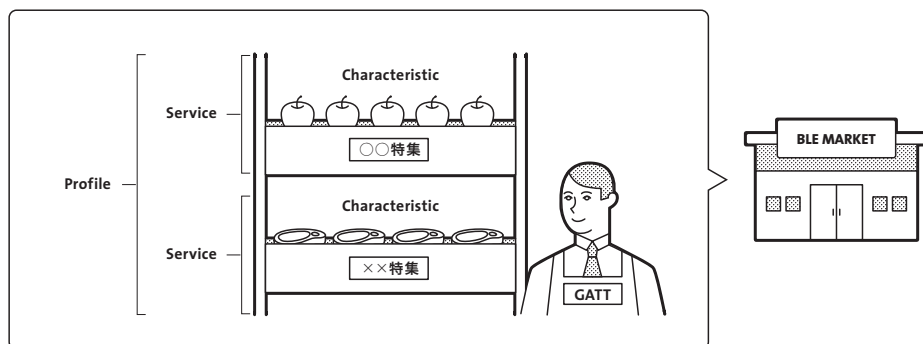


図 3-1 BLEの仕組み

市場の中ではBLEデバイス側、iOSデバイス側のどちらでも売り手もしくは買い手になることができます。では、それらの役割はどうやって決められているのでしょうか。当然、好き勝手に役割を申請してなれるわけではありません。BLEでは、市場の中での役割を決め調整する「市場の管理人」といえる機能が存在します。この機能を担っているのが**GAP (Generic Access Profile)**です。BLEで何らかの通信を行う場合は、**GAP**によってその振る舞いを調整してもらうことで周囲のデバイスとの通信が可能になります。

そして「陳列棚の利用方法」や「役割の申請の仕方」などの一連の手続きやフォーマットが**プロトコル**と呼ばれています。BLEでは、ATT、SM、L2CAP、LL、PHYという形で階層化さ

れた状態で、さまざまなプロトコルが決められていますが、一般的に、アプリケーション側からこのプロトコルを直接扱うことはありません。しかし、プロトコルの勘所をある程度理解していれば、市場で何が扱えるのか、流通の仕組みがどうなっているのか、どうやればより低コストで食材を陳列できるのか、などが理解できるようになり、iOSでのBLEがより楽しくなるはずです。

BluetoothとMFiプログラム

クラシックBTでは音楽、音声など実時間性、連続性が重要となるサービスを主体に扱い、そのプロファイルはサービスの品質を保つためペアリングを必須とし、その機能はプロファイルとしてiOSに組み込まれています。したがって通常、iOSアプリケーションがクラシックBTと直接やりとりすることはできません。たとえば、クラシックBTのBluetoothヘッドセットをiOSと接続した場合を考えます。ペアリングされたヘッドセットは各種アプリからの音が鳴りますが、iOSアプリケーションはあくまでオーディオファイルの再生を行っているのみで、クラシックBT側のデバイス进行操作・制御しているわけではありません。そしてiOSアプリケーション側からこのようなクラシックBTの操作を直接行う場合、必要となるのがMFi (Made for iPhone) プログラムです。MFiプログラムに参加することで独自のクラシックBTの開発ができるようになります。しかしながら、この契約には多額の費用が必要であり、必ずしも大きな組織に属しているとは限らないiOSエンジニアにとっては非常に大きい壁となっています。

他方、BLEで外部のデバイスと接続するiOSアプリケーションを開発・ストアで頒布する場合、MFiは必須ではありません（ただし、ペアリングが必要なデバイスとBLEでiOSアプリケーションを連携させる場合やMFiのロゴをデバイスに掲載する場合などはMFiプログラムへの参加は必須となります）。

3-2. BLEの構造

BLEの構造について説明する前に、冒頭でも登場した**プロトコル**と**プロファイル**という用語について整理しておきます。

プロトコル

プロトコルは、デバイス間での情報をやり取りする際の手順、ルール、データ構造を定めたものです。たとえば、某バラエティ番組の「英語禁止ボーリング」と呼ばれる企画をご存知でしょうか。これはボーリングを行う際の会話の中で英単語を口にすると、その回数に応じて罰金が課せられるという企画ですが、この「英単語を口にした回数に比例して罰金を払う」という取り決めがこの企画に対するプロトコルとなります。

さて、一般的に機器の通信では、そのデバイスがあらゆるデバイスで成立するように、アプリケーション・ソフトウェア・ハードウェアを横断する形で階層的にプロトコルが定められています。Bluetooth（クラシックBTおよびBLE）においても同様で、階層的にプロトコルが定められています。このように階層化されたプロトコルの総称を**プロトコルスタック**もしくは**アーキテクチャ**と呼び、これを確認すれば通信規格の仕組みとその概要を知ることができます。

プロファイル

通信があらゆるデバイスで成立するように…ということで決められているプロトコルですが、Bluetoothでは少々複雑です。たとえば、クラシックBTが扱う通信のアプリケーションはプリンタの印刷、ヒューマンインターフェース（マウスやキーボードなど）、オーディオインターフェース（ヘッドセットやスピーカーなど）など、多岐に渡ります。さまざまなアプリケーションに対応する一方で、アプリケーションごとにどのプロトコルがどのように必要なのか分かりづらいという難点があります。

そのため、Bluetoothではアプリケーションごとにプロトコルをまとめ、どのように振る

1

2

3

4

5

6

7

8

9

10

11

12

舞うかを定義し、標準化しました。これが**プロファイル**です。つまり、通信を行う機器間で同じBluetoothのプロファイルに対応できていれば、そのプロファイルが提供する機能とアプリケーションを無線通信で利用できるようになります。これをBluetoothでは**相互運用性 (interoperability)**と呼びます。

Bluetoothではさまざまなプロファイルが定められています。HID、A2DP、HFPなどのプロファイルは、パソコンの周辺機器としてよく利用するため、Bluetooth機器を利用する際によく聞くのではないのでしょうか。逆に、GAPなどのように、Bluetoothの動作に深く関与していても、非常に重要なプロファイルであっても、その動作がアプリケーションよりも下位の階層で行われているため、その存在がiOS開発者などのアプリケーション側に対して表だっていないものも多くあります。

3-2-1. BLEのプロトコルスタック（アーキテクチャ）

それでは、BLEのプロトコルスタックを見ていきましょう。BLEのプロトコルスタックを図3-2に示します。

図からわかるように、BLEのプロトコルスタックは下位から順に**Controller層**、**Host層**そして、**Application層**という3つの大きな階層によって構成されています。

まず、Controller層は**LE Physical (PHY)**、**Link Layer (LL)**の2つのプロトコルによって構成されています。この2つは無線接続に関する制御を司る回路、およびそのソフトウェア群によって実装されています。

Controller層の上位に位置するのが、Host層です。Host層は3種類のプロトコル**L2CAP (Logic Link Control & Adaptation Protocol)**、**SMP (Security Manager Protocol)**、**ATT (Attribute Protocol)**と、2種類のプロファイル**GATT (Generic Attribute Profile)**、**GAP (Generic Access Profile)**によって構成されています。

Host層のプロトコルとプロファイルはすべてソフトウェアであり、上位のApplication層に対するAPIとして働き、内部の処理を制御します。

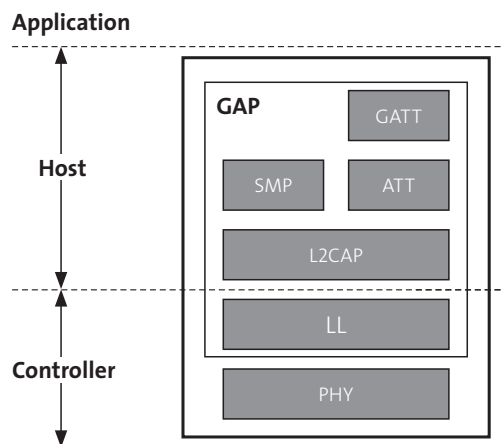


図 3-2 BLEのプロトコルスタック

プロトコルスタックにおいて最上位に位置するのがApplication層です。Application層は、その文字のとおり、ユーザーの作成するアプリケーション、つまり本書においてはiOSアプリが相当します。この層は、BLEでどのようなサービスを実現するかによって実装内容が異なるため、その設計はiOSエンジニアやハードウェアエンジニアに委ねられます。

BLEの構造を理解するにあたり、iOSエンジニアがとりあえず押さえておくべきプロトコル、プロファイルは次の3つです。

- ATT (Attribute Protocol)
- GAP (Generic Access Profile)
- GATT (Generic Attribute Profile)

以降では、まずこの3つのプロトコル、プロファイルを簡単に紹介します。

3-2-2. ATT (Attribute Protocol)

ATT (Attribute Protocol) は、Attributeと呼ばれる独自の単位をベースにデータのやり取りを行うサーバ/クライアント型のプロトコルです。BLEを市場とたとえると「食材のやり取りに関する最小単位の取り決め」と理解すればよいでしょう。

クライアントとして振る舞う ATT (ATTクライアント) は、L2CAP層を経由してサーバとして振る舞う ATT (ATTサーバ) に対して通信を行います。基本的には、ATTクライアントから通信の対となったデバイスの ATTサーバ上に展開された Attribute ベースのデータベースに対し、リード/ライトを行うのが、BLEにおけるデータのやり取りの基本となります。そして、このやり取りにおけるデータベースの構造や役割 (サーバ/クライアント) についてその振る舞いを管理しているのが、後述する GATT (Generic Attribute Profile) となります。

iOSエンジニアから見た場合、この GATTを通して ATTで規定された Attribute単位のデータベースのやり取りを行うことが、iOSデバイスとBLEとの通信における核といえます。

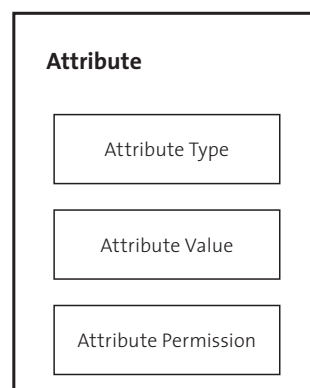


図 3-3 Attribute のデータ構造

Attribute

では、ATTが規定するAttributeと呼ぶデータ単位とはどのようなものでしょうか。Attributeは、ATT上でやり取りするデータの最小単位であり、データの値の他に **Attribute**、**Type**、**Attribute Handle**、**Permission** の3つのプロパティを持つデータの構造体として定義されています。Attributeのデータ構造を図3-3に示します。Attributeの詳細については3-9-2項で紹介します。

3-2-3. GAP (Generic Access Profile)

GAP (Generic Access Profile) は、3.1節でも解説したように、BLEという市場の中でデバイスがどのような役割 (Role) で振る舞うのかを管理する「市場の管理人」となるプロファイルです。「管理人」として機能するだけあり、すべてのBluetoothに実装されている基盤となるプロファイルです。つまりクラシックBTにおいてもGAPは実装されています。

BLEにおいては、管理人として **Broadcaster**、**Observer**、**Peripheral**、**Central** の合計4種類の役割を管理しており、Host層のL2CAP、SMP、ATT、Controller層のLL、PHYの各プロトコルの機能を、デバイスが実現するサービスに沿って一貫通貫に結びつけることで提供しています。この役割については3-8節で詳細を紹介します。



図3-4 GAPの役割

プロファイルの階層構造

GAPはすべてのBluetoothデバイスに実装される基盤となるプロファイルです。したがって、他の追加実装されるプロファイルはすべて、必ずGAPの上位集合となるプロファイルとなります。つまり、あらゆるBluetoothデバイスのサービスにおいて、あるプロファイルは必ずGAPを内包したプロファイルとなります。このプロファイルの階層構造を図3-5に示します。

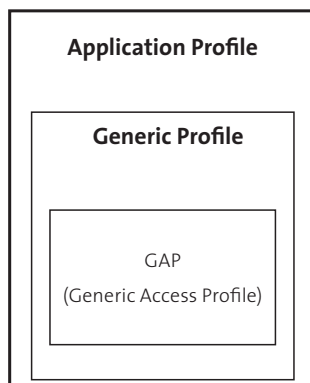


図3-5 プロファイルの階層構造

3-2-4. GATT (Generic Attribute Profile)

GATT (Generic Attribute Profile) は ATT (Attribute Protocol) を基盤としたプロファイルで、Application側もしくはそれ以外のプロファイルから利用されます。GATTでは、ATTが提供するAttributeを最小単位とした階層的なデータベースの構築と、そのデータベース上のやり取り (ATTによるサーバ/クライアント動作) の制御を行います。BLEを市場だとすると、GATTのデータベースは「食材を陳列するブース」、振る舞いを制御する様は「ブースを管理する店員」と考えることができます。つまるところ、GATTはiOSとの実質的な通信の窓口となるわけです。

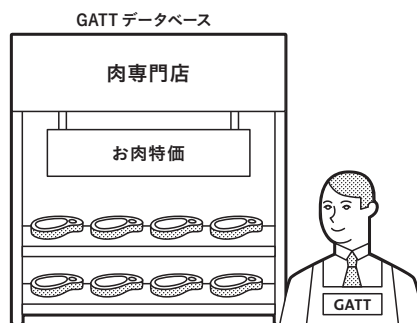


図3-6 GATTの役割

ここで一点、注意すべきは、GATTが制御する振る舞いにおける役割 (サーバもしくはクライアント) はGAPによって制御されるのではなく、GATTをベースとしたプロファイルの設計者、すなわちより上位のApplicationである、ということです。GAPが管理するのはあくまでBLEという市場全体としてどのような役割を担うかであり、GATTが提供する市場と外部とのやり取りにおける役割については上位のApplicationに委ねられます。

Attributeを用いた階層的なデータベース

GATTでは「食材を陳列するブース」を、Attributeを最小単位とする階層的なデータベースで実現します。データベースの構造を図3-7に示します。図からもわかるように **Service** と **Characteristic** という新しい概念が登場します。Service は、Characteristicやその他のServiceへの参照子（Reference）もしくは後述するInclude）により構成される構造体です。Characteristicは値（Value）、値の属性（Property）、そして値のディスクリプタ（Descriptor）による構造体となります。

図3-7のとおり、構造的にはServiceにCharacteristicが、Characteristicに値、属性、ディスクリプタが包含される形となり、この値、属性、ディスクリプタそれぞれがAttribute単位で管理・格納されています。

このデータ構造を「食材の陳列するブース」と考えると、「食材」となるのがCharacteristicです。その食材の実態を表すのが値、販売元に関する情報や食材がどのようなカテゴリのものなのかを示したのが属性、そして、その成分や含有物を示したものがディスクリプタといえます。

ここでServiceはさしずめ「カレーセット（じゃがいも、にんじん、たまねぎのパック）」や「カレールー（調合済みのスパイスの固まり）」といったお料理パックというところでしょう。いくつかのCharacteristicをパッケージ化してコンテンツをもたせたものがServiceとして定義されています。したがって、Serviceは必ず1つ以上のCharacteristicで構成されます。

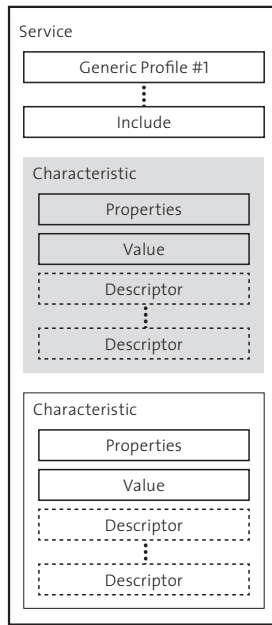


図3-7 GATTサーバ上のデータ構造

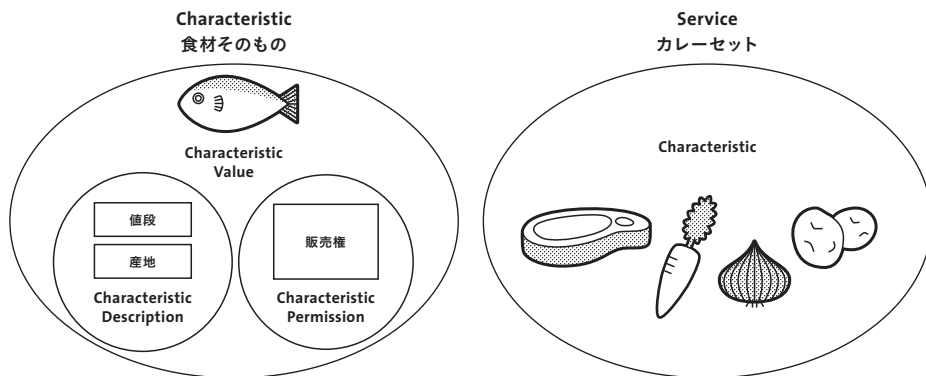


図3-8 CharacteristicとService

GATTをベースとしたプロフィール

「食材を陳列するブース」がGATTによるデータベースで提供されることがわかりました。あとは「何を売買するか」つまり「市場でどのようなサービスを提供するか」が問題となります。BLEでは、この「何を売買するか」を、GATTプロフィールをベースとしたプロフィールを作成することで実現します。このプロフィールは図3-10のように、Serviceを含むさらに上位の構造体として定義されます。したがって、GATTをベースとしたプロフィールは必ず1つ以上のService、もしくはServiceへの参照子（これをIncludeと呼びます）によって構成されます。

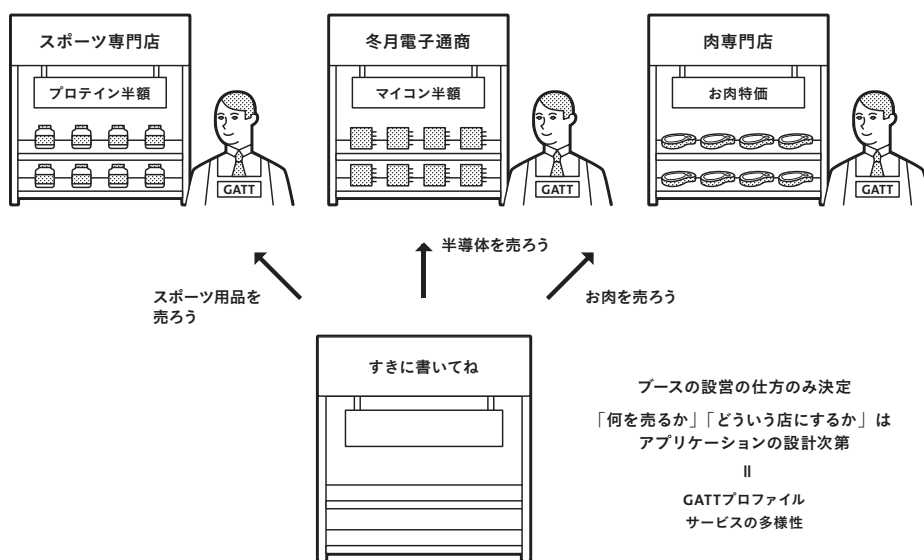


図3-9 GATTベースのプロフィールによって何を提供するかが決まる

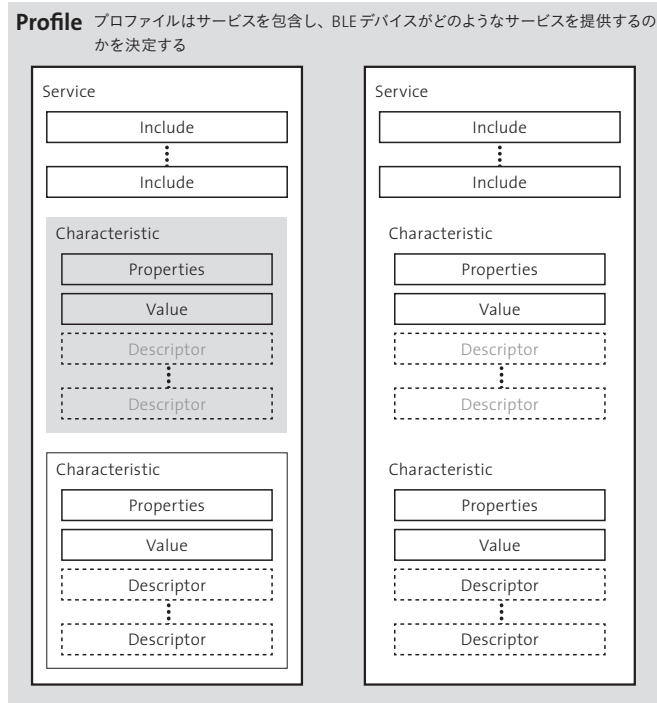


図 3-10 GATT ベースのプロファイル

BluetoothにおけるGATT

ちなみに、GATTはクラシックBTでも利用可能なプロファイルです。したがって、クラシックBTにおいても、BLEと同様にGATTをベースとしてプロファイルを設計することが可能です。クラシックBTとBLEで異なるのはGATTに対する立場です。BLEにおいては、GATTが必須である一方、クラシックBTにおいては必ずしも必要ではありません。

それでは、少しずつBLEについて踏み込んでいきましょう。

3-3. BLEのネットワークトポロジー

3-3節では、BLEデバイス同士がどのように接続・通信するのか、その形態（ネットワークトポロジー）を解説します。BLEネットワークトポロジーの形は、大きく**ブロードキャスト型**、**接続型**の2つに分けることができます。本節では、この2つは何が異なるのか、その違いを順に解説していきます。

3-3-1. ブロードキャスト型トポロジー

ブロードキャスト型のトポロジーを図3-11に示します。ブロードキャスト型は、あるデバイスが周囲の他のデバイスに自身の存在や情報を伝えるために利用するもので、片方向、かつ1対多数の形をとります。1章で「BLEとは市場である」とたとえましたが、同様にたとえるとブロードキャスト型は「試供品や広告の配布」といえます。

ここで配布される「試供品や広告」に相当するデータをBLEでは**Advertising Packet**（もしくは**Advertising Data**）と呼び、その名のとおり自身の情報を周囲に伝達する広告として機能します。なお、このブロードキャスト型のトポロジーにおいて Advertising Packet を周囲に配布するイベントをBLEでは**Advertising**、Advertising Packetを受信するイベントを**Scanning**と呼びます。

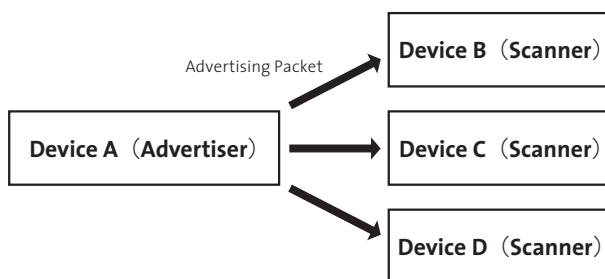


図 3-11 ブロードキャスト型トポロジー

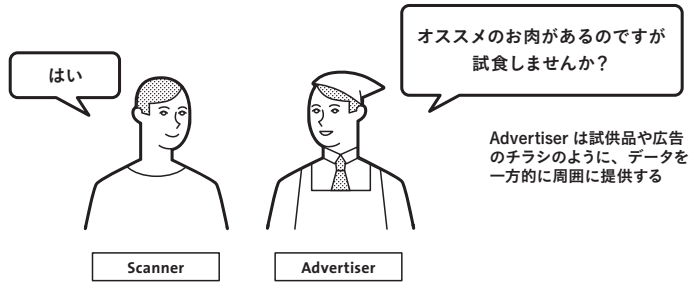


図 3-12 ブロードキャスト型は「試供品や広告の配布」

3-3-2. 接続型トポロジー

接続型のトポロジーを図3-13に示します。接続型は、通信する相手を特定して双方向での通信を行うものです。接続型は通信の形態として当初 (Bluetooth ver.4.0) は1対1と規定されていましたが、Bluetooth ver.4.1以降では複数のデバイスと接続型トポロジーを持つことが可能となり、1対多数の通信が可能になりました。1章と同様にたとえると、接続型のトポロジーは、店員と顧客が代金のやり取りをする双方向の通信モデルといえます。

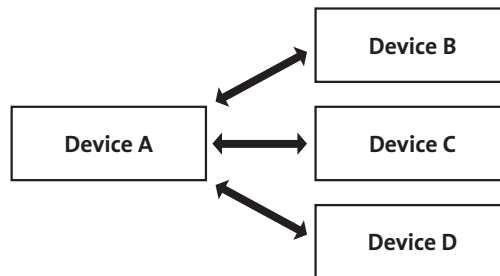


図 3-13 接続型トポロジー

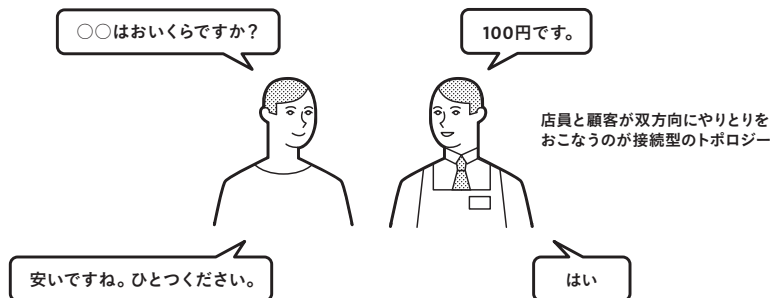


図 3-14 接続型トポロジーは店員と顧客のやり取り

3-4. BLE でのネットワークと通信の制御

3-3節ではBLEデバイスがどのようにネットワークを形成し、通信するのかを概説しました。本節では**BLEでネットワークと通信がどのように実現されているか**を、プロトコルスタックのControllerに属するLL層(Link Layer)とPHY層(LE Physical)に焦点を当てて解説し、ネットワークの基本となるAdvertisingについても解説します。

3-4-1. Bluetoothの無線仕様とPHY層(LE Physical)

Bluetoothは無免許で利用できる2.4GHz ISM帯域を利用した無線通信規格です。BluetoothのPHY層での鍵となるのが「周波数シフト・キーイング」という通信の変調の方式と「周波数ホッピング・スペクトル拡散」と呼ばれる通信方式の2つです。PHY層について述べる本項では、この2つの方式を足がかりにBLEの無線仕様について解説します。BLEの無線仕様について、概略を表3-1に示します。

周波数帯域	2.4GHz (2.400~2.4835GHz)
変調方式	GFSK (ガウシアン周波数シフト・キーイング)
データレート	1Mbps
チャンネル数	40ch (Advertising Channel =3ch. / Data Channel=37ch.)
最大出力	10.00mW (+10dBm)
最小出力	0.01mW (-20dBm)

表 3-1 BLEの無線仕様

周波数シフト・キーイング（FSK：Frequency-Shift Keying）

無線通信ではデータのやり取りを電波で行います。いくつかある電波のやり取り方法のうち、Bluetoothが採用しているのが、周波数シフト・キーイング（FSK：Frequency-Shift Keying）という方法です。FSKという手法では、データのやり取り、つまり「0/1」の情報のやり取りを、電波の周波数の「高い/低い」によって行います。図3-15に、FSKによるデータのやり取りの概要を示します。この「0/1」の情報を周波数の「高い/低い」に変換して通信を行うことを**周波数変調**と呼びます。特に、変調において0/1のデジタル信号を変調して通信を行うことを**キーイング（keying）**と呼び、デジタル信号を周波数で変調（Shifted）するキーイングということで**周波数シフト・キーイング**という名称となっています。なお、BLEでのFSKの変調速度は1Mbpsと規定されており、これがBLEにおける通信速度の理論限界となります。

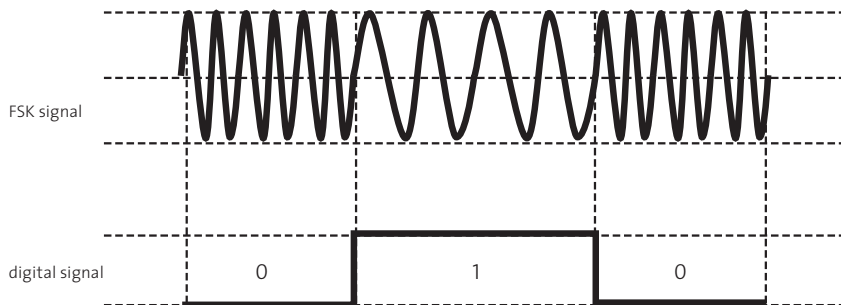


図 3-15 周波数シフト・キーイング

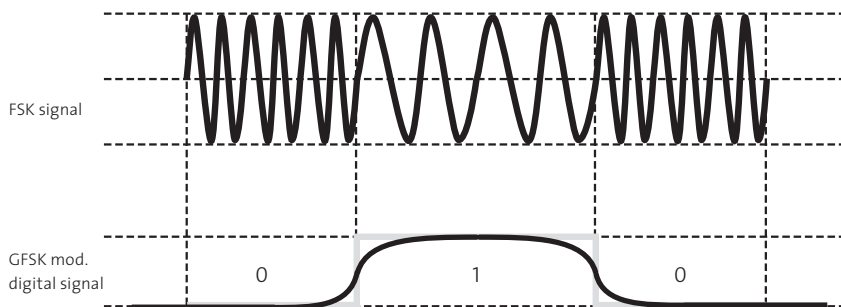


図 3-16 ガウシアン周波数シフト・キーイング

Bluetoothで利用されているFSKは、これにさらに工夫を加えた**ガウシアン周波数シフトキーイング（GFSK：Gaussian Frequency-Shift Keying）**という方法です。この方法では、やり取りする信号にガウシアンフィルタでフィルタ処理をかけることで、高周波成分を除去し、その除去後の信号で周波数変調を行うというものです。元の信号の高周波成分が除去されているので、

通常のFSKによる通信よりも通信中に高周波成分の除去分だけ、通信で占有する周波数の領域が狭くなっている点が特徴です。なぜ、占有する周波数を狭くするのかという点は、Bluetoothの通信におけるもう1つのトピックである、**周波数ホッピング・スペクトル拡散**が関係しています。

周波数ホッピング・スペクトル拡散 (FHSS : Frequency Hopping Spread Spectrum)

FSKで無線通信を行う場合、同一の周波数を利用する他のFSK通信の無線デバイスがあると原理的に通信の干渉を避けることはできません。Bluetoothの場合、IEEE802.11b、IEEE802.11g、IEEE802.11nなどが2.4GHz帯を利用するため、これらの機器に対して何らかの干渉対策が必要となります。これ以外にも信号自身による干渉(やまびこが重なって聞こえる現象が無線通信にも起こり得ます)への対策や無線通信の信号の盗聴への対策の必要性があります。これらの対策として利用されているのが、**周波数ホッピング・スペクトラム拡散 (FHSS : Frequency Hopping Spread Spectrum)**と呼ばれる技術です。

FHSSの技術のアイデアは非常にシンプルです。たとえば、他の機器との通信の干渉問題についてのFHSSのアイデアは「同一の周波数で変調された信号が干渉するのであれば、**その周波数を互いに逐次切り替えながら通信を行えば、干渉のリスクが抑えられるのではないか**」というものです。

これは厳密には異なりますが「渋滞している道路」を考えるとわかりやすいかもしれません。同じスタート地点から目的地を目指す車があるとして、1つの道路しかないとしします。すると渋滞を起こしてしまい、最悪、目的地に所定の日時に到着できなくなるでしょう。このとき、「車」が無線通信で送受信するデータ、「道路」がFSKの変調周波数となり、「渋滞」が通信の干渉といえます。そしてこの問題に対して、同じ目的地に到達する複数の道路を作り、適宜、別の道に車を誘導することで道路の混雑をなくすという解決方法をとっているのが、FHSSという技術になります。さらに、この「交通の整理などの管理を行って車を目的地に届ける」という役割を担っているのがPHY層といえます。

さてFHSSにおいて、切り替えられるいくつかの「道路」すなわち変調周波数をそれぞれ物理チャンネルと呼びます。また、この物理チャンネルを切り替える動作を**ホップ (Hop)**と呼びます。Bluetoothでは、2台のデバイス間でこのHopをそれぞれ示し合わせながら、同じ物理チャンネルで常に通信をすることで、干渉に対して頑健な無線通信を実現しています。

BLEの無線通信の物理チャンネル

冒頭でも述べたように、Bluetoothでは2.4 GHz ISM帯で動作します。その際の周波数帯、およびチャンネルと中心周波数の関係は次のようになります。

Bluetoothの周波数帯	RFチャンネルとRF中心周波数の関係式
2400.0 ～ 2483.5 MHz	$f = 2402.0 + k \text{ MHz} (k = 0, 1, 2, \dots, N)$

表3-2 BLEの周波数帯、およびチャンネルと中心周波数

BLEではFHSSで利用する物理チャンネルを40本と定義しています。この40本の物理チャンネルのうち、3本はブロードキャスト型のネットワークでの通信、すなわちAdvertising専用 to 確保しています。そして、残りの37本を接続型のネットワークでのデータ通信専用に利用しています。

BLEにおける各チャンネルの中心周波数とチャンネルIDについて表3-3に示します。ちなみに、この表でならぶ「RF」とは「Radio Frequency」の略で「無線周波数」を意味しています。無線周波数という表現は若干直訳的にはなりますが、理解の上では問題はありません。

RFチャンネル	RF中心周波数	チャンネルタイプ	Data チャンネルID	Advertising チャンネルID
0	2402MHz	Advertisingチャンネル	×	37
1	2404MHz	Dataチャンネル	0	×
2	2406MHz	Dataチャンネル	1	×
...	...	Dataチャンネル	...	×
11	2424MHz	Dataチャンネル	10	×
12	2426MHz	Advertisingチャンネル	×	38
13	2428MHz	Dataチャンネル	11	×
...	...	Dataチャンネル	...	×
37	2422MHz	Dataチャンネル	35	×
38	2424MHz	Dataチャンネル	36	×
39	2426MHz	Advertisingチャンネル	×	39

表3-3 FHSSで利用する物理チャンネル

データ通信専用チャンネルのホップ

データ通信専用チャンネルでは37本のチャンネルを逐次ホップさせながら、通信を行うこととなります。BLEではデータ通信専用チャンネルのホップは以下の式に従います。ここで unmappedChannel はホップ後のチャンネルID、lastUnmappedChannel はホップ前のチャンネルID、hpIncrement はホップ数を表します。

$$\text{unmappedChannel} = (\text{lastUnmappedChannel} + \text{hpIncrement}) \bmod 37$$

hpIncrement は、デバイス間で Connection が確立した際に生成される値です。したがって、新しい Connection が確立するたびに新しい値が割当てられます。

3-4-2. LL層（Link Layer）

LL層（Link Layer）は、3-3節で示したネットワーク中でのデバイスの状態（State）を制御し、PHY層を通じて無線通信を実現することが目的となります。LL層は、表3-4に示す5種類の状態をシーケンス制御によって管理するステートマシンです。

LL層では内部に少なくとも1台の Advertising State か Scanning State のいずれかをサポートするステートマシンを持ち、場合に応じて複数のステートマシンのインスタンスを内部に持つことができます。ただし、複数のステートマシンを実装する場合は特定の状態の組み合わせについて制限があり、ハードウェアエンジニアは実装時に配慮する必要があります。状態の組み合わせについての注意事項は後述する3-4-3項を参照してください。

状態	内容
Standby State	休止状態
Advertising State	Advertisingに関するイベントを扱う状態
Scanning State	Scanningに関するイベントを扱う状態
Initiating State	接続の開始状態を扱う状態
Connecting State	BLE デバイス同士の接続に関する状態

表 3-4 FLL 層の5種類の状態

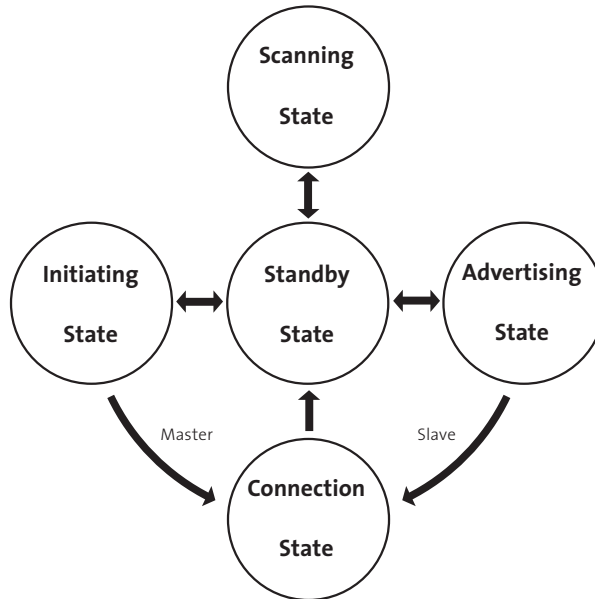


図 3-17 LL 層 (Link Layer) のステートマシンの概略

Standby State

Standby State は、いかなるデータの送信も受信も行わない、いわゆる休止状態です。図 3-17 に示すとおり、Standby State へは他のいずれの状態からでも遷移することができるよう設計されており、いかなる状態でも休止状態に移行できるように配慮されています。

Advertising State

Advertising State では Advertising Packet の送信や、そのレスポンスに対してリクエスト/レスポンスなどの対応を行います。ここでのレスポンスは、接続型ネットワークにおけるデバイス間での接続要求なども含まれます。図 3-17 に示すとおり Advertising State は Standby State から遷移することができます。また、Advertising State 状態のデバイスを指して **Advertiser** と呼びます。

Scanning State

Scanning State では、Advertising State のデバイスからの Advertising を受信し、その情報を収集します。図 3-17 に示すとおり、Scanning State は Standby State から遷移できます。また、Scanning State 状態のデバイスを指して **Scanner** と呼びます。

Initiating State

Initiating State では、周辺の BLE デバイスから Advertising を受信し、他のデバイスとの接続を確立します。図 3-17 に示すとおり、Initiating State は Standby State から遷移できます。また、Initiating State のデバイスを指して **Initiator** と呼びます。

Connection State

Connection State は、Initiating State もしくは Advertising State から遷移できる状態で、2 つのデバイスの接続が確立されている状態です。この Connection State では **Master** そして **Slave** の 2 つの役割が定義されています。デバイスが Connection State においていずれの役割となるかは、直前の状態で決定します。

Initiating State から遷移した場合は **Master**、一方、Advertising State から遷移した場合は **Slave** になります。Connection State に遷移後は、Master であるデバイスは Slave のデバイスと通信を行うとともにデータ送信のタイミングを管理します。他方、Slave のデバイスは Master のデバイスのリクエストに応じて通信を行います。

3-4-3. LL 層における注意点

上述したとおり、LL 層では、場合に応じて複数のステートマシンのインスタンスを内部に持つことができます。ただし、複数のステートマシンが実装される場合、状態の組み合わせについて、次のようにいくつかの注意事項があります。

- ① Connection State においてデバイスは同時に Master と Slave になることができる

- ② Connection StateでSlaveであるデバイスは、他の複数のデバイスと接続することができる
- ③ Connection StateでMasterであるデバイスは、他の複数のデバイスと接続することができる
- ④ 上記の①~③を満たす他のすべての状態と役割の組み合わせは、仕様上、すべてサポートが許可される

②、③については、Bluetooth ver.4.1で追加・修正された注意事項で、Bluetooth ver.4.0までは接続型のトポロジーにおいて1台のデバイスが複数のデバイスと接続することが許されていませんでした。したがって、BLEデバイスを設計する際、通信モジュールがBluetooth ver.4.1以降か否かで可能な接続のトポロジーが変化するため注意が必要です。なお、LL層の実装において必ずしもすべての状態を実装する必要はありません。つまり Advertising Stateのみ、Scanning Stateのみという実装も、仕様上は許可されています。

3-4-4. Bluetooth Device Address

LL層でデバイス間の接続を行う Connection State について述べました。本項では、接続の際にデバイスを同定するために用いる Bluetooth Device Address (BD_ADDR) を解説します。BD_ADDR は Public device address と Random device address の2種類があり、デバイスのアドレスにはいずれか一方が利用されます。いずれのアドレスも 48bit 長で定義されます。この BD_ADDR は、イーサネットにおける MAC アドレスと捉えるとわかりやすいでしょう。

Public Device Address

Public device address は、IEEE Registration Authority から与えられた組織固有識別子(OUI)を利用し、IEEE 802-2001 Standard Section 9.2 “48 bit universal LAN MAC addresses” にしたがって生成されます。Public device address は以下の2つの領域で構成されています。bitの構成を図3-18に示します。48 bitのうち、LSBからの24bitは company_assigned 値が、その後の24 bitには company_id 値が並びます。

Public device address	
LSB	MSB
company_assigned (24 bit)	company_id (24 bit)

図 3-18 Public Device Address

Random Device Address

Random Device Address は、Static address と Private address の 2 種類に分かれます。また Private address は、さらに Non-resolvable address と Resolvable private address の 2 種類に分かれます。

Static Address

Static Address は 48bit でランダムに生成されるアドレスで、以下の要件を満たすアドレスです。Static Address の構成を図 3-19 に示します。

- MSB から見て最初の 2 桁は必ず 1 となる
- Static Address の乱数部分がすべて 1 になることはない
- Static Address の乱数部分がすべて 0 になることはない

Static Address (48bit)		
LSB	MSB	
Static Address の乱数部分 (46bit)		1
		1

図 3-19 Static address

Static Address は起動ごとに新しいアドレスを生成します。したがって、デバイスの電源投入後、デバイスはこの Static Address を変更することはできません。

Non-Resolvable Private Address

Private Address のうち、以下の要件を満たすアドレスを Non-Resolvable Private Address と呼びます。Non-Resolvable Private Address の構成を図 3-20 に示します。なお、この Address はデバイスとの接続においてプライバシーに配慮するモードを利用する際に利用されます。本書では詳細は追いません。詳細については Core Specification v4.1 を参照ください。

1

2

3

4

5

6

7

8

9

10

11

12

- MSBからみて最初の2桁は必ず0となる
- Non-Resolvable Private Addressの乱数部分がすべて1になることはない
- Non-Resolvable Private Addressの乱数部分がすべて0になることはない
- このAddressがStatic Addressと同値になることはない
- このAddressがPublic Addressと同値になることはない

Non-Resolvable Private Address (48bit)			
LSB		MSB	
Non-Resolvable Private Addressの乱数部分 (46bit)		0	0

図 3-20 Non-Resolvable Private Address

Resolvable Private Address

Private Addressのうち、以下の要件を満たすアドレスをResolvable Private Addressと呼びます。Resolvable Private Addressの構成を図3-21に示します。このAddressはデバイスとの接続においてセキュリティに配慮するモードを利用する際に利用されます。本書では詳細は追いません。詳細についてはCore Specification v4.1を参照ください。

Resolvable Private Address (48bit)			
LSB		MSB	
hash (24bit)	prandの乱数部分 (22bit)	1	0

図 3-21 Resolvable Private Address

3-4-5. AdvertisingとScanning

3-3節では、BLEがサポートするブロードキャスト型、接続型の2つのネットワークトポロジーを紹介しました。BLEの動作はこれら2つのネットワークに従う通信が基本となりますが、この中でも重要となるのが**Advertising**と**Scanning**です。本項ではAdvertisingとScanningについて、その概要を解説します。

ブロードキャスト型のネットワークでの通信では周辺のBLEデバイスに対して接続することなく、自身が保有する情報をAdvertising Packetで配信/受信します。一方で接続型のネットワークでも、あるデバイスが周囲のデバイスとの接続を確立する前、つまり通信先の相手が未

知の状態相手先に自身を通知する、もしくは相手先を検出するために利用します。

なぜ、双方向の通信で単方向の Advertising を利用するのか、という疑問がわくかもしれませんが、これは無線通信であることに起因しています。無線通信では、接続が確立していない状態のデバイスでは、通信対象がどこにいるのか、どれだけいるのか、そして誰なのかがわかりません。これはたとえば、読者のみなさんが目隠しをしている状態で、同様に周囲の目隠しをした人とコミュニケーションを図ることを考えるとわかりやすいと思います。目隠しをしているので、当然、最初の段階ではどこに誰がいるのか皆目見当がつかないはずです。この状況を打破するには、自分が声を発するか、相手から声を発してもらうしかありません。この点は BLE を理解する上で非常に重要な要素なので、以降も念頭においておきましょう。

Advertising と Advertising チャンネル

上述したとおり、Advertising イベントでは物理チャンネル40本のうちの3本(Ch.37、Ch.38、Ch.39)を利用し、これを Advertising チャンネルと呼びます。では、この3本をどのように利用して BLE では Advertising を行っているのでしょうか。本項では3-3節でも少し触れた Advertising について掘り下げていきたいと思います。

Advertising Interval

Advertising は、ブロードキャスト型のネットワーク通信での周囲への情報発信、接続型のネットワーク通信でのデバイスの発見など、いずれのトポロジーでも利用している通信イベントです。BLE では、このイベントの周期 ($T_{advEvent}$) を次のように定めています。

$$T_{advEvent} = advInterval + advDelay$$

ここで、advInterval を Advertising Interval と呼び、デバイスの Advertising Packet の発信周期を表します。このインターバルはデバイスの設計者が仕様で定められた範囲の中で自由に設定することができます。仕様では、0.625ms 刻みで 20ms から 10.24s までの範囲と規定されています。しかしながら、3-6 節で後述する Advertising イベントの4種のうち、非接続・スキャン禁止型 Advertising イベント (ADV_NONCONN_IND) と非接続・スキャン可能型イベント (ADV_SCAN_IND) のいずれか一方である場合、advInterval は 100ms 未満にすることは禁止されています。

また、ここで advDelay は、Advertising に埋め込む遅延で 0ms から 10ms の範囲をとります。

このadvDelayはLL層によってイベントごとに生成される擬似乱数値となります。

AdvertisingイベントとT_advEvent、advInterval、advDelayとの関係を図3-22に示します。このとき、Advertisingイベントで消費する時間とadvIntervalが異なる点に注意が必要です。また、advDelayはイベント毎に異なる値にも注意が必要です。

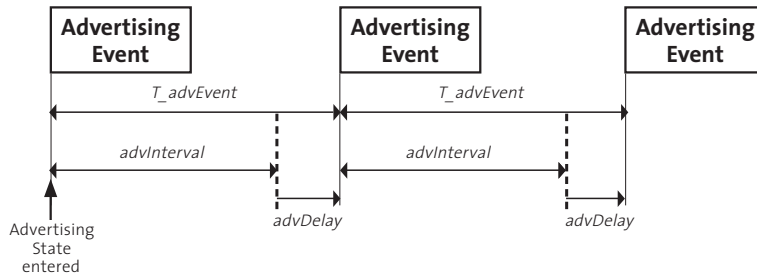


図 3-22 Advertising イベントと T_advEvent、advInterval、advDelay との関係

Advertising イベントと Advertising チャンネル

Advertising イベント中、デバイスは3本のAdvertisingチャンネルを逐次切り替えながら（Hopしながら）通信を行っています。仕様では各チャンネルの通信のうち、最初の連続した2つのチャンネルの通信が10ms以下と定められています。

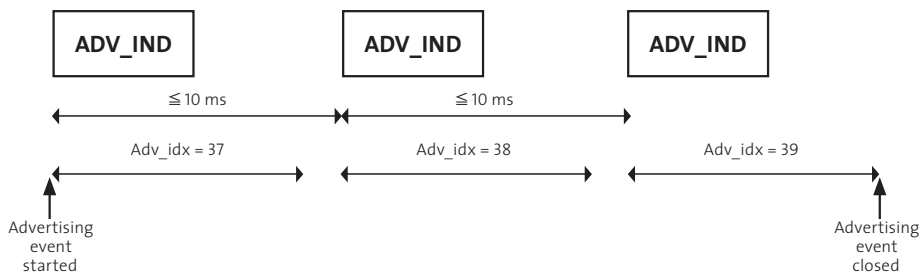


図 3-23 Advertising イベントと Advertising チャンネル

Scanning

ScanningはScanning StateであるデバイスがAdvertising Packetを受信するために利用する動作です。デバイスはScanningを行うことで、周囲に存在するデバイスの存在を知ることができます。Scanningにおいては、タイミングや物理チャンネルの選択に対して厳格なルールは設

けられていません。以下に示す設定値に従ってデバイス、アプリケーション、そしてサービスの設計者が仕様の範囲で任意に決定することができます。本項では、Scanningの概要やどのようなパラメータがあるのかを見ていきましょう。

Scanning状態に入ったデバイスのLL層は一定の期間、物理チャンネルからの Advertising Packetの受信を待ちます。この期間をscanWindowと呼びます。つぎにデバイスは、この受信待ちの期間（scanWindow）を一定の周期で繰り返します。この繰り返しの周期、すなわちscanWindow間の待ち時間をscanIntervalと呼びます。scanWindowとscanIntervalに関する概略を図で示すと図3-24のようになります。このscanWindowとscanIntervalは10.24s以下の値をとります。また、図3-24の関係からもわかるように、必ずscanWindowはscanInterval以下の値をとることとなります。仮にscanWindowとscanIntervalがHost側から同じ値にセットされた場合、LL層はScanning動作を連続して常に行うこととなります。

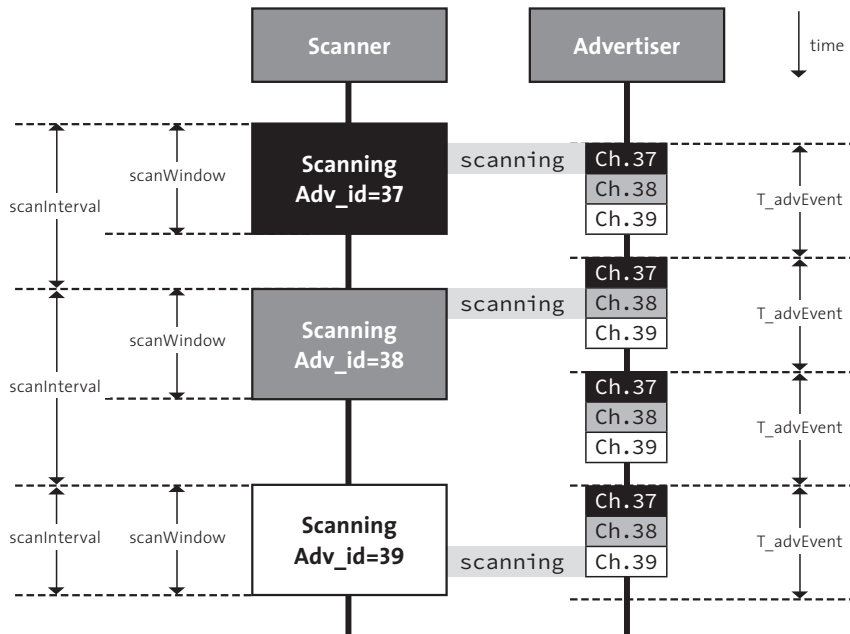


図3-24 scanWindowとscanInterval

BLEではScanningとして2種類のモード、Passive ScanningとActive Scanningを定めています。以降では、その振る舞いの違いについて見ていきます。

Passive Scanning

Passive ScanningではScannerとして動作するデバイスは、Scanning時においてAdvertising

Packetを常に受信するのみで、いかなるPacketも Advertiser に対して返信を行いません。Passive Scanning の概略を図3-25 に示します。

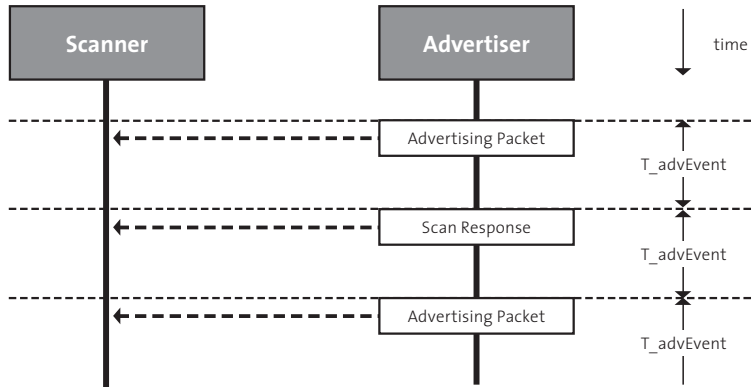


図 3-25 Passive Scanning

Active Scanning

Active Scanning では Passive Scanning とは一部挙動が異なります。Scanning時に Advertising Packetを受信する振る舞いはPassive Scanningと同様ですが、Active Scanningの場合は受信の後、さらに Advertiser に対して情報の提供を求めるパケットを送出します。Scannerからパケットを送出し、さらなる情報を Advertiser から引き出すことから、この Scanning 方式を Passive Scanning と区別して Active Scanning と呼びます。Active Scanning の概略を図3-26 に示します。

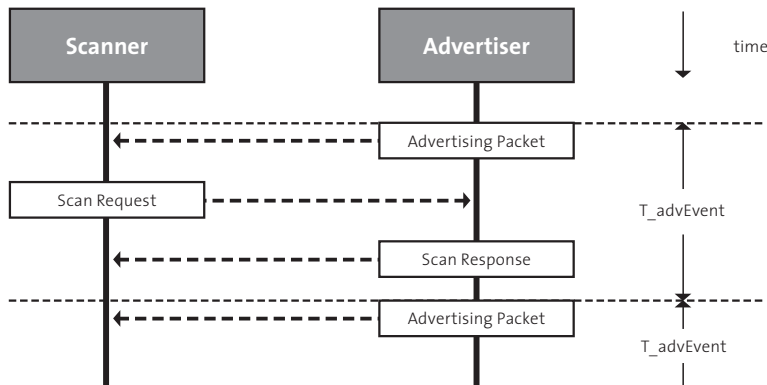


図 3-26 Active Scanning

ここで注意すべきは、Active ScanningでScannerがAdvertiserに対して何らかのビーコンを発信して **Advertising** の要求をするわけではないという点です。これはBLEの省電力化に起因

します。仮に、ScannerがAdvertiserに対してビーコンを発してAdvertising Packetを要求すると、GAPによってPeripheralやObserverの役割として振る舞う周辺デバイスは、常にそのビーコンに対して待機状態を維持しなければなりません。こうなると、周辺デバイスは常に待機状態を強いられることとなり、電力の管理という観点で主導権を失うこととなります。

このため、ネットワーク上でハブとなるデバイスは周辺に対して常に待機状態を強いられることになります。BLEにおいては、そのようなデバイスはiOSデバイスなどであるとの想定がされており、ハブとなるデバイスは周辺のデバイスに比して充分な動作時間が確保できるバッテリを持つはずという仮定のもと、周辺のデバイスの低消費電力化が優先されています。

なお、BLEではこのような2種類のScanningによって周辺のデバイスの情報を獲得し、周辺のデバイスを発見しますが、周辺のデバイスの存在が既知の場合は直接そのデバイスに接続することも可能です。この機能については、「3-6-2. ADV_DIRECT_IND」で詳細に述べます。

Advertising PacketとScan Response Packet

Advertising/Scanningでやり取りされるデータをAdvertising Packet（もしくはAdvertising Data）とScan Response Packet（もしくはScan Response Data）と呼びます。BLEのパケットフォーマットについては3-6節で詳しく解説しますが、この項では、Advertising Packet/Scan Response Packetの概要について先に簡単に解説します。

Advertising Packet/Scan Response Packetの概要を図3-27に示します。これらPacketはおおまかに分けてAdvertisingに関する有意なデータを送信する部分（Significant part）と0で埋められたダミーデータを送信する部分（Non-significant part）の2つの領域に分けられ、全体で31octetのデータサイズが与えられています。Significant partのAdvertisingに関する有意なデータはいくつかのデータブロックに分かれており、このブロックを各々AD structureと呼びます。

AD structureはLengthとDataの2つの領域で構成されており、Data領域にAdvertising/Scanningに関係するデータが含まれています。Length領域は1octetで続くData領域のデータ長を示しています。Data領域はLengthで指定されたデータ長となっており、さらにAD TypeとAD Dataの2つの領域に分かれています。AD TypeとAD Dataには、Advertising/Scanningのモードや動作に応じて、適切な値が与えられます。

AD Typeは、Bluetooth SIGのSpecifications/Assigned Numbers^{※2}にて掲載されています。iOSで利用頻度が高いと思われるAD Typeの一例を表3-5に示します。FlagsやLE Roleなどの詳細

※2 <https://www.bluetooth.org/ja-jp/specification/assigned-numbers/generic-access-profile>

についてはSupplement to the Bluetooth Core Specification（コア仕様補完（CSS））^{※3}にて詳細な仕様が与えられているのでそちらも参照ください。

AD Type	値	内容
Flag	0x01	BluetoothのPHY層のチャンネルのサポート情報
Incomplete List of 16-bit Service Class UUIDs	0x02	Service UUIDとそのフォーマット内容
Complete List of 16-bit Service Class UUIDs	0x03	
Incomplete List of 32-bit Service Class UUIDs	0x04	
Complete List of 32-bit Service Class UUIDs	0x05	
Incomplete List of 128-bit Service Class UUIDs	0x06	
Complete List of 128-bit Service Class UUIDs	0x07	Local Nameを示す
Shorted Local Name	0x08	
Complete Local Name	0x09	送信出力に関する値。Tx Power LevelからRSSIの値を引いた値が損失を意味する
Tx Power Level	0x0A	
Device ID	0x10	DeviceのIDを示す
LE Role	0x1C	GAPの役割 (Role) のサポート情報

表 3-5 AD Type の一例

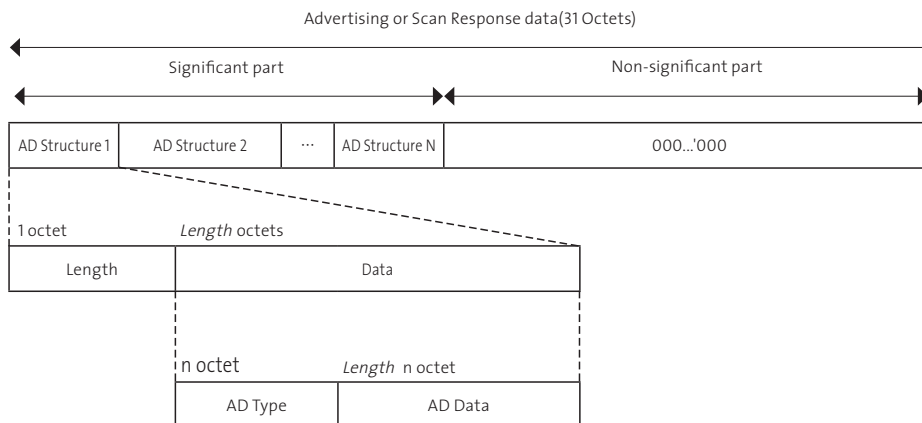


図 3-27 Advertising Packet/Scan Response Packet

※3 <https://www.bluetooth.org/ja-jp/specification/adopted-specifications>

Non-significant part は Significant part 以後に現れるデータで、Advertising Packet/Scan Response Packet が全体で 31 octet となるように 0 でデータを埋められています。なお、Significant part と Non-significant part の領域の境界は、AD Structure で Length の値が 0 になる点で生まれます。

この Advertising Packet/Scan Response Packet は送信することでデバイス間に特定の Advertising イベントを発生させることができます。Advertising については ADV_IND、ADV_NONCONN_IND、ADV_SCAN_IND が、Scan Response Packet については SCAN_RSP がそれぞれ発生します。

3-4-6. Connection

3-4-1 項でも述べたように、Advertising によって発見されたデバイスに対して接続を行う処理を Connection と呼びます。Connection は、Initiation State であるデバイスがリクエストを Advertising State のデバイスに送信するか、またはその逆で、Advertising State のデバイスが送信されたリクエストを受信するかで発生します。このリクエストを送信したパケットを CONNECTION_REQ_PDU と呼びます。CONNECTION_REQ_PDU についての詳細は 3-6-5 項に譲るとして、本項ではこの Connection について概説します。

LL 層でも説明したように、Connection を行ったデバイスのペアは互いに Connection State に移行しますが、そこで与えられる役割は異なります。Initiator から Connection State に遷移したデバイス、すなわち CONNECTION_REQ_PDU を送信した側を Master、CONNECTION_REQ_PDU を受信した側、つまり Advertiser から Connection State に遷移したデバイスを Slave と呼び、区別します。以降の説明での Master、Slave もこれに従う用語だと理解してください。

Connection イベントでの通信タイミングに関するパラメータ

Advertising/Scanning でもそれぞれの状態にイベントが定義されていたように、Connection においてもイベントが定義されています。この Connection イベントでは、Advertising における Advertising Channel PDU/Advertising Packet とは異なり、Data Channel PDU を利用します。Connection イベントが保持されている間は Master および Slave の両者から、Data Channel PDU を利用してパケットのやり取りをすることができます。ただし Connection に起因するイベントでは、少なくとも一度は Master からの送信が行われます。つまり、Slave からのパケッ

ト送信のみで完結する Connection での通信は存在しません。

Connection イベント上での通信タイミングは `connInterval` と `connSlaveLatency` の2つのパラメータで決定します。ここで `connInterval` は Connection イベントのインターバル（間隔）を決定するパラメータで、1.25ms の整数倍の値で定義され、7.5ms ～ 4.0s の範囲をとります。`connInterval` の設定については 3-3-4 項の `CONNECTION_REQ_PDU` で行います。一方、`connSlaveLatency` は Slave 側のデバイスの通信頻度を表す整数値で、slave のデバイスは `connSlaveLatency` の回数分だけ、Master からの Connection イベントを放棄します。これにより接続が維持できなかった場合におけるリスクを軽減しています。`connInterval` と `connSlaveLatency` の関係を図 3-28 に示します。

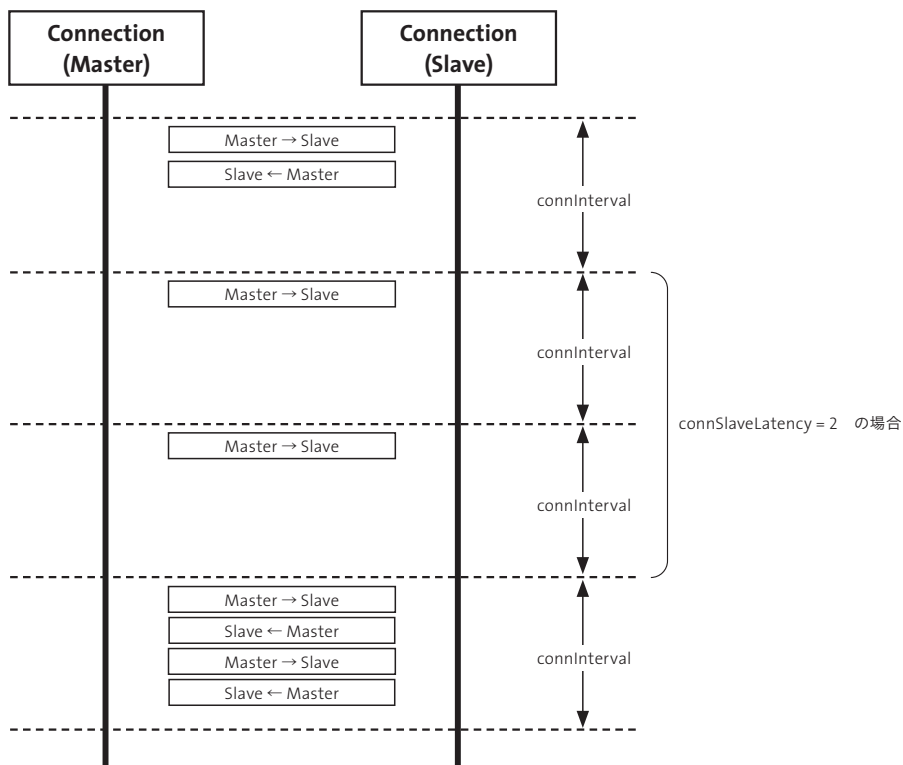


図 3-28 Data Channel PDU の概要

Connection イベントでの接続確認

connSlaveLatencyで、Slaveがイベントに対して応答する回数を定義していましたが、そもそも Connection イベントが途中で瞬断することも十分に考えられます。これに対応するために接続を監視します。BLEではこれをSupervision Timeoutと呼び、所定の時間 TLLconnSupervision の間に接続が確認できなければ、接続が切断されたと認識します。

$TLLconnSupervision = connSupervisionTimeout$ で定義され、Data Channel PDU を 2 度受信するのに要した最大時間として計算されます。connSupervisionTimeout は 10.0ms の整数倍となっており、100.0ms から 32.0s までの値をとり、 $(1+connSlaveLatency) \times connInterval \times 2$ 以上の値をとります。

Connection イベントの終了

Connection を終了するか否かを決定するのが Data Channel PDU 中のヘッダ中の MD bit です (3-6-6 項を参照)。MD bit が Master と Slave 双方で 1 にセットされている場合、Master と Slave は接続を継続します。逆に、Master/Slave の両方で MD=0 の場合であるとする接続を解除し、イベントを終了します。

		Master	
		MD=0	MD=1
Slave	MD=0	Master はパケット送信を行わず、Connection イベントを終了。Slave もパケット送信後の応答を求めない	Master は Connection イベントを継続。Slave はパケット送信後、Master に応答を求めなければならない
	MD=1	Master は Connection イベントを継続。Slave はパケット送信後、Master に応答を求めなければならない	Master は Connection イベントを継続。Slave はパケット送信後、Master に応答を求めなければならない

表 3-6 Data Channel PDU の MD bit の値

3-5. L2CAP（Logical Link Control and Adaption Protocol）によるパケットの制御

物理的な通信を制御する層として、3-4節ではLL層とPHY層をとりあげてきました。本項では、LL層やPHY層の通信によって得られたパケットの流れを制御し、ApplicationとController間のデータを分解・再構築する役割を担っているL2CAP層（Logical Link Control and Adaption Protocol）について解説します。

3-5-1. L2CAP層 （Logical Link Control and Adaption Protocol）

Controller層とHost層では、さまざまな形式のパケットをやり取りすることでBLEの機能を実現させています。このパケットを受け取り、分解・再構築する役割、すなわち一種のマルチプレクサのような役割を担っているのがL2CAP層です。L2CAP層が制御しているController～Application間のパケットの流れを図3-29に示します。図3-29中の最下部にあるBasic Packet Formatが、BLEデバイス間でやり取りされるパケットになります。Basic Packet Formatの長さは47octetで、Preamble、Access Address大きく6つの領域で構成されています（図3-3- Basic Packet Format）。この各領域が段階的にPHY層、LL層、L2CAP層と、それぞれの層でパケットの内容を解釈していき、最終的にPDUのペイロードに含まれるInformation Payloadの23 octetが、Applicationで実際にやり取りを行う値となります。これを図で表したものが図3-30となります。

具体的には、図3-30に示すとおり、受信したBasic Packet Formatは、まずPHY層でPreambleが取り除かれAccess Addressが解釈されます。この時点に残る42 octet分のデータが上位のLL層へ送信します。続くLL層では、この42octet文のデータをPDU Header、Header、CRCから判断した上でPayloadが取り出され、上位のL2CAPへ送信します。

最後のL2CAP層ではLL層から送られたPayloadから、パケットの内容をL2CAP HeaderとMICから判断し、Applicationが利用するInformation Payload (23octet) を取り出します。こ

のわずか23octetのデータを通信内容やプロトコルに沿って再構築するのがL2CAPの大きな役割の一つで、これが本項の冒頭で述べたマルチプレクサ的な機能の所以です。

なおBasic Packet Formatについては3-6-1節、LL層で利用するPayloadについては、「3-6-2. Advertising Channel PDU」、またL2CAP層のInformation Payloadについては3-6-8項で解説します。

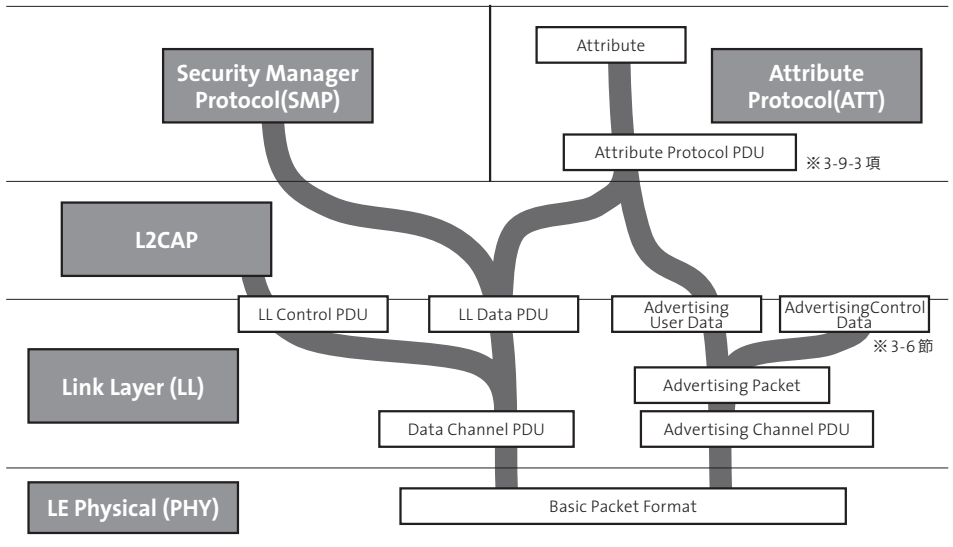


図 3-29 L2CAP層が制御する Controller～Application 間のパケットの流れ

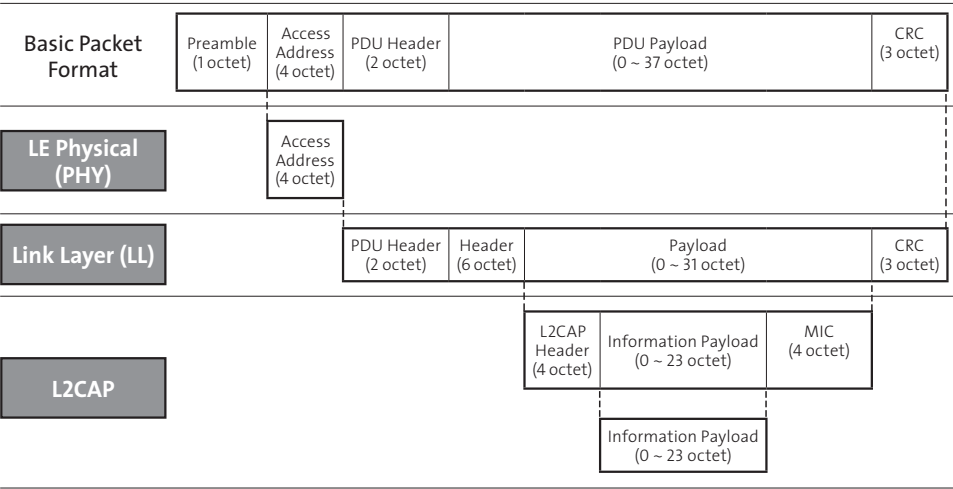


図 3-30 BLE のパケットフォーマット

L2CAPがマルチプレクサ的に動作するにあたって定めているのが、データの通信路となるChannelで、このChannelの識別に利用するIDをChannel Identifier (CID) と呼びます。L2CAPでは、LL層から上がってきたデータが「どのようなデータなのか」をこのCIDにもとづいて分類し、各ChannelでのデータをChannelごとに再構築し、Applicationへと流します。これは仕入れた食材を市場に陳列する前に、一度、仕分け、物品管理を行うことだと理解すればわかりやすいと思います。

BLEにおいてL2CAPでサポートされているCIDの内訳を表3-8に示します。CID = 0x0004でATT (Attribute Protocol)、CID = 0x0006でSMP (Security Manager Protocol) などがあるように、Host層の各プロトコルへの伝送用のChannelが定義されています。また、CID = 0x0005のLow Energy L2CAP signaling ChannelはL2CAPの制御・管理に関する情報を伝送するChannelとなります。なお、CID0x0040～0x007Fまでの領域で確保されているChannelはLE Credit Based Connectionと呼ばれるBluetooth ver.4.1で新規に搭載された機能のために利用されます。LE Credit Based Connectionでは、大容量のファイルの転送などでの利用を想定した機能^{※4}です。L2CAPで利用されるバケットについては、後述する3-6-8項に概要を記載しています。Assigned Numbersについては、現状でどのような用途なのか言及はされていません。

Channel Identifier(CID)	伝送先	内容/用途
0x0001～0x0003	NULL Identifier	×
0x0001～0x0003	予約領域	×
0x0004	ATT (Attribute Protocol)	ATTへのデータ伝送
0x0005	LE L2CAP Signaling Channel	L2CAPの制御・管理
0x0006	SMP (Security Manager Protocol)	SMPへのデータ伝送
0x0007～0x001F	予約領域	×
0x0020～0x003E	Assigned Numbers	×
0x003F	予約領域	×
0x0040～0x007F	動的割当て領域	LE Credit Based Connection
0x0080～0xFFFF	予約領域	×

表3-7 BLEにおいてL2CAPでサポートされているCIDの内訳

※4 この機能は、BLEにおいて実時間性を必要とはしないが容量が大きいファイルを転送したいという用途に利用されるものだと考えます。今後の展開が非常に興味深い機能の1つです。

3-6. BLEのパケットフォーマット

3-4節では、BLEデバイスがどのように接続・通信するのかを概説し、その中で Advertising/Scanning が通信の基本となっていると述べました。この Advertising/Scanning でやり取りされる情報が Advertising Packet と述べましたが、そもそも BLE でパケット構造はどうなっているのでしょうか。

本節では BLE におけるパケットの基本構造について解説し、3-3節で触れた Advertising Packet とそのやり取りについても解説します。

3-6-1. Basic Packet Format

BLE の Basic Packet Format を図3-31 に示します。BLE では、このフォーマットをベースに Advertising に利用する Advertising Channel PDU と、その他のデータ通信全般で利用する Data Channel PDU の2種類が定義されています。

ここで PDU は Protocol Data Unit の略で、通信に送信されるデータ列を指します。PDU は通信内容に関わる情報を含んだヘッダ (Header) と通信内容そのものであるペイロード (Payload) で構成されています。これが基本のパケットフォーマットとなります。LL 層を経由してデバイス間で通信を行い、LL 層を通過後にパケットを分解し、PDU から通信内容を各プロトコル、レイヤで解釈することで BLE による通信がデバイス間で成立します。

Basic Packet Format は合計で4つのデータ領域 **Preamble**、**Access Address**、**PDU**、**CRC** で構成されています。Preamble は1octet で Access Address は4octet となります。PDU の範囲は2～39octets、CRC は3octet です。Preamble を除くと、パケットのサイズは最小で10octet (80bit)、最大でも46octet (368bit) で、クラシック BT と比較すると非常にコンパクトに設計されています。

それでは Basic Packet Format の内容を順にひも解いていきましょう。

Basic Packet Format (47 octet)				
Preamble (1 octet)	Access Address (4 octet)	PDU Header (2 octet)	PDU Payload (0~37 octet)	CRC (3 octet)

図 3-31 Basic Packet Format

Preamble

Basic Packet Formatの先頭には必ず1octetのPreambleが配置されます。Preambleはメッセージの序文や前置きという意味であり、送信側と受信側の通信周波数の同調、通信タイミングの推定、そしてAGC (Auto Gain Control)の調整に利用します。無線でやり取りする場合、受信したデータのどこが開始地点かはそのままではわかりません。そこで、決まった一定のフォーマットの序文、すなわちPreambleを送信することで、通信に関わる調整を行う仕組みになっています。

Advertisingに利用するAdvertising Channel PDUの場合、10101010₍₂₎を利用するように定義されています。他方、Data Channel Packetは、その後送信されるAccess AddressのLSBに依存して10101010₍₂₎もしくは01010101₍₂₎のいずれかの一方を利用するように定義されています。たとえば、もしAccess AddressのLSBが1の場合、Preambleは01010101₍₂₎、他方、0の場合は10101010₍₂₎となります。

Access Address

Access Addressはデータの接続に関する関係性を定義するパケットです。Advertising Channel PDUについては、10001110100010011011111011010110⁽²⁾ (0x8E89BED6)の固定値を常に利用します。他方、Data channel PDUでは、定義された規定を満たす32bitの整数値を利用します。

PDU (Protocol Data Unit)

Preamble、Access Addressに続くのがPDU (Protocol Data Unit)となります。Advertisingの場合はAdvertising Channel PDUが、その他データ通信の場合はData Channel PDUがそれぞれPDUとして出力されます。

3-6-2. Advertising Channel PDU

Advertising Channel PDUは2octet（16bit）のHeaderとデータの実態となる可変長のペイロード Advertising PDU(最大37octet)からなります。Headerはさらに6つのデータ領域、PDU_TYPE（4bit）、RFU（2bit）、TxAdd（1bit）、RxAdd（1bit）、Length（6bit）、RFU（2bit）で構成されています。Advertising Channel PDUの構成を図3-32に示します。

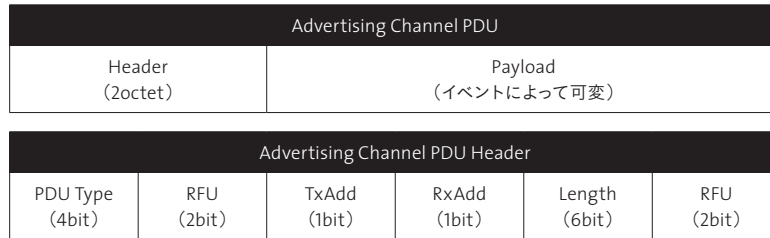


図 3-32 Advertising Channel PDU

PDU_TYPE

Advertising Channel PDUのPDU_TYPEの値と対応するパケット名、PDU名、その内容を表3-8に示します。

PDU_TYPE	パケット名	PDU	PDUの利用シーン
0000 ₍₂₎	ADV_IND	Advertising PDU	接続型・無指向性のAdvertisingイベントで利用
0001 ₍₂₎	ADV_DIRECT_IND		接続型・有指向性のAdvertisingイベントで利用
0010 ₍₂₎	ADV_NONCONN_IND		非接続型・無指向性のAdvertisingイベントで利用
0011 ₍₂₎	SCAN_REQ	Scanning PDU	LL層 (Scanning State) → LL層 (Advertising State) のScanningで利用
0100 ₍₂₎	SCAN_RSP		LL層 (Advertising State) → LL層 (Scanning State) のScanningで利用
0101 ₍₂₎	CONNECT_REQ	Initiating PDU	LL層 (Initiating State) → LL層 (Advertising State) のScanningで利用
0110 ₍₂₎	ADV_SCAN_IND	Advertising PDU	スキャン可能な無指向性のAdvertisingイベントで利用
0111 ₍₂₎ ～1111 ₍₂₎	Reserved	×	×

表 3-8 Advertising Channel PDUのPDU_TYPE

PDU_TYPEはAdvertising Channel PDUのPDUとしての種別を宣言するビットフィールドです。各種別に対して、ADV_IND、ADV_DIRECT_IND、ADV_NONCONN_IND、ADV_SCAN_INDの4つの種別のPDUをまとめてAdvertising PDUと呼びます。また、SCAN_REQ、SCAN_RSPの2つの種別のPDUをまとめてScanning PDU、CONNECT_REQをInitiating PDUと呼び、PDUを区別します。以降で、各PDUについて解説していきます。

3-6-3. Advertising PDU

ADV_IND、ADV_DIRECT_IND、ADV_NONCONN_IND、ADV_SCAN_INDの4つの種別のAdvertising Channel PDUをまとめてAdvertising PDUと呼びます。表3-9に示すとおり、Advertising PDUのいずれのPDUもAdvertising イベントで利用され、Advertising Stateのデバイスから周辺のScanning State、もしくはInitiating Stateのデバイスに対して送信されます。

各PDU_TYPEの違いは、Advertising イベントにおいてAdvertising PDUのどのようなレスポンスを求めるかという点にあります。このときレスポンスとして想定されているのは、SCAN_REQ、CONNECT_REQの2種類です。つまり、Advertising PDUを出力するAdvertising Stateであるデバイスが、周辺のデバイスに対してどのようなネットワークトポロジを許可するかという情報が、このPDU_TYPEには含まれています。各イベントにおいて、レスポンス(SCAN_REQ、CONNECT_REQ)の有無がどのように対応しているのかを表3-9に示します。

PDU TYPE	周辺の機器からのレスポンス	
	SCAN_REQ	CONNECT_REQ
ADV_IND	○	○
ADV_DIRECT_IND	×	○
ADV_NONCONN_IND	×	×
ADV_SCAN_IND	○	×

表3-9 Advertising PDU が対応するレスポンス

ADV_INDの場合は、Advertising PDUのレスポンスとしてSCAN_REQ、CONNECT_REQを受け付けます。この場合は、周辺のデバイスに対して自身の発見を許可するとともに、接続についても受け入れることを意味しています。ADV_DIRECT_INDの場合も接続を受け入れませんが、スキャンについて受け入れません。したがって、ADV_DIRECT_INDを発信するデバイスは、事前に接続相手が自明な相手からの接続要求のみを受け入れます。

他方、ADV_NONCONN_INDはスキャンや接続のいずれの要求に対しても受け入れませ

ん。したがって、特定の相手に対してのみAdvertisingをする状態を意味します。残るADV_SCAN_INDは、スキャン要求のみを受け入れます。したがって、ブロードキャスト型のネットワークでのみ動作するデバイスということになります。

それでは、次に各PDU_TYPEでのPDUの詳細について順に見ていきましょう。

ADV_IND

ADV_IND PDUのPDU構成を図3-33に示します。ADV_IND PDUはAdvA (6octets) と AdvData (0～31octets) の2つの領域で構成されます。AdvA領域にはTxAddの値に応じて (TxAdd=0の場合はpublic address、TxAdd=1の場合はrandom address) AdvertiserのPublic AddressもしくはRandom Addressが入力されます。また、AdvData領域にはAdvertising Packetが入ります。

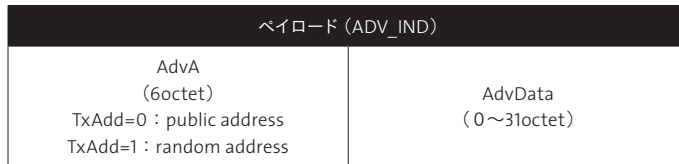


図3-33 ADV_INDのペイロード

ADV_DIRECT_IND

ADV_DIRECT_INDの持つペイロードの構成は図3-34のとおりです。このPDUは、AdvA (6octet) と InitA (6octet) の2つの領域で構成されます。AdvA領域はTxAddの値に応じて、TxAdd=0の場合はAdvertiserのpublic address、TxAdd=1の場合はAdvertiserのrandom addressが入力されます。InitA領域にはAdvertiserの相手先、すなわちInitiatorが入力されます。InitA領域の値はRxaddの値に応じて、RxAdd=0の場合はInitiatorのpublic addressが、RxAdd=1の場合はInitiatorのrandom addressが入力されます。

ペイロード (ADV_DIRECT_IND)	
AdvA (6octet) TxAdd=0 : public address TxAdd=1 : random address	InitA (6octet) RxAdd=0 : public address RxAdd=1 : random address

図 3-34 ADV_DIRECT_IND のペイロード

ADV_NONCONN_IND

ADV_NONCONN_IND PDUの持つペイロードの構成は図3-35のとおりです。このPDUは、AdvA (6octet)、AdvData (0～31octet) の2つの領域で構成されます。AdvAとAdvDataの値はADV_INDの場合と同様、AdvA領域はTxAddの値に応じて (TxAdd=0の場合はpublic address、TxAdd=1の場合はrandom address)、AdvertiserのPublic Addressもしくはrandom addressが入力されます。また、AdvData領域にはAdvertising Packetが入ります。

ペイロード (ADV_NONCONN_IND)	
AdvA (6octet) TxAdd=0 : public address TxAdd=1 : random address	AdvData (0～31octet)

図 3-35 ADV_NONCONN_IND のペイロード

ADV_SCAN_IND

ADV_SCAN_IND PDUの持つペイロードの構成は図3-36のとおりです。このPDUは、AdvA (6octet)、AdvData (0～31octet) の2つの領域で構成されます。AdvAとAdvDataの値はADV_INDの場合と同様、AdvA領域はTxAddの値に応じて (TxAdd=0の場合はpublic address、TxAdd=1の場合はrandom address)、Advertiserのpublic Addressもしくはrandom Addressが入力されます。また、AdvData領域にはAdvertising Packetが入力されます。

ペイロード (ADV_SCAN_IND)	
AdvA (6octet) TxAdd=0 : public address TxAdd=1 : random address	AdvData (0~31octet)

図 3-36 ADV_SCAN_INDのペイロード

3-6-4. Scanning PDU

SCAN_REQ、SCAN_RSPの2つの種別の Advertising Chanel PDUをまとめて Scanning PDUと呼びます。いずれのPDUも Scanning イベントで利用されます。Scanning Stateのデバイスから周辺の Advertising Stateのデバイスに対して送信されます。

SCAN_REQ

SCAN_REQ PDUの持つペイロードの構成は図3-37のとおりです。このPDUはScanA(6octet)、AdvA(6octet)の2つの領域で構成されます。ScanAはTxAddの値に応じてTxAdd=0の場合はScannerのpublic addressが、TxAdd=1の場合はScannerのrandom addressが入力されます。AdvA領域には、このPDUの送信先のデバイス(Advertiser)のアドレスが入力されます。Rxaddの値に応じてRxAdd=0の場合はpublic addressが、RxAdd=1の場合はrandom addressが入力されます。

ペイロード (SCAN_REQ)	
ScanA (6octet) TxAdd=0 : public address TxAdd=1 : random address	AdvA (6octet) RxAdd=0 : public address RxAdd=1 : random address

図 3-37 SCAN_REQのペイロード

SCAN_RSP

SCAN_RSP PDUの持つペイロードの構成は図3-38のとおりです。このPDUはScanA (6octet)、ScanRspData (0～31octet) の2つの領域で構成されます。ScanAはTxAddの値に応じてTxAdd=0の場合はScannerのpublic addressが、TxAdd=1の場合はScannerのrandom addressが入力されます。ScanRspData領域にはAdvertising Packetが入力されます。

ペイロード (SCAN_RSP)	
ScanA (6octet) TxAdd=0 : public address TxAdd=1 : random address	ScanRspData (0~31octet)

図 3-38 SCAN_RSP のペイロード

3-6-5. Initiating PDU

CONNECT_REQであるAdvertising Channel PDUをInitiating PDUと呼びます。Initiating PDUはInitiating Stateであるデバイスから、Advertising Stateである他の周辺のデバイスに送信されます。

CONNECT_REQ

CONNECT_REQの持つペイロードの構成は図3-39のとおりです。このPDUはInitA (6octet)、AdvA (6octet)、LLData (22octet) の3つの領域で構成されています。InitA領域にはTxAddの値に応じてTxAdd=0の場合はInitiatorのpublic addressが、TxAdd=1の場合はInitiatorのrandom addressが入力されます。AdvA領域には、RxAddの値に応じて、RxAdd=0の場合はAdvertiserのpublic addressが、RxAdd=1の場合はAdvertiserのrandom addressが入力されます。

ペイロード (CONNECT_REQ)									
ScanA (6octet) TxAdd=0 : public address TxAdd=1 : random address			AdvA (6octet) RxAdd=0 : public address RxAdd=1 : random address			LLData (22octet)			
LLData									
AA (4octet)	CRCInit (3octet)	WinSize (1octet)	WinOffset (2octet)	Interval (2octet)	Latency (2octet)	Timout (2octet)	ChM (5octet)	Hop (5bit)	SCA (3bit)

図 3-39 CONNECT_REQ のペイロードと LLData の内容

LLData は図 3-39 のとおり、さらに 10 個のフィールドで構成されています。

AA (Access Address)

AA 領域は 3-6-1 項で示した規則に基づいた LL 層の Access Address が記述されています。

CRCInit (CRC Initial value)

CRCInit 領域には LinkLayer の接続に利用される CRC の計算における初期値が入力されています。

WinSize

WinSize 領域は transmitWindowSize の値が入力されています。この transmitWindowSize の値は、 $\text{transmitWindowSize} = \text{WinSize} \times 1.25\text{ms}$ で定義されます。

WinOffset

WinOffset 領域は transmitWindowOffset の値が入力されています。この transmitWindowOffset の値は、 $\text{transmitWindowOffset} = \text{WinOffset} \times 1.25\text{ms}$ で定義されます。

Interval

Interval 領域には conInterval 値が入力されています。この conInterval の値は $\text{conInterval} = \text{Interval} \times 1.25\text{ms}$ で定義されます。

Latency

Latency 領域には connSlaveLatency の値が入力されます。この connSlaveLatency の値は $\text{connSlaveLatency} = \text{Latency}$ で定義されます。

Timeout

Timeout領域にはconnSupervisionTimeoutの値が入力されます。connSuperViusionTimeoutの値は $\text{connSuperViusionTimeout} = \text{Timeout} \times 10\text{ms}$ で定義されます。

ChM

ChM領域はデータチャンネルの使用/未使用を示したチャンネルマップを示しています。5octet（40bit分）の領域のうちLSBからbit0から順にチャンネル0～39までを表しています。つまり、36bit目の値はチャンネル36の使用/未使用を表します。また、各ビットの値は1の場合は使用中、0の場合は未使用を表します。ChM領域の5octet中37、38、そして39bitについては使用が禁止されています（Advertising Channelに相当するため）。

Hop

Hop領域は、hpIncrementの値が入力されます。BLEにおける無線通信中のチャンネル選択に関するアルゴリズムは3-4-1項で簡単に紹介しています。hpIncrementの値は5～16までのランダムな値となります。

SCA

最後のSCA領域は、最悪の状態を想定したMaster側のsleep clockについて定義したmasterSCAの値が入力されます。各SCAの値に対するクロックの精度範囲については、次の表3-10のとおりです。

SCA	masterSCA
0	251ppm ～ 500ppm
1	151ppm ～ 250ppm
2	101ppm ～ 150ppm
3	76ppm ～ 100ppm
4	51ppm ～ 75ppm
5	31ppm ～ 50ppm
6	21ppm ～ 30ppm
7	0ppm ～ 20ppm

表 3-10 SCAの値に対するクロックの精度範囲

3-6-6. Data Channel PDU

Data Channel PDUはAdvertising以外のデータ通信に利用するパケットです。このPDUは16bitのヘッダと、可変長のペイロード、そしてMIC（Message Integrity Check）と呼ばれる暗号化に関するデータによって構成されます。Data Channel PDUのパケット構成を図3-40に示します。

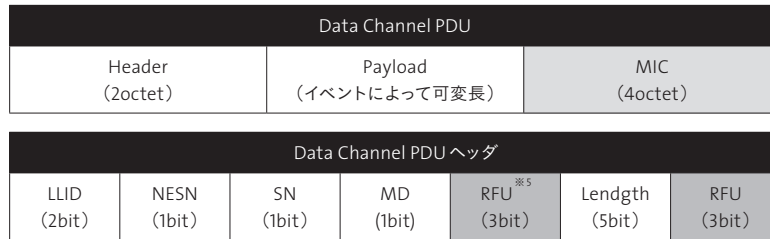


図 3-40 Data Channel PDU の構成

Data Channel PDU のヘッダ

Data Channel PDU のヘッダは、さらに6つの領域、LLID、NESN、SN、MD、Length、RFUで構成されています。ヘッダの構成を表3-11に示します。RFUは予約領域で将来への拡張を考慮して事前に確保されているビット領域で、現状は利用されていません。まずはこのヘッダからData channel PDUを読み解いていきましょう。

LLID

Data Channel PDUのペイロードのフォーマットはLLIDの値に依存します。つまり、LLIDはAdvertising PDUのPDU_TYPEと同様にペイロードの種別を定めるビット領域といえます。表3-10にLLIDに対応する種別を示します。LLIDが01₍₂₎、10₍₂₎の場合、Data Channel PDUのペイロードはLL Data PDUが、11₍₂₎の場合はData Channel PDUのペイロードはLL Control PDUがそれぞれ入力されます。これらのPDUの詳細については後述します。

※5 Reserved of Future Use (将来的な拡張のための予約領域)

LLID	パケットの種別
00(2)	予約領域。未使用
01(2)	LL Data PDU
10(2)	
11(2)	LL Control PDU

表 3-11 LLID に対応する種別

NESN (Next Expected Sequence Number)

NESNはパケットの確認応答に用いられる 1bit のパラメータで SN と組合わせて利用します。詳細については、3-7-3 項で解説します。

SN (Sequence Number)

SNはパケットの確認応答に用いられる 1bit のパラメータで NESN と組合わせて利用します。詳細については、3-7-3 項で解説します。

MD (More Data)

MDは Connection イベントの継続、終了を決定するための 1bit のパラメータです。詳細は 3-4-6 項で解説します。

Length

Lengthはペイロードのサイズを表します。単位は octet で、MIC を生成する場合は MIC とペイロードを含めた長さが計算されます。Length が 5bit 長であることから、その値は 0 ～ 31 の範囲となります。MIC が 32bit (= 4octet) であることから、Data Channel PDU のペイロードは最大で 27octet (= 31octet – 4octet) の長さをとることがわかります。

LL Data PDU

LLID の値が 01₍₂₎、10₍₂₎ である Data Channel PDU のペイロードをまとめて LL Data PDU と呼びます。また、LLID が 01₍₂₎ であり、かつヘッダの Length が 00000₍₂₎ である場合を特に Empty PDU と呼びます。LL 層が Master として動作するデバイスはこの Empty PDU を、Empty PDU を含む任意の Data Channel PDU に対する応答をスレーブに対して許可するために利用します。なお、LLID の値が 11₍₂₎ である LL Data PDU は、Length が 0 になることはありません。

LL Control PDU

LLIDが11₍₂₎であるData Channel PDUのペイロードをLL Control PDUと呼び、デバイス間のLL層の接続の操作を行います。LL Control PDUの構成は図3-41のとおりで、Opcode (1octet) と CtrData (0～26octet) の2つの領域で構成されています。

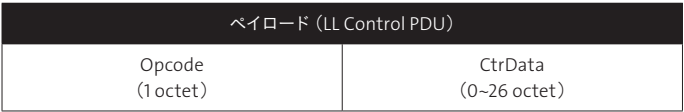


図 3-41 LL Control PDU の構成

LL Control PDUは前述したLL Data PDUとは異なり、Length領域が00000₍₂₎、つまりペイロード長が0になることはありません。Opcodeで指定される CtrlData の種別毎に固定長のペイロードが定義されています。

Opcode

Opcodeは前述したLLIDと同様、後に続くペイロードの種別を規定します。さっそく、Opcodeで定義されるLL Control PDUについて見ていきましょう。表3-12にOpcodeの値に対応するLL Control PDUの種別を示します。

1

2

3

4

5

6

7

8

9

10

11

12

Opcode	Control PDU 名
0x00	LL_CONNECTION_UPDATE_REQ
0x01	LL_CHANNEL_MAP_REQ
0x02	LL_TERMINATE_IND
0x03	LL_ENC_REQ
0x04	LL_ENC_RSP
0x05	LL_START_ENC_REQ
0x06	LL_START_ENC_RSP
0x07	LL_UNKNOWN_RSP
0x08	LL_FEATURE_REQ
0x09	LL_FEATURE_RSP
0x0A	LL_PAUSE_ENC_REQ
0x0B	LL_PAUSE_ENC_RSP
0x0C	LL_VERSION_IND
0x0D	LL_REJECT_IND
0x0E	LL_SLAVE_FEATURE_REQ
0x0F	LL_CONNECTION_PARAM_REQ
0x10	LL_CONNECTION_PARAM_RSP
0x11	LL_REJECT_IND_EXT
0x12	LL_PING_REQ
0x13	LL_PING_RSP
0x14～0xFF	RFU(Reserved for Future Use)

表 3-12 Opcode の値に対応する LL Control PDU の種別

LL_CONNECTION_UPDATE PDU

LL_CONNECTION_UPDATE PDU (Opcode = 0x00) での CtrData のフォーマットは図 3-42 のとおりです。図からわかるように、CtrData は WinSize (1octet)、WinOffset (2octet)、Interval (2octet)、Latency (2octet)、Timeout (2octet)、Instant (2octet) の 6 つの領域で構成されています。

CtrData (LL_CONNECTION_UPDATE)					
WinSize (1octet)	WinOffset (2octet)	Interval (2octet)	Latency (2octet)	Timeout (2octet)	Instant (2octet)

図 3-42 LL_CONNECTION_UPDATE PDU での CtrData のフォーマット

WinSize

WinSize 領域は transmitWindowSize の値が入力されています。この transmitWindowSize の値は、 $\text{transmitWindowSize} = \text{WinSize} \times 1.25\text{ms}$ で定義されます。

WinOffset

WinOffset 領域は transmitWindowOffset の値が入力されています。この transmitWindowOffset の値は、 $\text{transmitWindowOffset} = \text{WinOffset} \times 1.25\text{ms}$ で定義されます。

Interval

Interval 領域には conInterval 値が入力されています。この conInterval の値は $\text{conInterval} = \text{Interval} \times 1.25\text{ms}$ で定義されます。

Latency

Latency 領域には connSlaveLatency の値が入力されます。この connSlaveLatency の値は $\text{connSlaveLatency} = \text{Latency}$ で定義されます。

Timeout

Timeout 領域には connSupervisionTimeout の値が入力されます。connSuperViusionTimeout の値は $\text{connSuperViusionTimeout} = \text{Timeout} \times 10\text{ms}$ で定義されます。

Instant

Instant 領域には connInstant value の値が入力されます。connInstant vlue の値は 0 ～ 65535 の範囲で定義されています。

LL_CHANNEL_MAP PDU

LL_CHANNEL_MAP PDU (Opcode = 0x01) での CtrData のフォーマットは図 3-43 のとおりです。

CtrData (LL_CHANNEL_MAP)	
ChM (5octet)	Instant (2octet)

図 3-43 LL_CHANNEL_MAP PDU での CtrData のフォーマット

ChM

ChM領域はデータチャンネルの使用/未使用を示したチャンネルマップを示しています。5octet (40bit 分) の領域のうち、LSBから bit0 から順にチャンネル0～39までを表しています。つまり、36bit目の値はチャンネル36の使用/未使用を表します。また、各ビットの値は1の場合は使用中、0の場合は未使用を表します。

ChM領域の5octet中37、38、そして39bitについては利用が禁止されています(Advertising Channelであるため)。この項目はAdvertising Channel PDUのInitiating PDU (CONNECT_REQ) 中のChMと同様の定義です。

Instant

Instant領域にはconnInstant valueの値が入力されます。connInstant valueの値は0～65535の範囲で定義されています。

LL_TERMINATE_IND PDU

LL_TERMINATE_IND PDU (Opcode = 0x02) でのCtrDataのフォーマットは図3-44のとおりです。

CtrData (LL_TERMINATE_IND)	
Error Code (1octet)	

図 3-44 LL_TERMINATE_IND PDU での CtrData のフォーマット

ErrorCode

ErrorCode領域は「なぜ接続が終了したのか」という情報を接続していたデバイスに対して送信するために利用します。ErrorCodeの詳細は Bluetooth Core Specification vol.2 PartDに記載されています。

LL_ENC_REQ PDU

LL_ENC_REQ PDU (Opcode = 0x03) での CtrData のフォーマットは次の図 3-45 のとおりです。この PDU は暗号化に関する処理を司る PDU であり、本書では解説を割愛します。詳細は Bluetooth Core Specification v4.1 vol.3 Part-C section 10 を参照してください。

CtrData (LL_ENC_REQ)			
Rand (8octet)	EDIV (2octet)	SKDm (8octet)	IVm (4octet)

図 3-45 LL_ENC_REQ PDU での CtrData のフォーマット

Rand

Rand 領域は暗号化のために Host から提供される乱数が代入されます。また、暗号化の際に EDIV とともに利用します。

EDIV

EDIV 領域は Rand とともに暗号化の際に利用する EDIV (Encrypted Diversifier) が代入されます。

SKDm

SKDm 領域は Master の SKD (Session Key Diversifier) の一部が代入されます。

IVm

IVm 領域は Master の IV (Initialization Vector) の一部が代入されます。

LL_ENC_RSP PDU

LL_ENC_RSP PDU (Opcode = 0x04) での CtrData のフォーマットは図 3-46 のとおりです。この PDU は暗号化に関する処理を司る PDU であり、本書では解説を割愛します。詳細は Bluetooth Core Specification v4.1 vol.3 Part-C section 10 を参照してください。

CtrData (LL_ENC_RSP)	
SKDs (8octet)	IVs (4octet)

図 3-46 LL_ENC_RSP PDUでのCtrDataのフォーマット

SKDs

SKDs領域はSlaveのSKD (Session Key Diversifier) の一部が代入されます。

IVs

IVs領域はSlaveのIV (Initialization Vector) の一部が代入されます。

LL_START_ENC_REQ PDUとLL_START_ENC_RSP PDU

LL_START_ENC_REQ PDU (Opcode = 0x05) および LL_START_ENC_RSP PDU (Opcode = 0x06) では CtrData 領域を持ちません。このPDUは暗号化に関する処理を司るPDUであり、本書では解説を割愛します。詳細は Bluetooth Core Specification v4.1 vol.3 Part-C section 10 を参照してください。

LL_UNKNOWN_RSP PDU

通信先のデバイスで対応していない (実装されていない) LL Control PDUを送信した場合、受信デバイスのLL層はレスポンスとして LL_UNKNOWN_RSP PDU (Opcode = 0x07) を返信し、CtrDataにそのエラー内容を入力します。

また、LL Control PDUとして不適切な値、たとえば予約領域 (Opcode = 0x14～0xFF) などが送信された場合、もしくは CtrData が適切なものではなかった場合、この場合も受信デバイスのLL層は LL_UNKNOWN_RSP PDUを返信します。

CtrData (LL_UNKNOWN_RSP PDU)
Unknown type (1octet)

図 3-47 LL_UNKNOWN_RSP PDUでのCtrDataのフォーマット

Unknown Type

Unknown Type領域は、受信したLL Control PDUのOpcodeが未対応な場合、そのOpcodeの値が代入されます。

LL_FEATURE_REQ PDU

LL_FEATURE_REQ PDU (Opcode = 0x08) でのCtrDataのフォーマットは図3-48のとおりです。

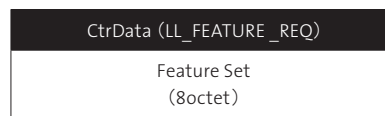


図 3-48 LL_FEATURE_REQ PDUでのCtrDataのフォーマット

Feature Set

Feature Set領域は、Master側のLL層が対応している機能についての情報が代入されます。

LL_FEATURE_RSP PDU

LL_FEATURE_RSP PDU (Opcode = 0x09) でのCtrDataのフォーマットは図3-49のとおりです。

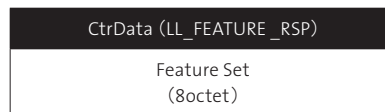


図 3-49 LL_FEATURE_RSP PDUでのCtrDataのフォーマット

Feature Set

Feature Set領域は、MasterもしくはSlave側のLL層が対応している機能についての情報が代入されます。

LL_PAUSE_ENC_REQ PDUとLL_PAUSE_ENC_RSP PDU

LL_PAUSE_ENC_REQ PDU (Opcode = 0x0A) および LL_PAUSE_ENC_RSP PDU (Opcode = 0x0B) では CtrData 領域を持ちません。

LL_VERSION_IND PDU

LL_VERSION_IND PDU (Opcode = 0x0C) での CtrData のフォーマットは図 3-50 のとおりです。

CtrData (LL_VERSION_IND)		
VersNr (1octet)	Compld (2octet)	SubVersNr (2octet)

図 3-50 LL_VERSION_IND PDU での CtrData のフォーマット

VersNr

VersNr 領域には、デバイスが対応する Bluetooth Core Specification のバージョンナンバーが代入されます。

Compld

Compld 領域には、製品の Bluetooth Controller の製造会社の ID の値が代入されます。

SubVersNr

SubVersNr 領域には、Bluetooth Controller のリビジョンもしくは固有 ID が代入されます。

LL_REJECT_IND PDU

LL_REJECT_IND PDU (Opcode = 0x0D) での CtrData のフォーマットは図 4-51 のとおりです。

CtrData (LL_REJECT_IND)
Error Code (1octet)

図 3-51 LL_REJECT_IND PDU での CtrData のフォーマット

ErrorCode

ErrorCode領域は「なぜ接続が終了したのか」という情報を接続していたデバイスに対して送信するために利用します。ErrorCodeの詳細は、Bluetooth Core Specification vol.2 PartDに記載されています。

LL_SLAVE_FEATURE_REQ PDU

LL_SLAVE_FEATURE_REQ (Opcode = 0x0E) での CtrData のフォーマットは図3-52のとおりです。

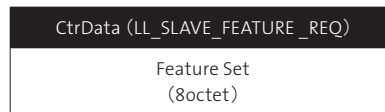


図 3-52 LL_SLAVE_FEATURE_REQ PDUでの CtrData のフォーマット

Feature Set

Feature Set 領域は、Slave 側の LL 層が対応している機能についての情報が代入されます。

LL_CONNECTION_PARAM_REQ PDU と LL_CONNECTION_PARAM_RSP PDU

LL_CONNECTION_PARAM_REQ PDU (Opcode=0x0F) および LL_CONNECTION_PARAM_RSP PDU (Opcode=0x10) での CtrData のフォーマットは図3-53のとおりです。いずれの PDU も CtrData の内容は同じフォーマットとなります。

CtrData (LL_CONNECTION_PARAM_REQ / LL_CONNECTION_PARAM_RSP)											
Interval_Min (2octet)	Interval_Max (2octet)	Latency (2octet)	Timeout (2octet)	PreferredPeriodicity (1octet)	ReferenceConn EventCount (2octet)	Offset0 (2octet)	Offset1 (2octet)	Offset2 (2octet)	Offset3 (2octet)	Offset4 (2octet)	Offset5 (2octet)

図 3-53 LL_CONNECTION_PARAM_REQ PDU と LL_CONNECTION_PARAM_RSP PDU での CtrData のフォーマット

Interval_Min

Interval_Min 領域は connInterval の最小値を示します。Interval_Min の値は、connInterval = Interval_Min × 1.25ms に従います。

Interval_Max

Interval_Max 領域は connInterval の最大値を示します。Interval_Max の値は、connInterval = Interval_Max × 1.25ms に従います。

Latency

Latency 領域は、connSlaveLatency の値を示します。Latency の値は connSlaveLatency = Latency に従います。Latency の値は Connection イベント数を示します。

Timeout

Timeout 領域には、connSupervisionTimeout の値が入力されます。connSuperViusionTimeout の値は connSuperViusionTimeout = Timeout × 10ms で定義されます。

PreferredPeriodicity

PreferredPeriodicity 領域の値に対して、connInterval は倍数が値となる必要があります。PreferredPeriodicity の値は 1.25ms の整数倍となります。たとえば、PreferredPeriodicity=100 の場合、connInterval は 125ms となります。PreferredPeriodicity ≠ 0 であり、その値は必ず Interval_Max 以下となります。

ReferenceConnEventCount

ReferenceConnEventCount 領域は、後述する Offset0 から Offset5 までの値に関連した connEventCounter の値を示します。ReferenceConnEventCount の値の範囲は 0～65535 となります。

Offset0～Offset5

Offset0～Offset5 までの領域は、ReferenceConnEventCount に関連する更新された接続パラメータとともに、BLE の接続におけるアンカーポイントの位置について設定可能な値を示します。いずれも 1.25ms 単位で、各値は Offset0 から優先度の高い順で並んでいます。つまり、Offset0 がもっとも優先度が高く、Offset1、Offset2 と順番に優先度が下がっていきます。Offset0 から Offset5 はいずれも Interval_Max よりも必ず小さな値となり、0xFFFF の値は禁止されています。

LL_REJECT_IND_EXT PDU

LL_REJECT_IND_EXT PDU (Opcode = 0x11) での CtrData のフォーマットは図 3-54 のとおりです。

CtrData (LL_REJECT_IND_EXT PDU)	
RejectOpcode (1octet)	Error Code (1octet)

図 3-54 LL_REJECT_IND_EXT PDU での CtrData のフォーマット

RejectOpcode

RejectOpcode 領域には、デバイスによってリジェクトされた LL Control PDU の Opcode が代入されます。

Error Code

ErrorCode 領域は「なぜ接続が終了したのか」という情報を接続していたデバイスに対して送信するために利用します。ErrorCode の詳細は Bluetooth Core Specification vol.2 PartD に記載されています。

LL_PING_REQ PDU と LL_PING_RSP PDU

LL_PING_REQ PDU (Opcode = 0x12) および LL_PING_RSP PDU (Opcode = 0x13) では CtrData 領域を持ちません。PING 機能については Bluetooth 4.1 にて実装された新しい機能で、本 PDU はそれに利用されます。本書では解説を割愛します。詳細は、Bluetooth Core Specifications v4.1 vol.6 Part-D section 6.13 を参照してください。

MIC (Message Integrity Check)

MIC 領域は、デバイスの接続において暗号化を利用する場合に生成されるデータとなります。暗号化を利用しない場合、MIC は Data Channel PDU には含まれません。また、暗号化する場合であってもペイロードの長さが 0 であれば含まれません。MIC 領域の生成については Bluetooth Core Specification v4.1 Vol.5 Part E Section 1 に記述されています。

3-6-7. CRC (Cyclic Redundancy Check)

Basic Packet Format の終端は、必ず 3octet 分の CRC (Cyclic Redundancy Check) となっています。CRC とは送信するデータの誤り検出に利用される値で、Peterson らによって CRC 値の計算方法が発明されました [参考文献]。CRC 値は、任意の長さのデータ列の入力として、その長さに応じた計算式 (生成多項式) から求めます。BLE の基本パケットフォーマットで利用される CRC 値は、直前に送信された PDU の値から計算され、その生成多項式は次のように定義されています。

$$x^{24} + x^{10} + x^9 + x^6 + x^4 + x^3 + x + 1$$

3-6-8. L2CAP 層でのパケットフォーマット

LL 層を通過した段階でのパケットフォーマットを図 3-57 に示します。この PDU は L2CAP PDU と呼ばれ、Connection Oriented Channel で利用する PDU です。本書では解説を割愛しますが、この PDU の他に、L2CAP の制御に利用する Signaling Packet、Bluetooth ver.4.1 で追加された LE Credit Based Connection で利用する PDU も存在します。

L2CAP PDU (27 octet)		
Basic L2CAP Header LSB		MSB
Length (2octet)	Channel ID (2octet)	Information Payload (23octet)

図 3-55 L2CAP PDU

この時点でパケットのサイズは 27octet となっています。このパケットはさらに L2CAP で利用する Length、Channel ID、そして Information Payload の 3 つの領域で構成されています。Length は 2octet で Information Payload のサイズを示し、最大値は 65535octet となります。このサイズは上位の Attribute などを利用するメソッドにおけるパラメータなどをすべて含んだ値であり、単一の Attribute のサイズや MTU のサイズとは関係がないことに注意が必要です。続く Channel ID は 2octet の CID を決定する値で、Information Payload の伝送先を決定します。CID については 3-5-1 節を参照ください。最後の Information Payload が Host 層の Application とのやり取りに利用されるデータでユーザーのデータが含まれる領域になります。この領域のサイズは 23octet (=27octet - 2octet - 2octet) となり、これが ATT_MTU のデフォルトのサイズ ATT_MTU_{default} (=23octet) となります。

3-7. LL層における通信のやり取り

3-6節ではBLEの通信上でどのようなパケットが利用されているか解説しました。本節では、そのパケットを用いてどのような通信のやり取りが行われているかについて解説します。

3-7-1. Bluetooth Device Addressによるフィルタリング

LL層はLink Layer Device Filteringと呼ばれるフィルタリング機能を用いて、周辺とのデバイスからの応答を目的に応じて最小化することができます。このフィルタリングはBluetooth Device Addressを利用して行われ、AdvertisingイベントのうちADV_IND、ADV_DIRECT_INDで利用することができます。

White List

Link Layer Device Filteringを利用してフィルタリングするデバイスのリストをWhite Listと呼びます。White ListはWhite List Recordから構成されており、White List RecordはBluetooth Device AddressとDevice Addressの種類（publicなのかrandomなのか）の両方のデータを含みます。White Listは、リセット直後ではゼロ・クリアされており、起動完了後Host側によって設定されます。

3-7-2. Advertising Channelにおける通信

3-5節で述べたように、AdvertisingではADV_IND、ADV_DIRECT_IND、ADV_NONCONN_IND、ADV_SCAN_INDの4種類のイベントが定義されています。ここではイベントごとに、その通信の流れを確認していきましょう。

1

2

3

4

5

6

7

8

9

10

11

12

ADV_IND (Connctable Undirected Event) での通信

ADV_IND イベントにおける通信の流れの概略を図3-56に示します。このタイプのイベントでは、AdvertiserがScannerもしくはInitiatorいずれかとのスキャン要求に応じることができます。また、接続要求にも応じることができます。図中のT_IFSは同じチャンネルIDで通信するパケット間のインターバルを示し、その間隔は150us（固定）となります。

AdvertiserがADV_INDを送信後、Advertiserの相手がScannerの場合、Scannerは応答としてSCAN_REQ PDUを送信します（図3-57）。一方、相手がInitiatorの場合、InitiatorはCONNECT_REQ PDUを送信し、両デバイスがConnection状態に移行します（図3-58）。

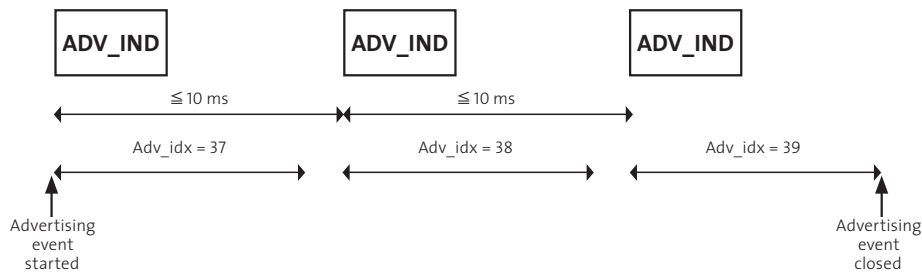


図3-56 ADV_IND イベントでの通信の流れ

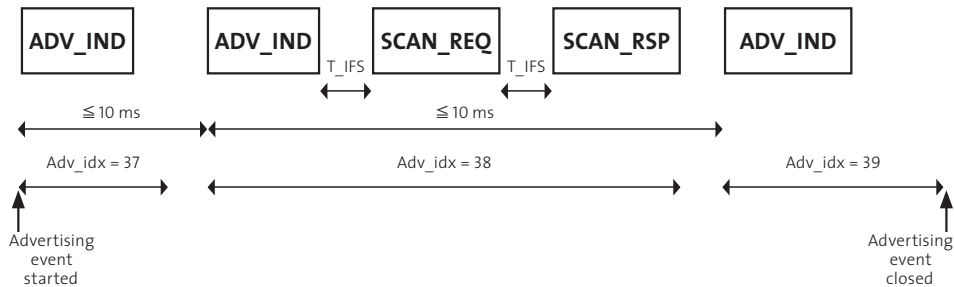


図3-57 ADV_IND イベントでの通信の流れ（Advertiserの相手がScannerの場合）（Advertiserの相手がScannerの場合）

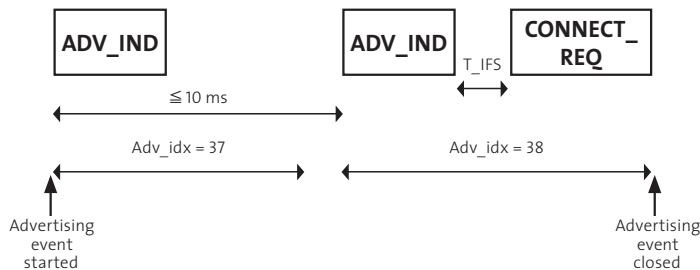


図3-58 ADV_IND イベントでの通信の流れ（Advertiserの相手がInitiatorの場合）（Advertiserの相手がInitiatorの場合）

ADV_DIRECT_IND (Connctable Directed Event) での通信

ADV_DIRECT_IND イベントにおける通信の流れの概略を図3-59に示します。このタイプのイベントでは、AdvertiserはInitiatorとの接続要求に応答することができます。

AdvertiserがADV_DIRECT_INDを送信後、InitiatorはCONNECT_REQ PDUを送信し、両デバイスがConnection状態に移行します。このとき、ScannerからSCAN_REQ PDUを受信したとしてもAdvertiserはこれを無視します。

なお、ADV_DIRECT_IND イベントでは Low Duty Cycle モードおよび High Duty Cycle モードの 2 つのモードに対応します。Low Duty Cycle モードは特定のデバイスとの再接続が要求されたときに、接続時間に関する要求が問題とならない場合に利用します。他方、High Duty Cycle モードは特定のデバイスとの再接続において、接続時間に対して余裕がなく、いち早い再接続が望まれる場合に利用します。High Duty Cycle モードでは、通常の Advertising での物理チャンネルの切り替え間隔に関する要求がさらに厳しくなるため、チャンネルの通信帯域が狭まると同時に電力の消費が増加する点に注意する必要があります。

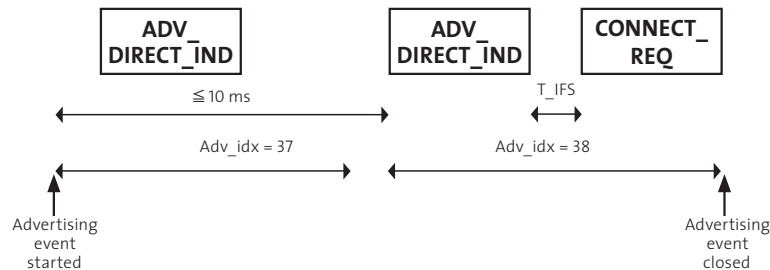


図3-59 ADV_DIRECT_IND イベントでの通信の流れ

Low Duty Cycleモードでの通信

Low Duty Cycleモードでの通信の概略を図3-60に示します。Low Duty Cycleでは、Advertisingの物理チャンネルの切り替え間隔が通常のAdvertisingと同様に10ms以下に設定されます。

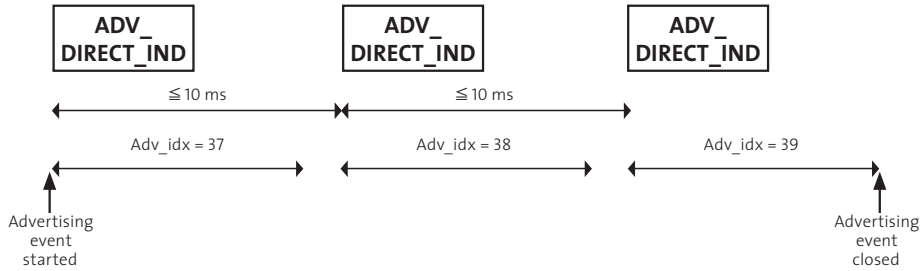


図3-60 Low Duty Cycle モードでの通信の流れ

High Duty Cycle モードでの通信

High Duty Cycle モードでの通信の概略を図3-61に示します。High Duty Cycleでは、Advertisingの物理チャンネルの切り替え間隔が通常のAdvertisingよりも高速となり、3.75ms以下に設定されます。また、このモードではAdvertiserのLL層はAdvertising Stateに切り替わってから1.28秒後にAdvertising Stateを強制的に終了します。

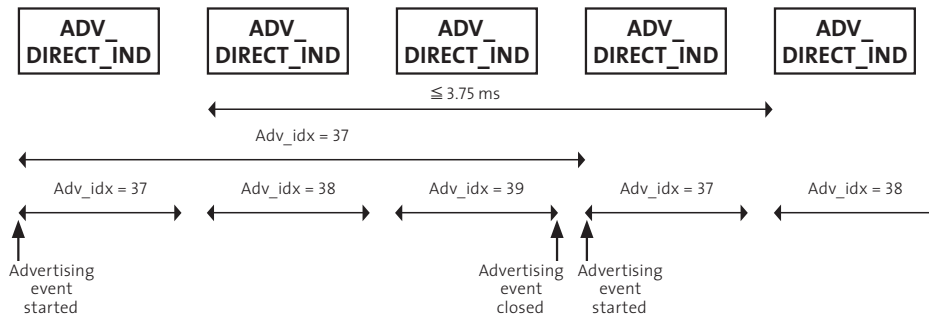


図3-61 High Duty Cycle モードでの通信の流れ

ADV_SCAN_IND (Scannable Undirected Event) での通信

ADV_SCAN_IND イベントにおける通信の流れの概略を図3-62に示します。このタイプのイベントでは、AdvertiserはScannerからのスキャン要求に応答することができます。

AdvertiserがADV_SCAN_INDを送信後、Scannerは応答としてSCAN_REQ PDUを送信します。SCAN_REQ PDUを受信したAdvertiserはその応答としてSCAN_RSP PDUを返信します。

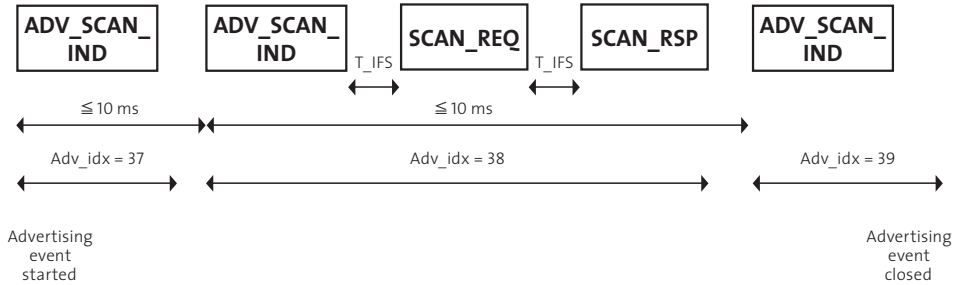


図 3-62 ADV_SCAN_IND イベントでの通信の流れ

ADV_NONCONN_IND (Non-connectable Undirected Event) での通信

ADV_NONCONN_IND イベントにおける通信の流れの概略を図 3-63 に示します。このタイプのイベントでは、Advertiser は Scanner、Initiator いずれのデバイスからの応答も受け付けません。

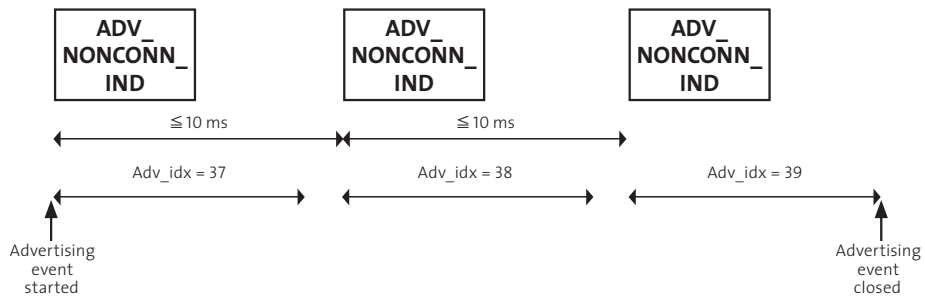


図 3-63 ADV_NONCONN_IND イベントでの通信の流れ

3-7-3. Data Channel における通信

次に、Data Channel における通信について解説します。Data Channel における物理チャネルの切り替えについては 3-4-1 項で解説したとおりです。本項では Data Channel PDU の確認応答 (ACK: Acknowledgement) とそのデータフローがどのように制御されているのかを解説します。

ACKは一般的な有線データ通信、たとえばI2Cなどのシリアル通信でも利用されている受信側からの確認応答で、データが正しく受信側に到達したか否かを受信側が送信側に伝えるために利用します。特にBluetoothなどの無線通信においては、通信の干渉や電波の強度不足などさまざまな要因で、データの送信が正しく完了できない可能性が有線通信と比較して格段に高く、ACKの重要性はさらに高いといえるでしょう。

BLEにおいてこのACKの役割を担っているのが、Data Channel PacketのHeader領域に含まれるSN (Sequence Number) およびNESN (Next Expected Sequence Number) の2つのパラメータで、このパラメータを送信側管理するtransmitSeqNum、受信側が管理するnextExpectedSeqNumによって制御することでACKの機能を実現させています。transmitSeqNumとnextExpectedSeqNumはいずれも1 bitのパラメータであり、この2つのパラメータはConnection Stateに遷移した時点でゼロ・クリアされます。

送信側のデバイスがData Channel PDUを送信する時、Header領域のSN bitにtransmitSeqNumの値をセットします（接続完了後、初回の送信ではSN = transmitSeqNum = 0となります）。パケット送信後、受信側からの応答を確認します。この受信側の応答に含まれるHeader領域のNESNがtransmitSeqNumと等しくない場合、デバイス間でパケットの送信が成功したとみなします。この確認の後、SN = transmitSeqNum+1として符号を反転させて更新します。更新後、SNを受信したNESNと組み合わせて再度Header領域にセットし、受信側に新しいパケットを送信します。仮にNESNがtransmitSeqNumと等しかった場合、送信側のデバイスは送信が失敗したとみなし、同じデータを再送します。

他方、受信側のデバイスではData Channel PDUを受信したとき、Header領域のSN bitとnextExpectedSeqNumを比較し、等しい場合はその受信データを新規のデータとして受信します（接続完了後、初回の受信ではnextExpectedSeqNum = SN = 0となり、必ず等しくなります）。この確認の後、NESN=nextExpectedSeqNum+1として符号を反転させて更新します。他方、逆にSN bitがnextExpectedSeqNumと等しくなかった場合、受信が失敗しているとみなし、受信パケットを破棄します。

以上の制御フローを図示すると図3-64となります。

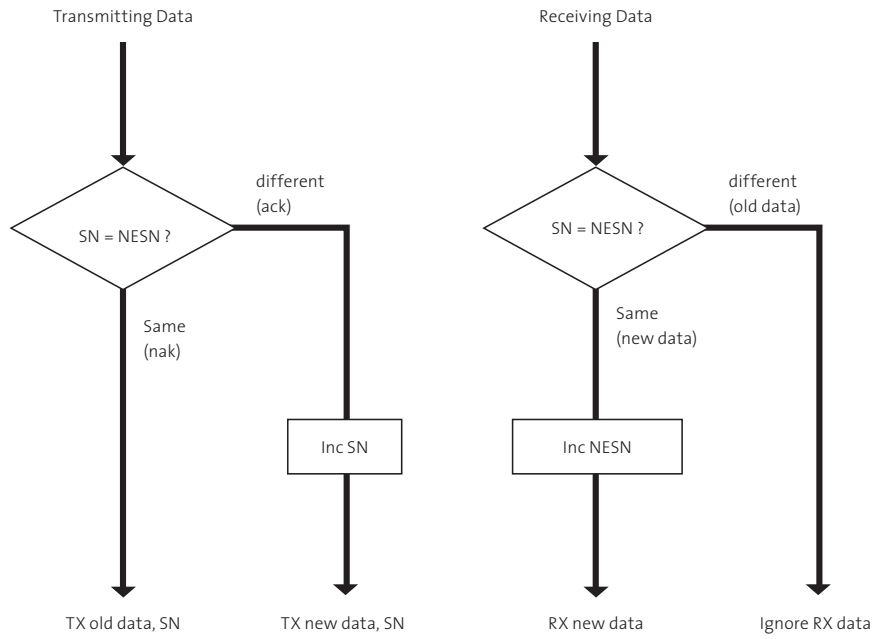


図 3-64 Data Channel における確認応答

1

2

3

4

5

6

7

8

9

10

11

12

3-8. GAP（Generic Access Profile）の詳細を知る

これまでの項でBLEデバイス同士がどのようなネットワークトポロジーを持つか、そしてそのネットワークの中でどのようなパケットがやり取りされているのか、その概略を示しました。本項では、それらを管理し、振る舞いを制御するGAP（Generic Access Profile）について解説します。GAPについては3-2-3項でも概略を述べましたが、本項ではその詳細を解説します。

3-8-1. GAPとは何か

GAPは、**Bluetooth**における振る舞いを包括的に定めたプロファイルであり、Bluetoothにおいて基盤となるプロファイルです。したがって、GAPはBLE独自のものではなくクラシックBTにおいても搭載されています。本書では混乱をさけるため、主にBLEについてのみ解説を行います。クラシックBTにおけるGAPの詳細についてはBluetooth仕様書（Bluetooth Core Specifications - Core Version 4.1 – Vol.3 - Part C）が参考になります。

さて、BLEにおいてGAPは、—BLEという市場の中でデバイスがどのような役割で振る舞うのかを管理する「市場の管理人」となるプロファイル（3-2-3項）—と述べましたが、具体的に管理している機能は**役割**、**動作**、**セキュリティ**の3つです。以上の機能は3-2-1節で示したプロトコルスタック上のプロトコル（LL、L2CAP、SMP、ATT）、プロファイル（GATT）のいずれにも関係が深いものです。図3-2に示すように、GAPはHost層からController層までを一気通貫に串刺したようなプロファイル構成となっています。このような一気通貫なプロファイルが3つの機能をそれぞれどのように管理しているのか、順を追って、その内容を見ていきましょう。

3-8-2. GAPによる「役割」の管理

GAPが管理する役割 (Role) は、**Broadcaster**、**Observer**、**Peripheral**、**Central**の4つです。BroadcasterとObserverはブロードキャスト型のトポロジー (3-3節) で利用する役割です。あるデバイスがAdvertising Packetを周辺のデバイスに送出するとき、送出する側のデバイスはBroadcasterに、Advertising Packetを受信する周辺のデバイスはObserverの役割がGAPによって与えられます (図3-65 (a))。

残るPeripheral、Centralは主に接続型のトポロジー (3-3節) で利用する役割です。あるデバイスが周辺のデバイスと双方向通信を確立するとき、トポロジー上でルートノード (根) になるデバイスはCentral、リーフノード (葉) になるデバイスはPeripheralの役割がGAPによって与えられます (図3-65 (b))。

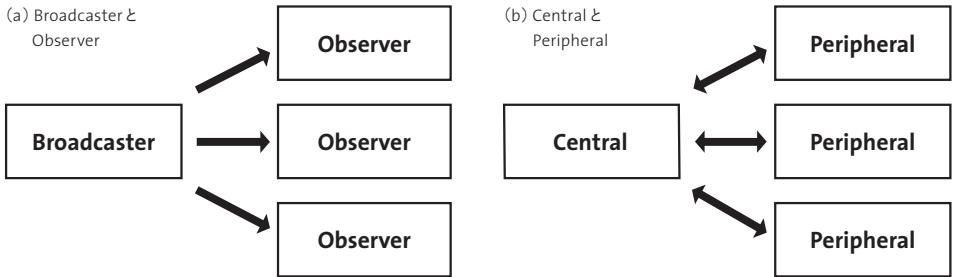


図3-65 GAPにおける役割 (Role) とトポロジーの違い

なお、BLEデバイスは同時に1つ以上の役割を担うことができます。つまり、あるタイミングでBroadcasterかつPeripheralであるように複数の役割を振る舞うことができ、仕様上も制限はありません。したがって、図3-66のような、ブロードキャスト型と接続型をミックスしたような混合型トポロジーも実現可能です。

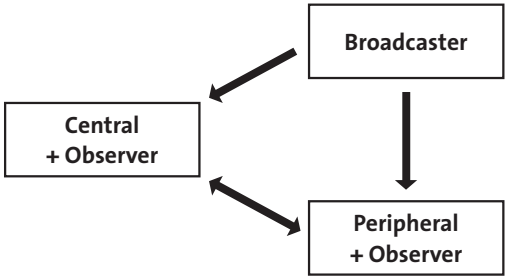


図3-66 役割をミックスした場合のトポロジー

それでは次に、4つの役割について個別にみていきましょう。

Broadcaster

Broadcasterはデータの送信に特化した役割で、ブロードキャスト型のトポロジーにおいて、Advertisement Packetを周囲に送出することがデバイスの目的となります。iOSエンジニアにとって身近なBroadcasterの応用例は、iOS7で実装された「iBeacon」でしょう。iBeaconはBLEデバイスをBroadcasterとして利用しており、送出したAdvertisement Packetに載せて周囲にユニークなデータを配信することで、BLEデバイスをランドマークとして利用することができます。

Observer

Observerはデータの受信に特化した役割で、ブロードキャスト型のトポロジーにおいて、BroadcasterからのAdvertisement Packetを受信することが主な目的となります。Observer側はBroadcasterを常にスキャンする立場になるため、**Broadcaster側と比較すると電力消費は大きくなります。**

Peripheral

Peripheralは、Central側のデバイスが自身を発見できるようにするためにAdvertisement Packetを送出し、Central側と接続を確立することが目的となります。iOSエンジニアから見ると、通常、iOSデバイスとBLEでつながる周辺デバイスがPeripheral側となります。

Central

Centralは、Peripheral側のデバイスから送出されたAdvertisement Packetをスキャンして受信し、Peripheral側と接続を確立することが目的となります。iOS開発者から見ると、通常はiOSデバイスがCentral側となります。

3-8-3. ModeとProcedure

GAPによる**動作**、**セキュリティ**の管理を説明する前に、両者で用いられている2つの概念**Mode**と**Procedure**について簡単に整理します。GAPではある役割における「動作」の状態をMode、Procedureの2つの概念で表現しています。ModeはBluetoothの特定のイベントに対して、デバイスが特定の手続きを行う状態を表す概念です。他方、Procedureは一連の動作の流れを定めた行為を表す概念です。GAPの役割（Role）とMode、Procedureとのそれぞれの関係は図3-67のように考えると捉えやすくなります。

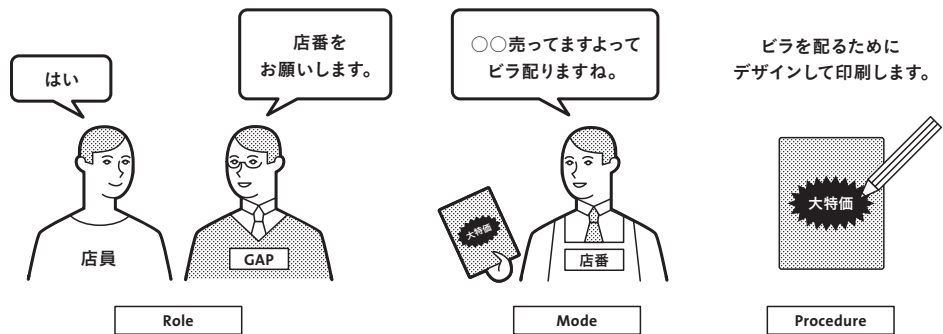


図 3-67 Gap における役割 (Role) と Mode、Procedure の関係

3-8-4. GAPによる「動作」の管理

BLEにおいてGAPで定義される「動作」は4種類あります。いずれの動作も Modeと Procedureの組み合わせで定義されており、これらの動作は同時に実行することができます。

- Broadcast mode および Observation procedure
- Discovery mode/procedure
- Connection mode/procedure
- Bonding mode/procedure

それぞれの動作は互いに独立していますが、通常、BLE デバイスが互いに通信するためにはこれらの動作が複雑かつ密接に連携することになります。これらの動作がBLEデバイスの役割

に対してどのように作用しているのか、順に確認していきましょう。

Broadcast mode および Observation procedure

Broadcast mode および Observation Procedure は、BLE デバイス間で Advertising などの単方向のブロードキャスト型の通信を行うイベントを司ります。GAP における役割 (Role) と、各 mode と Procedure との関係は次のようになります。

動作 (Mode / Procedure)	役割 (Role) ○=必須、×=不要	
	Broadcaster	Observer
Broadcast mode	○	×
Observation mode	×	○

表 3-13 Broadcast mode および Observation Procedure

Discovery mode および procedure

Discovery mode および procedure として定義されているものは次の表 3-14 のとおりです。これらの Mode および Procedure は、BLE デバイスの発見に関する動作をサポートするもので、いずれも Peripheral、Central のいずれかでサポートされます。

動作 (Mode / Procedure)	役割 (Role) ○=必須、×=不要	
	Broadcaster	Observer
Non-Discoverable mode	○	×
Limited Discoverable mode	Option	×
General Discoverable mode	Option	×
Limited Discovery procedure	×	Option
General Discovery procedure	×	○
Name Discovery procedure	Option1	Option

表 3-14 Discovery mode および procedure

Connection modeおよびprocedure

Connection modeおよびprocedureとして定義されているものは次の表3-15のとおりです。これらのModeおよびProcedureは、BLEデバイスの接続とそれに伴って発生する動作についてサポートするものです。

動作 (Mode / Procedure)	役割 (Role) ○=必須、×=不要			
	Peripheral	Central	Broadcaster	Observer
Non-connected mode	○	×	○	○
Directed connectable mode	Option	×	×	×
Undirected connectable mode	○	×	×	×
Auto connection establishment procedure	×	Option	×	×
General connection establishment procedure	×	Option	×	×
Selective connection establishment procedure	×	Option	×	×
Direct connected establishment procedure	×	○	×	×
Connection parameter update procedure	Option	○	×	×
Terminate connection procedure	×	○	×	×

表 3-15 Connection modeおよびprocedure

Bonding modeおよびprocedure

Bluetoothにおいて「ボンディング (Bonding)」とは、異なる2つのデバイスの接続を確立させる、具体的には**2つのデバイス間での共通鍵暗号を作成、交換、共有することで互いを信頼する作業**を指します。その名のとおり、共通鍵暗号という糊でデバイス間の糊付けを行うイメージです。なお、デバイスがボンディングについての情報を互いに共有した状態をBluetoothでは「devices have bonded」「a bond is created」と呼びます。

さて、Bonding modeおよびprocedureでは、BLEデバイス間でのボンディング処理に関する動作をサポートしています。これらは2つのデバイスがボンディング可能 (Bondable) かどうかを規定します。定義されているmodeおよびprocedureは表3-16のとおりです。本書では、セキュリティやプライバシーについて解説を割愛します。詳細はBluetooth Core Specification Part-C Section 9-4を参照してください。

動作 (Mode / Procedure)	役割 (Role) ○=必須、×=不要	
	Peripheral	Central
Non-Bondable mode	○	○
Bondable mode	Option	Option
Bonding procedure	Option	Option

表 3-16 Bonding mode および procedure

3-8-5. GAPによる「セキュリティ」の管理

一般的な通信ネットワークと同様、Bluetoothにおいてもデバイス間の接続に関してセキュリティ対策が施されています。BLEにおいては、GAPにHost側で**SMP (Security Manager Protocol)**層が組み込まれており、GAPによるセキュリティの管理をサポートしています。GAPによって定義されているmodeおよびprocedureは表3-17の通りです。

表3-17を見ると明らかですが、ブロードキャスト型のネットワークで利用されるPeripheralとCentralにおいては、セキュリティを要した接続がサポートされていないことがわかります。これはブロードキャスト型のネットワークがそもそも接続の手続きを取っていないためで、このネットワーク上でのデータのやり取りが、ブロードキャストのためのチャンネル（これをAdvertisement Channelと呼びます）にデータが流れてくるかどうか、だからです。

他方、コネクション型のネットワークでは、オプションとしてデバイス側が対応しているのであればセキュリティを確保した通信が可能です。

動作 (Mode / Procedure)	役割 (Role) ○=必須、×=不要			
	Peripheral	Central	Broadcaster	Observer
LE Security mode 1	×	×	Option	Option
LE Security mode 2	×	×	Option	Option
Authentication procedure	×	×	Option	Option
Authorization procedure	×	×	Option	Option
Connection data signing procedure	×	×	Option	Option
Authenticate signed data procedure	×	×	Option	Option

表 3-17 SMP (Security Manager Protocol) の mode および procedure

3-9. ATT (Attribute Protocol) と GATT (Generic Attribute Profile) の詳細を知る

前節までで、BLEデバイス同士がどのようなネットワークプロトコルを持ち、そのネットワークの中でどのような役割や振る舞いを行い、それがどう管理されているのかをGAPの機能の解説から概説しました。本節では、BLEにおいてiOSとの実質的な通信の窓口となってデータをやり取りする仕組みであるGATT (Generic Attribute Profile) と、そのデータに関しての規定したATT (Attribute Protocol) について概説します。まずはBLE上のデータ構造の基盤となるATTからひも解いていきましょう。

3-9-1. ATTとは何か

ATTについては3-2-1項で概略を述べましたが、本項ではその詳細を解説します。簡単に振り返ると、ATTは「Attribute」と呼ばれる独自の単位を定義し、それを元にしてデータのやり取りを行うサーバ/クライアント型のプロトコルです。サーバ側 (GATTサーバ) は構造化されたAttributeを公開し、通信相手となるクライアント側 (GATTクライアント) はその内容をReadもしくはWriteなどの処理を通じて参照します。

3-9-2. Attributeの構造

Attributeは、ATT上でやり取りするデータの最小単位であり、GATTサーバ上のデータはこのAttributeによって構造化されます。Attributeでは、データの「値 (Attribute Value)」の他に、「Attribute Type (UUID)」 「Attribute Handle」 「Permission」 の3つのプロパティのセットが1単位となっています。Attributeの構成を図3-68に示します。ちなみに図3-68はL2CAPで再構成された、あくまで論理レベルでの構成なので、実際のメモリ上の配置が必ずしも図と同じデータ構造になるものではありません。

1

2

3

4

5

6

7

8

9

10

11

12

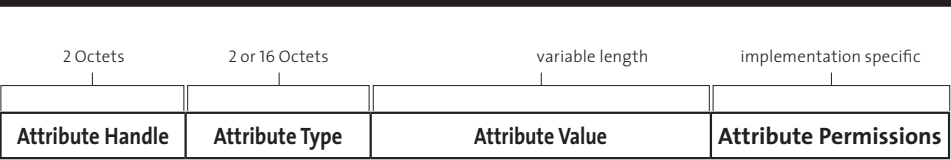


図 3-68 Attribute のパケット構成

それでは Attribute の内容を、順に追って、その構造を確認していきましょう。

Attribute Type (UUID)

Attribute Type は GATT サーバ上に展開されたデータの種類の示すもので、UUID (Universally Unique Identifier) によって記述されています。

UUID は 128bit (16octet) の長さの数値で表されたデータを一意に識別するための識別子です。ISO/IEC ISO/IE 9834-8:2005 において文書化、規格化されていますが、誰でも作成、配布が可能で必要に応じて公開することができます。しかし、BLE において 128bit 長のデータを識別のためだけにやり取りするのは全体のペイロード長からしても効率がよいとはいえません。そこで、より利用がしやすいように、Bluetooth では一定の長さの UUID をベース UUID (Bluetooth_Base_UUID) として確保し、用途に応じて 16bit もしくは 32bit の UUID を Attribute Type として運用しています。

ベース UUID (Bluetooth_Base_UUID) の値を 16 進数で表記すると図 3-69 のとおりです。

Bluetooth_Base_UUID :	00000000-0000-1000-8000-00805F9B34FB (16)
-----------------------	---

図 3-69 ベース UUID (Bluetooth_Base_UUID)

16bit/32bit の UUID から 128bit の UUID への変換は以下の式で定義されています。ちなみに、式中の 2^{96} は 96bit 分の左シフトと同義の演算となり、Bluetooth_Base_UUID の MSB の 32bit 分の 0 の部分に挿入される形となります。

$$128\text{bit UUID} = 16\text{ bit UUID} \times 2^{96} + \text{Bluetooth_Base_UUID} \text{ (eq.1)}$$

$$128\text{bit UUID} = 32\text{ bit UUID} \times 2^{96} + \text{Bluetooth_Base_UUID} \text{ (eq.2)}$$

16bit UUID は他の 16bit UUID と、32bit UUID は他の 32bit UUID と直接比較することが可能で、Bluetooth では比較に際して 128bit の UUID に変換する必要を求めています。しかし、

16bitもしくは32bitのUUIDを128bitのUUIDと比較する際は、16bit/32bitのUUIDを128bit UUIDに変換した値を用いなければならず、その128bit UUIDは式 eq.1、eq.2を必ず満たしている必要があります。

一方、Application側すなわちiOS開発者側で、他の128bit UUIDを比較するために式eq.1およびeq.2を用いて、16bit/32bit UUIDを逆変換によって生成することはBluetoothの仕様上で禁止されています。これは与えられた128bit UUIDがBluetooth_BASE_UUIDに基づくものである保証がないこと、他の開発者が独自に定義した128bit UUIDである可能性があることが理由です。

いくつかの16bit/32bit UUIDはBluetooth SIGによって規定値として割り当てられています。割り当てられたUUIDはAssigned Numbersとして、Bluetooth SIGのサイト上^{※6}から確認することができます。

Attribute Handle

Attribute HandleはGATTサーバ上でAttributeを互いに識別するために利用する16bitの識別子です。Attribute Handleは、Attributeのアドレスともいえるパラメータで、Bluetoothデバイス間のやり取りの中で変化することはありません。

Attribute Handleの値については特にAttribute Typeのような制限はありませんが、その値は0(0x0000)がBluetoothの仕様上で確保されているため、必ず0でない値を取ることになります。また、値の最大値は0xFFFFであるため、GATTサーバが扱うことができるAttribute Handleは0x0001～0xFFFFまでの65534(=0xFFFE)個となります。

Attribute Value

Attribute Valueはその名のとおりAttributeの値を格納したパラメータです。データは固定長、可変長いずれでも定義されており、4octetの整数値や可変長の文字列など多様な値が代入されます(可変長のデータで単一のパケットで送信するには大きすぎる場合、複数のパケットに分解して送信されます)。Attribute Valueそれ自体では目的や種類が一意に定まることはなく、どのようなものであるかはAttribute Typeで定義されてはじめて一意に決定します。

※6 Bluetooth SIG Website – Assigned Numbers : <https://www.bluetooth.org/en-us/specification/assigned-numbers>

Attribute Permission

Attribute Permission は、サーバ/クライアント間での Attribute へのアクセス権やセキュリティレベルについて規定したパラメータになります。Attribute Permission はより上位の層で管理されている値であるため、ATT から編集やその値を確認することはできません。規定されている Permission は次の表 3-18 のとおりです。

区分	Attribute Permissions
Access permissions	Readable
	Writable
	Readable & Writable
Encryption permissions	Encryption Required
	No encryption Required
Authentication Permissions	Authentication Required
	No Authentication Required
Authorization Permissions	Authorization Required
	No Authorization Required

表 3-18 Attribute Permission

MTU (Maximum Transfer Unit)

MTU(Maximum Transfer Unit)とは、一般的に通信においてデバイスが送信可能なパケットの最大量を表します。ATTにおいてもATT_MTUとしてサーバ/クライアント間でのAttributeのサイズの最大量が定義されています。ATT_MTUの値については、ATTではなく上位アプリケーションによって定義されます。このATT_MTUは動作オプションとして、特定のリクエスト(Exchange MTU Request)を送信することで拡張することができます。

単一のパケットで送信することができる最大のAttributeのパケットサイズは(ATTT_MTU-1)octetとなります。Attribute Valueが(ATTT_MTU-1)octetよりも大きく定義される場合、このようなAttributeはLong Attributeと呼ばれます。なお、Attribute Valueの規格上の最大値は512octetと定義されています。また、デフォルト値はATT_MTU_{default}=23octetとなります。

3-9-3. ATTサーバ/クライアントの対応するメソッドとPDU

ATTでは、サーバ/クライアント間でのAttributeのやり取りをReques、Response、Command、Notification、Indication、Confirmationの6種類のメソッドで定義しています。各メソッドの定義するデータのやり取りの概要を図3-70に示します。

クライアントがサーバに対して問い合わせるメソッドをRequest、それに応じてサーバがクライアントに返答するメソッドをResponseと呼びます。RequestとResponseは一对のメソッドとなっています。一方、このResponseを必要としないクライアントからサーバへの問合せをCommandと呼びます。逆にサーバからクライアントへの問い合わせをNotificationと呼びます。サーバからクライアントに問い合わせを行い、さらにクライアントからの応答を求める場合、サーバからの問い合わせをIndication、クライアントからの応答をConfirmationと呼びます。

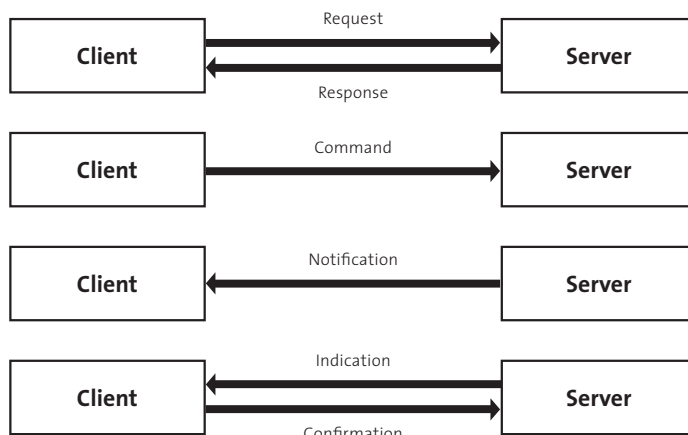


図3-70 ATTサーバ/クライアントの対応するメソッドとPDU

Attribute PDU

サーバ/クライアント間のAttributeのやり取りは、Attribute PDUと呼ばれるPDUを介して行われます。また、より上位に位置するGATTの利用するメソッドごとにAttribute PDUの仕様をまとめたものをAttribute Protocol PDUと呼びます。なおAttribute PDUは、3-6-6項で述べたData Channel PDUにおいて、LL_DATA PDUが指定されたバケットをL2CAPによって再構成したものとなります。

Attribute PDUのフォーマット

Attribute PDUのフォーマットは極めてシンプルです。基本的なPDUの形式としては、ヘッダであるAttribute OpcodeとペイロードであるAttribute Parameterの2つの領域で構成され、認証を要する書き込みを行う場合にオプションとしてAuthentication Signatureが追加されます。Authentication Signatureの有無によらず、Attribute PDU全体のサイズはATT_MTU octetで固定となります（ATT_MTUについては3-6-8項を参照してください）。

Attribute PDU (ATT_MTU octet)		
Attribute Opcode (1octet)	Attribute Parameter (0~ATT_MTU-X octet)	Authentication Signature (12octet)

図 3-71 Attribute PDUのパケットフォーマット

ヘッダであるAttribute Opcodeは1octetのビットフィールドとなっており、その内訳は図 3-72に示すとおりです。bit7のAuthentication Signatureに関するフラグの有無で続くAttribute Parameterのサイズ、およびAuthentication Signatureの有無が決定します。

Attribute Opcodeのbit7がクリアされている場合、Authentication Signatureは付与されず、 $X = \text{Attribute Opcode}$ となり、Attribute Parameterのサイズは $(\text{ATT_MTU} - X) = (\text{ATT_MTU} - 1)$ octetとなります。他方、bit7がセットされている場合、Authentication Signatureの12octetが確保されるため、 $X = \text{Attribute Opcode} + \text{Authentication Signature} = 12 + 1$ となり、Attribute Parameterは $(\text{ATT_MTU} - X) = (\text{ATT_MTU} - 13)$ octetとなります。

Attribute Opcode							
bit7	bit6	bit5	bit4	bit3	bit2	bit1	Bit0
Authentication Signature	Command Flg	Method					

図 3-72 Attribute Opcode

では、このAttribute PDUがメソッドごとにどのようなAttribute Protocol PDUとして利用されるのかを次項で確認していきましょう。

Attribute Protocol PDUのフォーマット

Attribute PDUは、GATTが利用するATTのメソッドに応じてAttribute Protocol PDUとして細分化されています。本項では、Attribute Protocol PDUをメソッド中の処理内容に応じて確認していきます。

Error Response: エラー処理を返却する

NotificationやResponseを求めないデータの読み込み/書き込みを行う場合をのぞいて、各メソッド間のやり取りでは、Requestに対してResponseが返信されます。Responseには後述するようにさまざまな種類がありますが、その中で与えられたRequestに対して問題が発生した場合のエラー処理に用いられるAttribute Protocol PDUがError Responseです。Error Responseのフォーマットを図3-73に示します。

Error Response			
Attribute Opcode (1octet)	Request Opcode in Error (1octet)	Attribute Handle in Error (2octet)	Error Code (1octet)

図3-73 Error Response

フォーマットの各領域を簡単に説明すると、Error ResponseのOpcodeは0x01となります。Request Opcode in Errorはエラーの発端となったRequestのOpcodeが、Attribute Handle in Errorはエラーの発端となったAttribute Handleが指定されます。

それぞれ、OpcodeとAttribute Handleが返却されるので当然、それぞれのサイズはRequest Opcode in Errorが1octet、Attribute Handle in Errorが2octetとなります。最後のError Codeにエラー内容についてのフラグが与えられます。Error Codeの内容については、次の表3-19のとおりです。

Error Code 名	値
Invalid Handle	0x01
Read Not Permitted	0x02
Write Not Permitted	0x03
Invalid PDU	0x04
Insufficient Authentication	0x05
Request Not Supported	0x06
Invalid Offset	0x07
Insufficient Authorization	0x08
Prepare Queue Full	0x09
Attribute Not Found	0x0A
Attribute Not Long	0x0B
Insufficient Encryption Key	0x0C
Invalid Attribute Value Length	0x0D
Unlikely Error	0x0E
Insufficient Encryption	0x0F
Unsupported Group Type	0x10
Insufficient Resources	0x11
Reserved	0x12～0x7F
Application Error	0x80～0x9F
Reserved	0xA0～0xDF
Common Profile and Service Error Code	0xE0～0xFF

表 3-19 Error Code の内容

Exchange MTU Request / Exchange MTU Response: MTUの容量を変更する

サーバ/クライアント間のメソッドにおいて扱うことができる Attribute の最大サイズを ATT_MTU と呼びます（定義については 3-6-8 項を参照ください）。ATT_MTU は、通信のペアとなったデバイス間で通信中に一度だけ再定義することができ、クライアント側からサーバ側に対して ATT_MTU のサイズの変更を要求することができます。この操作を行う Attribute Protocol PDU が Exchange MTU Request および Exchange MTU Response です。

Exchange MTU Request のフォーマットを図 3-74 に示します。Attribute Opcode は

Exchange MTU Requestを示す0x02にセットされます。Client Rx MTUはクライアントが変更を希望するATT_MTUのサイズが与えられ、この値がクライアント側でその後扱うことができるATT_MTUの最大値となります。

なお、Client Rx MTUの値はデフォルト値（23octet）と等しいか、それ以上の値をとります。これはデフォルト値以下の場合、そもそもExchange MTU Requestを利用する必要がないためで、逆に考えると、デフォルト値以下のATT_MTUにすることは規格上できないことを意味しています。デフォルト値以下のATT_MTUを送信した場合、ATT_MTUの変更は行われません。

Exchange MTU Request	
Attribute Opcode (1octet)	Client Rx MTU (2octet)

図 3-74 Exchange MTU Request の Attribute Protocol PDU のフォーマット

クライアントからのリクエストに応答し、サーバはExchange MTU Responseをレスポンスとして返信します。Exchange MTU Responseのフォーマットを図3-75に示します。基本的にはRequestとフォーマットの内容は変わりませんが、Attribute OpcodeはExchange MTU Responseを示す0x03にセットされます。Server Rx MTUはサーバ側でその後扱うことができるATT_MTUの最大値となります。またClient Rx MTU同様、Server Rx MTUの値はデフォルト値（23 octet）と等しいか、それ以上の値をとります。

Exchange MTU Response	
Attribute Opcode (1octet)	Server Rx MTU (2octet)

図 3-75 Exchange MTU Response の Attribute Protocol PDU のフォーマット

なおATT_MTUの変更について利用上の注意点をあげておきます。

- ATT_MTUのサーバ/クライアント間の対称性
- ATT_MTUの変更前後での通信処理
- ATT_MTU変更前後での Notification、Indication の扱い

以下、見ていきましょう。

ATT_MTUのサーバ/クライアント間の対称性

サーバ/クライアント間で取り扱うことができるATT_MTUの最大値が異なる場合、つまり Client Rx MTU \neq Server Rx MTUであった場合、低い値のほうでATT_MTUの変更が行われます。したがって、接続されているBLEデバイス間のATT_MTUがサーバ側とクライアント側で異なることはありません（サーバ/クライアント間では常にATT_MTUの対称性が保たれます）。これは、クライアントがLong Attributeを受診する際の最終バケットサイズを確実に規定するためです。

また、接続しているデバイスが双方でサーバ/クライアントのいずれにもなり得る場合、一方の組み合わせ、たとえばデバイスAがクライアント、デバイスBがサーバの状態で、AからBに対してExchange MTU Requestが要求され、かつATT_MTU = MTU_{Changed}の変更が成立した場合、その後にBがクライアント、Aがサーバとなって通信が行われる場合も、先に変更されたMTU_{Changed}での通信となります（なお規格上、双方からExchange MTU Requestを要求することは認められています。しかしながら、この場合においてもATT_MTUの対称性は常に保たれていなければならないため、双方からExchange MTU Requestを利用することは意味がない冗長な操作といえるでしょう）。

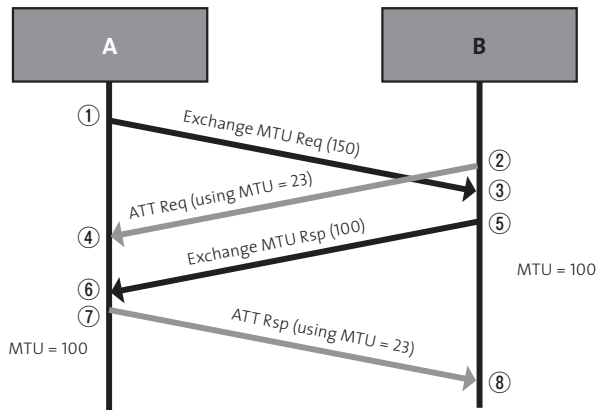


図 3-76 ATT_MTU の変更とその前後での通信処理

ATT_MTUの変更前後での通信処理

すべてのAttribute Protocolにおいて、あるRequestに対するResponseは、Request送信時点でのATT_MTUにしたがってResponseが返信される点に注意が必要です。図3-76に一例を示します。

先に説明したように、クライアント側によるExchange MTU RequestによるMTUの変更要

求は、サーバ側の Exchange MTU Response の発行をトリガーにして、双方でその変更が反映されます。仮に、接続しているデバイスが双方でサーバ/クライアントのいずれにもなり得る場合、一方の組み合わせ、たとえばデバイス A がクライアント、デバイス B がサーバの状態、A から B に対して Exchange MTU Request が要求されたとします。その際、Exchange MTU Request の受信よりも前に B がクライアントとして A に対して何らかの Attribute Protocol に関するリクエストを送信した場合、このリクエストにおける ATT_MTU は、当然、変更前のデフォルト値 (=23octet) となります。A はサーバとして Attribute Protocol に関するリクエストを受け、その後、Exchange MTU Request に関するレスポンスを受けることになりますが、このとき、先に受けた Attribute Protocol に対するレスポンスでの ATT_MTU は変更前のデフォルト値が適応されることになります。

ATT_MTU 変更前後での Notification、Indication の扱い

クライアントとして振舞うデバイスが Exchange MTU Request を発行して ATT_MTU の変更を行った後、サーバとして Notification もしくは Indication を送信するケースも考えられます。この場合、Notification および Indication は、接続先の相手からの Exchange MTU Response を受信するまでは発信を停止するように規定されています。これは ATT_MTU が不定の状態では Notification や Indication を送信するリスクを回避するためです。たとえば、Exchange MTU Request で $ATT_MTU_{Client}=100$ octet として発行したとしても、サーバ側がデフォルト値である $ATT_MTU_{Server}=23$ octet しか認めないデバイスだった場合、互いの要求に齟齬が発生するためです。

これらのことからわかるように、ATT_MTU の変更に関しては iOS 開発者の一存で対応することはできません。ATT_MTU の変更が必要な場合はデバイスの設計者も含めて議論するとよいでしょう。

サーバ上の Attribute Type を検索する

サーバ上に展開されている Attribute について、どのような Attribute 値を持つのかを検索するために、ATT では Find Information Request / Find Information Response と、Find by Type Value Request / Find by Type Value Response の 2 対の Attribute Protocol PDU を定義しています。

Find Information Request / Find Information Response

サーバ上の特定の範囲内の Attribute のリストとそれらの Attribute 値を検索する Attribute

Protocol PDUを、Find Information Request および Find Information Response と呼びます。

Find Information Request の Attribute Protocol PDU のフォーマットを図3-77に示します。Attribute OpcodeはFind Information Requestを示す0x04が与えられます。続く Starting Handle および Ending Handle は GATT サーバ上に展開される Attribute の検索範囲を示し、検索範囲は Attribute Handle をもとにして指定します。仮にサーバ上のすべての Attribute の Attribute 値を検索する場合、Starting Handle=0x0001、Ending Handle=0xFFFF と与えることで検索することができます。1つ以上の検索結果が得られると、サーバはFind Information Responseを返信します。

Find Information Request		
Attribute Opcode (1octet)	Starting Handle (2octet)	Ending Handle (2octet)

図 3-77 Find Information Request の Attribute Protocol PDU のフォーマット

Find Information Response の Attribute Protocol PDU のフォーマットを図3-78に示します。Attribute OpcodeはFind Information Responseを示す0x05が付与されます。Format には UUID のサイズフォーマット（16bit or 128bit）についての指定が挿入され、続く Information Data には検索された Attribute Handle と Attribute 値とのデータセットのリストが返信されます。Format の値と Information Data の関係について表3-21に示します。

Find Information Response		
Attribute Opcode (1octet)	Format (1octet)	Information Data (4～(ATT_MTU-2) octet)

図 3-78 Find Information Response の Attribute Protocol PDU のフォーマット

Format 値	Information Data の形式	Information Data (1セットあたりの消費サイズ)
0x01	Attribute Handle (2 octet) と Attribute 値 (16 bit Bluetooth UUID) (2 octet)	2 + 2 = 4 octet
0x02	Attribute Handle (2 octet) と Attribute 値 (128 bit UUID) (16 octet)	2 + 16 = 18 octet

表 3-20 Format の値と Information Data の関係

表3-20に示すように、Format の値によって続く Information Data の形式が決定します。Format=0x01ではAttribute Typeは16bit BluetoothUUIDを、Format=0x02では128bit UUIDを対象とします。したがって、Information Dataは4octet (Format=0x01の場合の最小のデータセットのサイズ) から [ATT_MTU-2]octet (ATT_MTU - Attribute Opcode - Format) の

可変長となります。

なお、GATTサーバ上の Attribute 値は16bitと128bitがサポートされていますが、BLEの仕様上ではサーバ上の Attribute 値はいずれかに統一していることが望ましいとされています。これはレスポンス中の Information Data において異なる型の UUID を利用した Attribute が存在すると、「Attribute Handle、Attribute 値」のペアで消費する容量が不定となり、リスト中のペアの最大数が予測できなくなるためです。したがって Attribute の境界で UUID が異なる場合、その境界で Find Information Request を区切り、境界での Attribute Handle+1 を新たな Starting Handle として更新し、再度、Find Information Request を行うこととなります。

ちなみに、16bit の場合は $(ATT_MTU-2)/4$ ペア、128bit の場合は $(ATT_MTU-2)/18$ ペアとなります。

Find by Type Value Request / Find by Type Value Response

サーバ上の Attribute に対し、Attribute Type と UUID (Attribute 値) を指定して検索を行う Attribute Protocol PDU を Find by Type Value Request、および Find by Type Value Response と呼びます。

Find by Type Value Request の Attribute Protocol PDU のフォーマットを図 3-80 に示します。Attribute Opcode は Find by Type Value Request を示す 0x06 が与えられます。続く Starting Handle および Ending Handle は GATT サーバ上に展開される Attribute の検索範囲を示し、検索範囲は Attribute Handle で指定します。仮にサーバ上のすべての Attribute の Attribute 値を検索する場合、Starting Handle=0x0001、Ending Handle=0xFFFF と与えることで検索することができます。ここまでは先の Find Information Request と同様です。Find by Type Value Request では、それに加え Attribute Type と Attribute Value を検索条件として付与します。Attribute Type は 2octet の UUID、Attribute Value は発見したい UUID 値です。1つ以上の検索結果が得られると、サーバは Find by Type Value Response を返信します。

ちなみに、Find by Type Value Request では Attribute Value に与えられる UUID の最大サイズですが、ATT_MTU のデフォルト値の $ATT_MTU_{default} = 23octet$ となるため、デフォルトの状態では 128bit UUID 分の 16octet (= 23octet - 7octet) は最低限、確保されています。一方、 $(ATT_MTU-7)octet$ 以上の値を利用することはできないことがわかります。

Find by Type Value Request				
Attribute Opcode (1octet)	Starting Handle (2octet)	Ending Handle (2octet)	Attribute Type (2octet)	Attribute Value (0~(ATT_MTU-7) octet)

図 3-79 Find by Type Value Request の Attribute Protocol PDU のフォーマット

Find by Type Value Response の Attribute Protocol PDU のフォーマットを図 3-81 に示します。Attribute Opcode は Find by Type Value Response を示す 0x07 が付与されます。続く Handles Information List には検索条件に合致する Attribute の発見地点 (Found Attribute Handle) と終了地点 (Group End Handle) のペアのリストが返信されます。Handles Information List のパケットフォーマットについて図 3-81 に示します。

Find by Type Value Response	
Attribute Opcode (1octet)	Handles Information List (4~(ATT_MTU-1) octet)

図 3-80 Find by Type Value Response の Attribute Protocol PDU のフォーマット

Find by Type Value Response	
Attribute Opcode (1octet)	Handles Information List (4~(ATT_MTU-1) octet)

図 3-81 Handles Information List の内訳

Handles Information List の Group End Handle において終了地点を決定するのは、上位の GATT プロファイルの Service の設計に依存します。この Service の設計によっては、リクエストで指定した Ending Handle よりも大きい Group End Handle が返却される可能性があります。なお、GATT プロファイル上で適切な設定が行われていない場合、Found Attribute Handle と Group End Handle の値は一致します。

サーバ上の Attribute の値を読み出す

サーバ上の Attribute を読み出すための Attribute Protocol PDU として、ATT では表 3-21 に示す 5 対を定義しています。それぞれの違いについては表 3-21 のとおりです。

Attribute Protocol	内容
Read by Type Request / Read by Type Response	Attribute Handleの範囲と Attribute Type を指定して読み出す
Read Request / Read Response	Attribute Handleから直接、読み出す
Read Blob Request / Read Blob Response	Attribute HandleからLong Attributeを読み出す
Read Multiple Request / Read Multiple Response	複数の Attribute Handle から直接、読み出す
Read by Group Type Request / Read by Group Type Response	Attribute Handleの範囲と Attribute Group Type を指定して読み出す

表 3-21 Format の値と Information Data の関係

Read by Type Request / Read by Type Response

サーバ上の Attribute の Attribute Type を検索条件として Attribute を検索し、その値を読み込む Attribute Protocol PDU を Read by Type Request と Read by Type Response と呼びます。

Read by Type Request の Attribute Protocol PDU のフォーマットを図 3-82 に示します。Attribute Opcode は Read by Type Request を示す 0x08 が与えられます。続く Starting Handle および Ending Handle は GATT サーバ上に展開される Attribute の検索範囲を示し、検索範囲は Attribute Handle をもとにして指定します。仮にサーバ上のすべての Attribute の Attribute 値を検索する場合、Starting Handle=0x0001、Ending Handle=0xFFFF と与えることで検索することができます。続く Attribute Type は検索条件として利用されます。Attribute Type の指定は 16bit Bluetooth UUID、もしくは 128bit UUID で指定します。以上の条件で検索対象が存在し、かつその Attribute 値が得られると、サーバは Read by Type Response を返信します。

Read by Type Request			
Attribute Opcode (1octet)	Starting Handle (2octet)	Ending Handle (2octet)	Attribute Type (2 or 16octet UUID)

図 3-82 Read by Type Request の Attribute Protocol PDU のフォーマット

Read by Type Response の Attribute Protocol PDU のフォーマットを図 3-83 に示します。Attribute Opcode は Read by Type Response を示す 0x09 が与えられます。Length には続く Attribute Data List 中の Attribute1 セットあたりのサイズサイズが、Attribute Data List には Read by Type Request で指定した検索条件に合致する Attribute の Attribute Value が提供されます。Attribute Data List のフォーマットを図 3-84 に示します。

Read by Type Response		
Attribute Opcode (1octet)	Length (1octet)	Attribute Data List (2~(ATT_MTU-2) octet)

図 3-83 Read by Type Response の Attribute Protocol PDU のフォーマット

Attribute Data List (Read by Type Response)	
Attribute Handle (2octet)	Attribute Value (Length-2octet)

図 3-84 Attribute Data List のフォーマット

Attribute Data List では、たとえば読み込んだ Attribute が GATT 上の Include 宣言だった場合 (Include 宣言については 3-10-1 節の「Include の定義」、147 ページ参照)、Attribute Value として Include Service Attribute Handle(2octet)、End Group Handle(2octet)、Service UUID (仮に 16bit BluetoothUUID として 2octet) 以上の 3 つのフィールドを持つ Attribute 値 (合計 6octet) が返却されます。この場合、Length は

$$\text{Length} = \text{Attribute Handle (2octet)} + \text{Attribute Value(6octet)} = 8\text{octet}$$

となり、Length = 0x08 が代入されます。ちなみに Attribute Data List で読み出すことができるサイズは 255octet に制限されるため、実際に読み込むことが可能な Attribute 値の最大サイズは

$$\text{Size}_{\text{Actual Attribute Value}} = \text{Size}_{\text{max}} - \text{Attribute Handle} = 255\text{octet} - 2\text{octet} = 253\text{octet}$$

となり、253octet となります。なお、Attribute Protocol PDU 全体から見て Attribute Data List が確保できる仕様上での最大サイズは図 3-86 からわかるように $(\text{ATT_MTU} - 2)\text{octet}$ 、さらにその中の Attribute Value は

$$(\text{ATT_MTU} - 4)\text{ octet} = (\text{ATT_MTU} - 2) - \text{Attribute Handle} = (\text{ATT_MTU} - 2) - 2$$

となりますが、 $(\text{ATT_MTU} - 4)$ の値が 253 を超える場合は、想定する $(\text{ATT_MTU} - 4)\text{octet}$ の Attribute 値の内の最初の 253octet が返却されます。残る Attribute 値を読み出すには、後述する Read Blob Request を利用します。

Read Request / Read Response

Attribute Handleからサーバ上のAttributeを読み出す処理に利用するAttribute Protocol PDUをRead RequestとRead Responseと呼びます。こちらのAttribute Protocol PDUでは検索条件を設けず、Attribute Handleを直接指定して値を読み出します。したがって、Attribute Protocol PDUの内容も極めてシンプルです。

Read RequestのAttribute Protocol PDUのフォーマットを図3-85に示します。Attribute OpcodeはRead Requestを示す0x0Aが与えられます。Attribute Handleには読み込みたいAttributeのアドレス(Handle)が与えられます。サーバは読み込みが成功するとRead Responseを返却します。

Read Request	
Attribute Opcode (1octet)	Attribute Handle (2octet)

図 3-85 Read Request の Attribute Protocol PDU のフォーマット

Read Response	
Attribute Opcode (1octet)	Attribute Value (0 ~ (ATT_MTU-1) octet)

図 3-86 Read Response の Attribute Protocol PDU のフォーマット

Read ResponseのAttribute Protocol PDUのフォーマットを図3-86に示します。Attribute OpcodeはRead Responseを示す0x0Bが与えられます。続くAttribute ValueはRead Requestによって指定されたAttribute Handleが示すAttributeの値です。Attribute Valueで読み込みが可能なサイズはOpcodeで消費された分を差し引いた(ATT_MTU-1)octetとなります。[(ATT_MTU-1)octet以上のAttribute値 (Long Attribute)を読み出す場合、読み出される値は[(ATT_MTU-1)octet分のみとなるため、それ以上のサイズの値を読み出す場合は、後述するRead Blob RequestをRead Requestに加えて利用します。

Read Blob Request / Read Blob Response

先ほどのRead RequestではAttributeから読み出すことができるサイズは[(ATT_MTU - 1) octetに制限されていました。このような[(ATT_MTU-1)octetに納まらないサイズ(これを仮にLengthRequestedとしましょう)のAttributeを読み出すには2つの方法が用意されています。

1つは、先のExchange MTU Requestを利用してATT_MTUのサイズそのものを拡張する方法です。[LengthRequested + 1)octet以上のサイズに拡張することで、所望のAttribute

から値を取り出すことができます。この方法の制約としては、接続されているデバイス間で $[\text{LengthRequested} + 1]\text{octet}$ 以上の値が ATT_MTU で確保可能である必要があります。

もう1つの方法は、Read Blob Request を利用する方法です。Read Blob Request では指定の Attribute Handle の Attribute 値を指定の位置から読み出す処理を行う Attribute Protocol PDU です。したがって、Read Request 後に Read Blob Request で $[\text{ATT_MTU}-1]\text{octet}$ 分だけオフセットした位置から連続で読み出し、LengthRequested を超えるまで繰り返せば $[\text{ATT_MTU}-1]\text{octet}$ 以上の Attribute 値であっても読み出すことが可能となります。Read Blob Request の利点は、ATT_MTU の拡張なしに利用することができるため、接続先のデバイスの仕様を気にすることなく（たとえば、接続先のデバイスの ATT_MTU が拡張できない場合など）利用することができます。

Read Blob Request の Attribute Protocol PDU のフォーマットを図3-87に示します。Attribute Opcode は Read Blob Request を示す 0x0C が与えられます。Attribute Handle は値を読み出したい Attribute のアドレス（Handle）を、Value Offset は読み出す Attribute 値のオフセット量を示します。値が正しく読み出された場合、サーバから Read Blob Response が返信されます。

Read Blob Request		
Attribute Opcode (1octet)	Attribute Handle (2octet)	Value Offset (2octet)

図 3-87 Read Blob Request の Attribute Protocol PDU のフォーマット

Read Blob Response の Attribute Protocol PDU のフォーマットを図3-88に示します。Attribute Opcode は Read Blob Response を示す 0x0D が与えられます。Attribute Value には Read Blob Request で指定した Attribute 値の一部が与えられます。

Read Blob Response	
Attribute Opcode (1octet)	Attribute Value (0~(ATT_MTU-1) octet)

図 3-88 Read Blob Response の Attribute Protocol PDU のフォーマット

なお、Read Request + Read Blob Request による Long Attribute の読み出しでは、その処理中に Attribute の変更が加えられた場合、変更後の値を読み出すこととなる点に注意が必要です。したがって、GATT プロファイルで Long Attribute を利用する場合、この点を考慮して設計する必要があります。

Read Multiple Request / Read Multiple Response

以上の項ではLong Attributeを扱う方法を解説しましたが、複数の Attribute を同時に読み出す Attribute Protocol PDU も存在します。それが Read Multiple Request と Read Multiple Response です。

Read Multiple Request の Attribute Protocol PDU のフォーマットを図3-89に示します。Attribute Opcode は Read by Multiple Request を示す 0x0E が与えられます。続く Set of Handles には、読み出したい Attribute の Attribute Handle のリストを与えます。なお、Read Multiple Request は読み出したい Attribute の数 $N_{\text{Attribute}} \geq 2$ である必要があります ($N_{\text{Attribute}}$ が 2 未満、すなわち $N_{\text{Attribute}} = 1$ であれば Read Request で十分なためです)。したがって、Set of Handle のサイズは Attribute Handle 2 個分のサイズ、すなわち $2 \text{ octet} \times 2 = 4 \text{ octet}$ が最小値となります。サーバは、正しく値が読み出せると Read Multiple Response で複数の Attribute 値を返信します。

Read Multiple Request	
Attribute Opcode (1octet)	Set of Handles (2~(ATT_MTU-1) octet)

図 3-89 Read Multiple Request の Attribute Protocol PDU のフォーマット

Read Multiple Response の Attribute Protocol PDU のフォーマットを図3-90に示します。Attribute Opcode は Read Multiple Response を示す 0x0F が与えられます。Set of Values は、Read Multiple Request によって指定された Attribute Handle のセットそれぞれに対応する Attribute 値が、セットで与えられます。なお、Set of Values のサイズが $(\text{ATT_MTU}-1) \text{ octet}$ を超える場合、Attribute 値のセットの最初の $(\text{ATT_MTU}-1) \text{ octet}$ が返信されてしまうため、注意が必要です。この観点から、Set of Values の値が $(\text{ATT_MTU}-1) \text{ octet}$ である可能性があるなら、Read Multiple Request は使うべきではないといえます。これは Set of Values の受信が完了しているのか、それとも実際は $(\text{ATT_MTU}-1) \text{ octet}$ 以上のサイズがあり、オーバーフローを起こしているのかが Read Multiple Response からはわからないためです。この点からも GATT プロファイルを設計するにあたっては、MTU のサイズ設計が非常に重要な要素であることがわかんと思います。

Read Multiple Response	
Attribute Opcode (1octet)	Set of Values (0~(ATT_MTU-1) octet)

図 3-90 Read Multiple Response の Attribute Protocol PDU のフォーマット

Read by Group Type Request / Read by Group Type Response

冒頭の Read by Type Request では、サーバ上の Attribute の Attribute Type を検索条件として Attribute を検索し、その値を読み込みました。これとは別に、検索条件として GATT プロファイルで定める Attribute のグループ、たとえば Service（詳細は 3-10-1 節の「Service の定義」、146 ページを参照）などの値を読み出すための Attribute Protocol PDU も併せて定義されています。それが Read by Group Type Request と Read by Group Type Response です。

Read by Group Type Request の Attribute Protocol PDU のフォーマットを図 3-91 に示します。Attribute Opcode は Read by Group Type Request を示す 0x10 が与えられます。基本的な PDU のフォーマットは Read by Type Request と同様ですが、Attribute Type を指定する部分が Attribute Group Type に変更されています。Attribute Group Type は先にも述べたように GATT プロファイル上の Service 上の Attribute を指定するのに利用します。サーバは正しく値が読み出した場合、Read Multiple Response を送信します。

Read by Group Type Request			
Attribute Opcode (1octet)	Starting Handle (2octet)	Ending Handle (2octet)	Attribute Group Type (2 or 16octet UUID)

図 3-91 Read by Group Type Request の Attribute Protocol PDU のフォーマット

Read by Group Type Response の Attribute Protocol PDU のフォーマットを図 3-92 に示します。Attribute Opcode は Read by Group Type Response を示す 0x11 が与えられます。Length には続く Attribute Data List 中の Attribute 1 セットあたりのサイズが与えられます。Attribute Data List には Read by Group Type Request で指定した検索条件に合致する Attribute がリストとして提供されます。Attribute Data List のフォーマットを図 3-93 に示します。リスト中の Attribute Handle は発見された Attribute の開始地点のアドレス (Handle) を、End Group Handle は終了地点のアドレス (Handle) を示します。残る Attribute Value は Attribute の値を示します。

Read by Group Type Response		
Attribute Opcode (1octet)	Length (1octet)	Attribute Data List (2 ~ (ATT_MTU-2) octet)

図 3-92 Read by Group Type Response の Attribute Protocol PDU のフォーマット

Attribute Data List (Read by Group Type Response)		
Attribute Handle (2octet)	End Group Handle (2octet)	Attribute Value (Length - 4 octet)

図 3-93 Attribute Data List のフォーマット

なお、Attribute Data Listで読み出すことができるサイズは255octetに制限されるため、実際に読み込むことが可能なAttribute値の最大サイズSize_{Actual Attribute Value}は

$$\text{Size}_{\text{Actual Attribute Value}} = \text{Size}_{\text{max}} - (\text{Attribute Handle} + \text{End Group Handle}) = 255\text{octet} - (2\text{octet} + 2\text{octet}) = 251\text{octet}$$

となり、Size_{Actual Attribute Value} = 251octetとなります。なお、Attribute Protocol PDU全体から見てAttribute Data Listが確保できる仕様上での最大サイズは図3-95からもわかるように〔ATT_MTU - 2〕octet、さらにその中のAttribute Valueは

$$\begin{aligned} (\text{ATT_MTU} - 6) \text{ octet} &= (\text{ATT_MTU} - 2) - (\text{Attribute Handle} + \text{End Group Handle}) \\ &= (\text{ATT_MTU} - 2) - (2 + 2) \end{aligned}$$

となりますが、〔ATT_MTU - 6〕の値が251を超える場合は、想定する〔ATT_MTU - 6〕octetのAttribute値の内の最初の251octetが返却されます。なお、残るAttribute値を読み込む場合は、先に紹介したRead Blob Requestを利用します。

サーバ上のAttributeに書き込む

サーバ上のAttributeに書き出すためのAttribute Protocol PDUとして、ATTでは3種類を定義しています。

Write Request / Write Response

サーバのAttributeに情報を書き込む際「書き込んだ結果、どうなったのか」をレスポンスとして受ける、つまり書き込みに対する確認応答（ACK）をサーバに要求する場合、Attribute Protocol PDUとしてWrite RequestとWrite Responseを利用します。

Write RequestのAttribute Protocol PDUのフォーマットを図3-94に示します。Attribute OpcodeはWrite Requestを示す0x12が与えられます。Attribute Handleは書き込む先のアドレス（Handle）を示し、Attribute ValueはAttributeに書き込む値を示します。Write Requestで書き込むことができる最大サイズは〔ATT_MTU-3〕octet（= ATT_MTU - (Attribute Opcode + Attribute Handle)）となります。サーバ上のAttributeが正しく書き換えられた場合、Write Responseが返信されます。ちなみにデフォルトのMTU（ATT_MTU_{default}）の場合、書き込み

利用できるサイズは20octet ($= \text{ATT_MTU}_{\text{default}} - 3 \text{ octet} = 23 \text{ octet} - 3 \text{ octet}$) となります。

Write Request		
Attribute Opcode (1octet)	Attribute Handle (2octet)	Attribute Value (0~(ATT_MTU-3) octet)

図 3-94 Write Request の Attribute Protocol PDU のフォーマット

Write Response の Attribute Protocol PDU のフォーマットを図3-95に示します。内容は非常にシンプルで、Attribute Opcodeのみとなります。OpcodeはWrite Responseを示す0x13が与えられます。

Write Response
Attribute Opcode (1octet)

図 3-95 Write Response の Attribute Protocol PDU のフォーマット

Write Command

先の Write Request とは異なり、ACKを不要とする書き込み処理を Write Command と呼びます。ACKを利用しないため、基本的には信頼性が求められるAttributeの書き換えに利用します。Write Command の Attribute Protocol PDU のフォーマットを図3-96に示します。Attribute OpcodeはWrite Commandを示す0x52が与えられます。Attribute Handleは書き込む先のアドレス (Handle) を示し、Attribute ValueはAttributeに書き込む値を示します。Attribute Valueに関する諸条件はWrite Requestと同じです。ちなみにデフォルトのMTU ($\text{ATT_MTU}_{\text{default}}$) の場合、書き込みに利用できるサイズは20 octet ($= \text{ATT_MTU}_{\text{default}} - 3 \text{ octet} = 23 \text{ octet} - 3 \text{ octet}$) となります。

Write Command		
Attribute Opcode (1octet)	Attribute Handle (2octet)	Attribute Value (0~(ATT_MTU-3) octet)

図 3-96 Write Command の Attribute Protocol PDU のフォーマット

Signed Write Command

Write Commandとともに認証証明を付与して書き込みを行う処理をSigned Write Commandと呼びます。Signed Write CommandのAttribute Protocol PDUのフォーマットを

図3-97に示します。Attribute OpcodeはSigned Write Commandを示す0xD2が与えられます。Attribute Handleは書き込む先のアドレス(Handle)を示し、Attribute ValueはAttributeに書き込む値を示します。Attribute Valueに関する諸条件はWrite Requestと同じです。続くAuthentication Signatureは書き込みに関する認証に利用する証明用のデータになります。Signed Write CommandではAuthentication Signatureがサーバ側で認証された場合にのみ、書き換えが実行されます。

Signed Write Command			
Attribute Opcode (1octet)	Attribute Handle (2octet)	Attribute Value (0~(ATT_MTU-15) octet)	Authentication Signature (12octet)

図3-97 Signed Write CommandのAttribute Protocol PDUのフォーマット

複数のAttributeを同時にサーバ上に書き込む

サーバ上のAttributeに書き込む際、同時に複数のAttributeに書き込む処理もAttribute Protocol PDUで定義されています。同時に書き込む値をキューとしてサーバ上にスタックする処理をPrepare Write Request/Prepare Write Responseと呼びます。サーバ上にスタックされた値を反映する処理をExecute Write Request/Execute Write Responseと呼びます。

Prepare Write Request / Prepare Write Response

複数のAttributeへの同時書き込みを行うため、サーバ上にキューをスタックするAttribute Protocol PDUをPrepare Write RequestとPrepare Write Responseと呼びます。

Prepare Write RequestのAttribute Protocol PDUのフォーマットを図3-98に示します。Attribute OpcodeはPrepare Write Requestを示す0x52が与えられます。Attribute Handleは書き込む先のアドレス(Handle)を示し、Value Offsetは値を書き込む位置のオフセット量を示します。Part Attribute Valueは書き込むAttribute Valueの一部を示します。Part Attribute Valueで利用できるサイズは $(ATT_MTU-5)\text{octet}$ ($= (ATT_MTU - (\text{Attribute Opcode} + \text{Attribute Handle} + \text{Value Offset})) = (ATT_MTU - (1+2+2))$) となります。Value OffsetとAttribute Valueの関係を図3-99に示します。Prepare Write Requestがサーバ側で正しく処理された場合、サーバからPrepare Write Responseが返却されます。

Prepare Write Request / Prepare Write Response			
Attribute Opcode (1octet)	Attribute Handle (2octet)	Value Offset (2octet)	Part Attribute Value (0~(ATT_MTU-5) octet)

図3-98 Prepare Write RequestのAttribute Protocol PDUのフォーマット

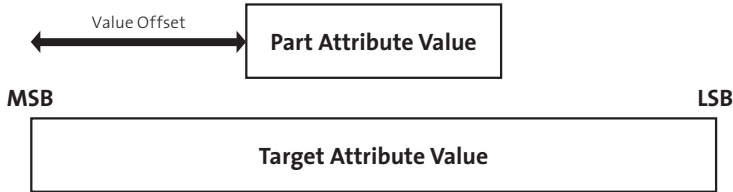


図 3-99 Value Offset と Attribute Value の関係

Prepare Write Response の Attribute Protocol PDU のフォーマットは、Preapre Write Request と全く同じフォーマットとなっています。Attribute Opcode は Prepare Write Response を示す 0x17 が与えられます。Attribute Handle は書き込む先のアドレス (Handle) を示し、Value Offset は値を書き込む位置のオフセット量を示します。Part Attribute Value は書き込まれた Attribute Value の一部を示します。Part Attribute Value に関する諸条件は Prepare Write Request の場合と同じです。

Execute Write Request / Execute Write Response

サーバ上にスタックされた Part Attribute Value を、指定の Attribute に同時書き込みを行う Attribute Protocol PDU を ExecuteWrite Request と ExecuteWrite Response と呼びます。

Execute Write Request の Attribute Protocol PDU のフォーマットを図 3-100 に示します。Attribute Opcode は Execute Write Request を示す 0x18 が与えられます。続く Flags はスタックしたキューを反映するか否かを制御するためのフラグを示します。Flags=0x00 の場合、キューの書き込みをキャンセルします。Flags=0x01 の場合、キューの書き込みを実行します。いずれの Flags であっても Execute Write Request 実行後、キューの内容はクリアされます。キューの実行が成功した場合、サーバーから Execute Write Response が返信されます。

Execute Write Request	
Attribute Opcode (1octet)	Flags (1octet)

図 3-100 Execute Write Request の Attribute Protocol PDU のフォーマット

Execute Write Response の Attribute Protocol PDU のフォーマットを図 3-101 に示します。Attribute Opcode は Execute Write Response を示す 0x19 が与えられます。

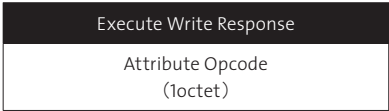


図 3-101 Execute Write Response の Attribute Protocol PDU のフォーマット

サーバからの Notification/Indication

サーバからの Notification および Indication をサポートする Attribute Protocol PDU は、Handle Value Notification および Handle Value Indication/Handle Value Confirmation と呼ばれています。

Handle Value Notification

サーバは Handle Value Notification を利用することで、任意のタイミングで Notification をクライアントに対して実行することができます。Handle Value Notification の Attribute Protocol PDU のフォーマットを図 3-102 に示します。Attribute Opcode は Handle Value Notification を示す 0x1C が与えられます。Attribute Handle は書き込む先のアドレス (Handle) を示し、Attribute Value は Attribute に書き込む値を示します。Attribute Value に関する諸条件は Write Command (3-9-3 項の「Write Command」、140 ページを参照ください) と同じです。ちなみにデフォルトの MTU ($ATT_MTU_{default}$) の場合、書き込みに利用できるサイズは 20octet ($= ATT_MTU_{default} - 3\ octet = 23\ octet - 3\ octet$) となります。

Handle Value Notification		
Attribute Opcode (1octet)	Attribute Handle (2octet)	Attribute Value (0~(ATT_MTU-3) octet)

図 3-102 Handle Value Notification の Attribute Protocol PDU のフォーマット

Handle Value Indication / Handle Value Confirmation

Indication を実行する場合、Notification とは異なり、クライアント側からの ACK が返信されます。この処理を行う Attribute Protocol PDU を Handle Value Indication と Handle Value Confirmation と呼びます。

サーバは Handle Value Indication を利用することで、任意のタイミングで Indication をクライアントに対して実行することができます。Handle Value Indication の Attribute Protocol PDU のフォーマットを図 3-103 に示します。Attribute Opcode は Handle Value Indication を示す 0x1D が与えられます。Attribute Handle は書き込む先のアドレス (Handle) を示し、

Attribute ValueはAttributeに書き込む値を示します。Attribute Valueに関する諸条件はWrite Command（3-9-3項の「Write Command」、140ページを参照ください）と同じです。ちなみに、クライアントがLong AttributeをサーバからIndicationされる場合、Indication後にRead Blob Requestを実行し、残るAttribute Valueの取得を行います。クライアントはHandle Value Indicationを正しく受信すると、Handle Value Confirmationを返信します。

Handle Value Indication		
Attribute Opcode (1octet)	Attribute Handle (2octet)	Attribute Value (0~(ATT_MTU-3) octet)

図 3-103 Handle Value IndicationのAttribute Protocol PDUのフォーマット

Handle Value ConfirmationのAttribute Protocol PDUのフォーマットを図3-104に示します。このPDUはIndicationに対するACKに相当するもので、Attribute Opcodeのみの非常にシンプルな構成となっています。Attribute OpcodeはHandle Value Indicationを示す0x1Eが与えられます。

Handle Value Confirmation
Attribute Opcode (1octet)

図 3-104 Handle Value ConfirmationのAttribute Protocol PDUのフォーマット

3-10. GATTとService

GATTの概略については3-2-1項で述べました。本項ではGATTの詳細とGATTが提供するServiceについての詳細について解説します。GATTについて簡単に振り返ると、GATTはATTを基盤としたプロファイルで、Application側もしくはそれ以外のプロファイルから利用されます。GATTではAttributeを用いた階層的なデータベースを構築することでServiceを定義する他、そのService上でのやり取りをATTによるサーバ/クライアント動作で制御します。

それでは順に、GATT上のServiceについて、その詳細を見ていきましょう。

GATTの対応するイベント

GATTでは、次のユーザイベントに対応しています。これらのイベントをCoreBluetoothフレームワークで制御することで、iOSデバイスからBLEデバイスのServiceにアクセスすることができます。

- 設定の交換 (Exchange)
- BLEデバイス上のServiceとCharacteristicの発見 (Discovery)
- Characteristicの値の読み取り (Read)
- Characteristicの値の書き込み (Write)
- Characteristicの値の通知 (Notification)
- Characteristicの値の表示 (Indication)

3-10-1. Serviceの構造

GATTではAttributeを最小単位とした2種類のデータ構造、CharacteristicおよびServiceが定義されています。図3-9でもすでに述べたとおり、ServiceはCharacteristicやその他のService

への参照子による構成される構造体で、Characteristicは値（Value）、値の属性（Property）、そして値のディスクリプタ（Descriptor）による構造体となります。図3-9のとおり、構造的にはServiceにCharacteristicが、Characteristicに値、属性、ディスクリプタが包含される形となり、この値、属性、ディスクリプタそれぞれがAttribute単位で管理・格納されています。

本項ではServiceやCharacteristicがAttributeによってどのように構成されているのか、具体的に見ていきます。

Serviceの定義

Serviceは3-2-4項でも述べたように、Charactersticによって構成される一種のクラスとして機能しますが、その実体はAttribute単位でメモリ上に展開されているデータベースです。このデータベース上のAttributeの中の記述に従って、ServiceそしてCharacteristicが定義されています。

Serviceは、データベース上でService宣言（Service Declaration）と呼ばれる型のAttributeで宣言されます。Service宣言型のAttributeの記述を図3-108に示します。ここで、Attribute Handleはデータベース上のアドレスを意味します。Attribute TypeはService宣言型であることを示すUUID_{Primary Service}（= 0x2800）、もしくはUUIDSecondary Service（= 0x2801）を持ちます。Attribute Valueは、Serviceを表す16bit Bluetooth UUIDもしくは128bit UUIDのService UUIDを持ちます。Permissionに関しては読み取り専用です。

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	UUIDPrimary Service or UUID Secondary Service	ServiceUUID (16bit Bluetooth UUID or 128bit UUID)	読み取り専用 認証不要 承認不要

図3-105 Service宣言型のAttributeの構造

ここで、Attribute Typeで出てきたUUIDPrimary ServiceとUUIDSecondary Serviceにあるように、Serviceには2種類の属性が存在する点に注意が必要です。Serviceでは、他のServiceをIncludeという機能によってデータベース上で参照し、自身のServiceに組み込む機能がサポートされています。このとき、Primary Serviceは他のServiceをIncludeしたり、また他のサービスからIncludeされたりすることができます。他方、Secondary Serviceは、PrimaryからのIncludeによってのみ利用されるServiceが想定されています。

Includeの定義

上述したように、ServiceではIncludeによって他のServiceを参照することができます。このIncludeはServiceの再利用性やGATTサーバ上のデータベースのメモリの節約に活用することができます。Includeによって参照されたServiceは、参照元のServiceの一部とみなされ、Serviceが定義できるかぎり（GATTサーバ上のデータベース上のメモリの枯渇、もしくはAttribute Handleを0xFFFFまで利用しきらない限り）いくらかでも追加することが可能です。もちろん、1つもService上でIncludeを行わなくても問題はありません。

IncludeはInclude宣言（Include Declaration）と呼ばれる型のAttributeで宣言されます。Include宣言型のAttributeの記述を図3-109に示します。ここで、Attribute Handleはデータベース上のアドレスを意味します。Attribute TypeはInclude宣言型であることを示すUUID_{Include}（= 0x2802）を持ちます。

Include宣言のAttribute Valueは3つの領域で構成されます。Include Service Attribute Handleは、実際にIncludeするServiceのメモリ上の開始地点を示すAttribute Handleを示します。次にEnd Group Handleは、IncludeするServiceの終端のAttribute Handleを示します。ServiceUUIDはサービスを表す16bit Bluetooth UUID / 128bit UUIDを示します。GATTサーバ上で与えられるPermissionは読み取り専用です。

Include宣言型のAttributeを見てもわかるように、Includeは参照先のアドレスであるAttribute Handleによって管理されているので、GATTサーバ上のServiceの配置の順番は任意です。

Attribute Handle	Attribute Type	Attribute Value			Attribute Permission
0xNNNN	UUID _{Include}	Include Service Attribute Handle	End Group Handle	Service UUID	読み取り専用 認証不要 承認不要

図 3-106 Include宣言型のAttributeの構造

なお、Include宣言の参照先のServiceでIncluded元となるServiceがIncludeされる場合を循環参照と呼びます。この循環参照は仕様で禁止されており、GATTサーバにアクセスしたクライアントが循環参照を発見した場合、Attributeのやり取りを停止させます。また、IncludeされるServiceのServiceUUIDが16bitの場合はInclude先のServiceUUIDも16 bit、128bitの場合はInclude先も128bitである必要があります。

3-10-2. Characteristicの定義

CharacteristicはCharacteristic宣言、Characteristic値、Characteristicディスクリプタの3種類の型のAttributeに定義されます。それぞれの型はAttribute単位で区切られます。

では、まずはCharacteristic宣言（Characteristic Declaration）から見ていきましょう。

Characteristic宣言のAttribute

Characteristic宣言のAttributeの記述を図3-107に示します。ここでAttribute Handleはデータベース上のアドレスを意味します。Attribute TypeはCharacteristic宣言型であることを示すUUID_{Characteristic} (= 0x2803) を持ちます。

Characteristic宣言のAttribute Valueは3つの領域で構成されます。Characteristic PropertiesはCharacteristicに付与される、さまざまな属性について定義します。Characteristic Value Attribute Handleは、後述するCharacteristic値を表す型のAttributeへの参照先のAttribute Handle、つまりアドレスを指定します。Characteristic UUIDは参照先のCharacteristic値のUUIDの値が指定されます。

Attribute Handle	Attribute Type	Attribute Value			Attribute Permission
0xNNNN	Characteristic UUID	Characteristic Properties	Characteristic Value Attribute Handle	Characteristic UUID	読み取り専用

図 3-107 Characteristic 宣言型のAttributeの構造

Characteristic Propertiesで定義される属性

Characteristic Propertiesは宣言するCharacteristicにおいて「Characteristic値がどのように利用可能か」「Characteristicディスクリプタがどのようにアクセスされるか」などの属性を定めるフラグです。フラグのサイズは1octetとなっており、各ビットに対して1つずつ属性が割り振られており、合計で8つの属性が定義されています。付与したい属性のビットに1をセットすることで、その属性が有効化されます。各属性の内容は表 3-22 のとおりです。

プロパティ名	値	内容
Broadcast	0x01	セットされた場合、Broadcast で利用される
Read	0x02	セットされた場合、読み取りが可能になる
Write Without Response	0x04	セットされた場合、書き込み可能になる（書き込み先からの応答なし）
Write	0x08	セットされた場合、書き込み可能になる
Notify	0x10	セットされた場合、Notification で利用可能になる
Indicate	0x10	セットされた場合、Indication で利用可能になる
Authenticated Signed Write	0x40	認証署名付きでの書き込みが可能
Extended Properties	0x80	Extended Properties で利用

表 3-22 Characteristic Properties で定義される属性

Characteristic 値の Attribute

Characteristic 値の Attribute の記述を図3-108に示します。ここで、Attribute Handle はデータベース上のアドレスを意味します。Attribute Type は設計者が定義する CharacteristicUUID が指定されます。Attribute Value は実際の Characteristic 値が与えられます。Characteristic 値の Permission については、上位の GATT を利用したプロファイルや実装時のデバイスの仕様に依存します。

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	CharacteristicUUID	Characteristic Value	上位の GATT ベースドのプロファイル もしくは 実装時の仕様に依存

図 3-108 Characteristic 値の Attribute の構造

Characteristic ディスクリプタの Attribute

Characteristic ディスクリプタは、Characteristic 値についての関連情報を記述するために利用される Attribute です。たとえば、Characteristic が食材であると 1-1-1 項でたとえましたが、そのたとえにもとづくと、Characteristic ディスクリプタは「その食材の生産者が誰であるのか」「その食材の価格はいくらなのか」「食材の消費期限はいつなのか」など、食材に添付されている商品情報のシールに相当します。

Characteristic では Characteristic Properties としていくつかの属性を定めていましたが、

Characteristic ディスクリプタは Characteristic に対してというより、その値そのものの属性や情報を追加的に定めるために利用されます。この Characteristic ディスクリプタは、表 3-23 に示すようにさらに 6 種類の Attribute が定義されています。それぞれで役割が異なるので、1 つずつその機能を確認していきましょう。

ディスクリプタ	内容
Characteristic Extended Properties	Characteristic の拡張定義用のディスクリプタ
Characteristic User Description	ユーザーによる情報追記のためのディスクリプタ
Client Characteristic Configuration	ATT クライアントの設定用のディスクリプタ
Server Characteristic Configuration	ATT サーバの設定用のディスクリプタ
Characteristic Presentation Format	Characteristic の値の単位やフォーマットを記述するディスクリプタ
Characteristic Aggregate Format	列挙型の Characteristic に関する設定を行うディスクリプタ

表 3-23 Characteristic ディスクリプタの種類

Characteristic Extended Properties

Characteristic Extended Properties は、Characteristic Properties の拡張定義に利用するディスクリプタです。Characteristic Extended Properties の Attribute の記述を図 3-109 に示します。Attribute Type は Characteristic Extended Properties 型であることを示す `UUIDCharacteristic Extended Properties` (= 0x2900) を持ちます。Attribute Value の特定のビットを制御することで Extended Properties の属性を制御することができます。

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	UUID _{Characteristic Extended Properties}	Characteristic Extended Properties Bit Field	読み取り専用

図 3-109 Characteristic Extended Properties の Attribute の構造

Attribute Value	値	内容
Reliable Write	0x0001	セットすると、後述する 3-10-6 項の Prepare Write Request (176 ページ) の書き込みに対応する
Writable Auxiliaries	0x0002	セットすると Characteristic User Description の書き込みを許可する
Reserved for Future Use (RFU)	0xFFFC	予約領域

表 3-24 Characteristic Extended Properties の Attribute Value の値

Characteristic User Description

Characteristic User Descriptionは、Characteristic値に対してのテキストでの情報記述に利用されるディスクリプタです。Characteristic User DescriptorのAttributeの記述を図3-110に示します。Attribute TypeはCharacteristic User Descriptor型であることを示すUUID_{Characteristic User Descriptor} (= 0x2901)を持ちます。Attribute ValueはUTF-8の文字列を入力することができます。なお、サイズは可変長です。

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	UUID Characteristic User Description	Characteristic User Description UTF-8 String	上位のGATTベースドのプロファイル もしくは実装時の仕様に依存

図 3-110 Characteristic User DescriptorのAttributeの構造

Client Characteristic Configuration

Client Characteristic Configurationは、宣言するCharacteristicにおいてクライアント側へのCharacteristicの振る舞いを設定するフラグです。Client Characteristic PropertiesのAttributeの記述を図3-111に示します。Attribute TypeはClient Characteristic Properties型であることを示すUUID_{Client Characteristic Properties} (= 0x2902)を持ちます。Attribute Valueの特定のビットを制御することでCharacteristicのクライアントに対する振る舞いを制御することができます。なお、この値はボンディング中、ボンディングしたデバイス間で変更されることはありません。

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	UUID Client Characteristic Configuration	Characteristic Configuration Bit	Readable with no authentication or authorization
			Writable with authentication and authorization ^{※7}

図 3-111 Client Characteristic PropertiesのAttributeの構造

Attribute Value	値	内容
Notification	0x0001	Characteristic PropertiesでNotifyビットがセットされている状態でセットされるとCharacteristic値がNotificationされる
Indication	0x0002	Characteristic PropertiesでIndicateビットがセットされている状態でセットされるとCharacteristic値がIndicationされる
Reserved for Future Use (RFU)	0xFFFF	予約領域

表 3-25 Client Characteristic PropertiesのAttribute Valueの値

※7 Authentication・Authorizationについては、上位のGATTベースドのプロファイルもしくは実装時の仕様に依存します。

Server Characteristic Configuration

Server Characteristic Configuration は、宣言する Characteristic においてサーバ側の Characteristic の振る舞いを設定するフラグです。Server Characteristic Properties の Attribute の記述を図3-115 に示します。Attribute Type は Server Characteristic Properties 型であることを示す UUID_{Server Characteristic Properties} (= 0x2903) を持ちます。Attribute Value の特定のビットを制御することで Characteristic のサーバ側の振る舞いを制御することができます。

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	UUID _{Server Characteristic Configuration}	Characteristic Configuration Bit	読み取り可能
			書き込み可能 ^{※8}

図 3-112 Server Characteristic Properties の Attribute の構造

Attribute Value	値	内容
Broadcast	0x0001	Characteristic Properties の Broadcast ビットがセットされている場合にセットすると、サーバが Broadcast を許可する
Reserved for Future Use (RFU)	0xFFFF	予約領域

表 3-26 Server Characteristic Properties の Attribute Value の値

Characteristic Presentation Format

Characteristic Presentation Format は、宣言する Characteristic のデータフォーマットや Characteristic 値の単位を記述するディスクリプタです。Characteristic Presentation Format の Attribute の記述を図3-113 に示します。Attribute Type は Characteristic Presentation Format 型であることを示す UUID_{Characteristic Presentation Format} (= 0x2904) を持ちます。Attribute Value は Format、Exponent、Unit、NameSpace、Description の5つの領域で与えられます。Permission は読み取り専用となります。

Attribute Handle	Attribute Type	Attribute Value					Attribute Permission
0xNNNN	UUID _{Characteristic Presentation Format}	Format	Exponent	Unit	Name Space	Description	読み取り専用

図 3-113 Characteristic Presentation Format の Attribute の構造

Attribute Value のデータ領域については次のとおりです。

※8 Authentication・Authorizationについては、上位の GATT ベースのプロファイルもしくは実装時の仕様に依存します。

• Format

FormatはCharacteristic値がどのようなデータフォーマットかを定義します。Formatの値とデータフォーマットの対応を表3-27に示します。^{※9}なお、Formatで指定されたデータフォーマットがoctet単位（すなわちビット単位）の場合、Characteristic値はそのAttribute ValueでLSBに実際の値が配置されている必要があります。また、残りのビットはゼロにセットされている必要があり、この点は注意が必要です。また、表3-23に示している指数値は後述するExponentに対応するか否かを示しています。

Bluetooth ver.4.x以降でBLE上からIPv6を直接運用する機能が定義されていますが、Characteristic値をIPv4アドレスに利用する場合はuint32型が、IPv6アドレスに利用する場合はuint128型が利用されます。また、Bluetooth BDADDRに利用する場合はuint48型が利用されます。

• Exponent

ExponentはCharacteristic値で一種の固定小数点を扱うために利用します。したがって、Exponentは整数型のFormatでのみ利用することができます。また、Exponentの値自体は必ず整数型となります。

実際のCharacteristic値 = Characteristic値 × 10^{Exponent}

たとえば、Exponent = 2、Characteristic値 = 23の場合、実際の値は2300となります。また、Exponent = -3、Characteristic値 = 3892の場合、実際の値は3.892となります。

• Unit

UnitはCharacteristic値の単位を指定します。UnitはBluetooth SIGによって規格が定められており、Assigned Numbers Documentにその記述を見ることができます。^{※10}

• Namespace

Namespaceは後述するDescriptionの定義（数値と意味を対応付けるデータテーブル）を策定している組織を同定するために利用されます。通常はBluetooth SIGの定義を利用するので、BluetoothSIGを示す値（=0x01）が利用されます。^{※11}

※9 この対応はBluetooth SIGによって定められており、<https://developer.bluetooth.org/gatt/Pages/FormatTypes.aspx>でも確認することができます。

※10 詳しくは、<https://developer.bluetooth.org/gatt/units/Pages/default.aspx>を参照ください。

※11 詳しくは、<https://developer.bluetooth.org/gatt/Pages/GattNamespaceDescriptors.aspx>を参照ください。

Format	短縮名称	内容	指数値
0x00	RFU	RFU (Reserved for Future Use)	No
0x01	boolean	unsigned 1 bit (0 = false, 1 = true)	No
0x02	2bit	unsigned 2 bit integer	No
0x03	nibble	unsigned 4 bit integer	No
0x04	uint8	unsigned 8 bit integer	Yes
0x05	uint12	unsigned 12 bit integer	Yes
0x06	uint16	unsigned 16 bit integer	Yes
0x07	uint24	unsigned 24 bit integer	Yes
0x08	uint32	unsigned 32 bit integer	Yes
0x09	uint48	unsigned 48 bit integer	Yes
0x0A	uint64	unsigned 64 bit integer	Yes
0x0B	uint128	unsigned 128 bit integer	Yes
0x0C	sint8	signed 8 bit integer	Yes
0x0D	sint12	signed 12 bit integer	Yes
0x0E	sint16	signed 16 bit integer	Yes
0x0F	sint24	signed 24 bit integer	Yes
0x10	sint32	signed 32 bit integer	Yes
0x11	sint48	signed 48 bit integer	Yes
0x12	sint64	signed 64 bit integer	Yes
0x13	sin128	signed 128 bit integer	Yes
0x14	float32	IEEE-754 32 bit floating point	No
0x15	float64	IEEE-754 64 bit floating point	No
0x16	SFLOT	IEEE-11073 16 bit floating point	No
0x17	FLOAT	IEEE-11073 32 bit floating point	No
0x18	duint16	IEEE-20601 format	No
0x19	utf8s	UTF-8 string	No
0x1A	utf16s	UTF-16 string	No
0x1B	struct	Opaque structure	No
0x1C ~ 0xFF	RFU	RFU (Reserved for Future Use)	No

表 3-27 Format の値とデータフォーマットの対応

- Description

DescriptionはCharacteristic値を補足する記述を行うために利用します。このDescriptionはNameSpaceに示された組織によって定義されます。^{※12}

Characteristic Aggregate Format

Characteristic Aggregate Formatは宣言するCharacteristic Presentation Formatのオプションとして利用できるディスクリプタです。1つのCharacteristicに対して複数のCharacteristic Presentation Formatを列挙するために利用します。Characteristic Aggregated FormatのAttributeの記述を図3-114に示します。

Attribute TypeはCharacteristic Aggregated Format型であることを示すUUID_{C_{characteristic} Presentation Format} (= 0x2905) を持ちます。Attribute ValueはCharacteristic Presentation FormatのAttribute Handle (16bit) が連結した値をとります。連結されたAttribute Handleの順番は有意な点に注意する必要があります。また、1つのCharacteristicに対して複数のCharacteristic Presentation Formatが定義されている場合、必ずCharacteristic Aggregate Formatの記述がCharacteristic中に含まれます。

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	UUIDCharacteristic Aggregate Format	Characterisistic Presentation FormatのAttribute Handle リスト	読み取り専用

図 3-114 Characteristic Aggregate FormatのAttributeの構造

3-10-3. GATTによるService Changed、Characteristic

さて、これまででServiceとCharacteristicの詳細を確認してきました。ここではGATT自体が提供するServiceである、その名もService Changedと、そのCharacteristicについて解説します。

Service Changedはその名のとおりに、Serviceを変更するためのコントロールポイント（電車の線路の切り替え機のイメージですね）として機能するServiceで、GATTによってあらかじめ提供されています。なお、ここでの「変更」とは、GATTサーバ上のServiceの追加、削除、そして改変です。

※12 詳しくは、<https://developer.bluetooth.org/gatt/Pages/GattNamespaceDescriptors.aspx>を参照ください。

それでは、Service Changedについて見ていきましょう。

GATT Service 宣言

GATT ServiceはGATT Service宣言（Service Declaration）と呼ばれる型のAttributeで宣言されます。GATT Service宣言型のAttributeの記述を図3-115に示します。ここでAttribute Handleはデータベース上のアドレスを、Attribute TypeはService宣言型であることを示すUUID_{GATT} Service (= 0x1801)を持ちます。Attribute Valueは、Serviceを表す16bit Bluetooth UUIDもしくは128bit UUIDのService UUIDを持ちます。Permissionに関しては読み取り専用です。

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	UUID _{GATT} Service	ServiceUUID (16bit Bluetooth UUID or 128bit UUID)	読み取り専用 認証不要 承認不要

図 3-115 GATT Service 宣言のAttributeの構造

Service Changed Characteristic 宣言

Service Changed Characteristic宣言のAttributeの記述を図3-116に示します。基本的には、Characteristic宣言と同様のAttributeとなります。Attribute Handleはデータベース上のアドレスを、Attribute TypeはCharacteristic宣言型と同様のUUIDCharacteristic (= 0x2803)を持ちます。

Attribute ValueもCharacteristic宣言と同様に3つの領域で構成されますが、その値がService Changedでは固定になっています。Characteristic Propertiesはベースの値(=0x20)が指定されます。他の属性をPropertiesに与える場合は0x20をベースとして他のビットを立てていきます。Characteristic Value Attribute Handleは、後述するService Changed Characteristic値を表す型のAttributeへの参照先のAttribute Handleを指定します。Characteristic UUIDはServiceChanged専用のUUIDServiceChangedの値(=0x2A05)が指定されます。

Attribute Handle	Attribute Type	Attribute Value			Attribute Permission
0xNNNN	UUIDCharacteristic	Characteristic Properties	Characteristic Value Attribute Handle	UUIDServiceChanged	読み取り専用

図 3-116 Service Chancged Characteristic 宣言の Attribute の構造

Service Changed Characteristic 値

Service Chancged Characteristic 宣言の Attribute の記述を図3-117に示します。Attribute Handleはデータベース上のアドレスを示し、先のService Change Characteristic 宣言の Attribute Value内のCharacteristic Attribute Handleで指定されるAttribute Handleになります。Attribute TypeはUUIDServiceChanged (= 0x2A05) を持ちます。Attribute Valueは2つの領域に分けられており、Service Changedによって影響を与えるGATTサーバ上のAttributeHandleの範囲の開始地点から終了地点までが与えられています。

Attribute Handle	Attribute Type	Attribute Value		Attribute Permission
0xNNNN	UUIDCharacteristic	Start of Affected Attribute Handle Range	End of Affected Attribute Handle Range	読み取り不可 書き込み不可

図 3-117 Service Changed Characteristic 値の Attribute の構造

3-10-4. GAPによるService、Characteristic

GATTによるServiceと同様に、GATTプロファイルに包含されるGAP自体が提供するServiceが存在します。ここではGAPによるService、Characteristicについて解説します。なお、GAP ServiceはGATTサーバ上で常に1つのインスタンスを持ち、GAP ServiceはBLEのCentral、Peripheralではサポートが必須です。

GAP Service 宣言

GAP ServiceはGAP Service宣言（GAP Service Declaration）と呼ばれる型のAttributeで宣言されます。GAP Service宣言型のAttributeの記述を図3-118に示します。ここでAttribute Handleはデータベース上のアドレスを、Attribute TypeはService宣言型であることを示すUUID_{GAP Service}（= 0x1800）を持ちます。Attribute Valueは、Serviceを表す16bit Bluetooth UUIDもしくは128bit UUIDのService UUIDを持ちます。Permissionに関しては読み取り専用です。

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	UUID _{GAP Service}	ServiceUUID (16bit Bluetooth UUID or 128bit UUID)	読み取り専用 認証不要 承認不要

図3-118 GAP Service 宣言のAttributeの構造

Device Name Characteristic 値

Device Name Characteristicはデバイス名を記述するCharacteristicです。Device Name Characteristic値のAttributeの記述を図3-119に示します。Attribute Handleはデータベース上のアドレスを示します。Attribute TypeはUUID_{Device Name}（= 0x2A00）を持ちます。Attribute ValueはDevice Nameを0～248 octetまで記述することができます。

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	UUID _{Device Name}	Device Name	読み取り可 オプションで書き込みも可能

図3-119 Device Name Characteristic 値のAttributeの構造

Appearance Characteristic 値

Appearance Characteristicはデバイスの外観、つまり外部から見えるデバイスのカテゴリなどを記述するCharacteristicです。Appearance Characteristic値のAttributeの記述を図3-120に示します。Attribute Handleはデータベース上のアドレスを示します。Attribute TypeはUUID_{Appearance Characteristic}（= 0x2A01）を持ちます。Attribute Valueはデバイスのカテゴリ（10bit）

とサブカテゴリ（6bit）で構成される Appearance（合計2 octet）が記述されます。このカテゴリとサブカテゴリを表す Appearance の定数は参考文献を参照ください。^{※13}

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	UUIDAppearance Characteristic	Appearance	読み取り専用

図 3-120 Appearance Characteristic 値の Attribute の構造

Peripheral Privacy Flag 値

Peripheral 側のデバイスのプライバシー機能を制御する Characteristic です。Peripheral Privacy Flag 値の Attribute の記述を図 3-121 に示します。Attribute Handle はデータベース上のアドレスを示します。Attribute Type は UUID_{Peripheral Privacy Flag} (= 0x2A02) を持ちます。Attribute Value の値が 0x00 の場合、デバイス内のプライバシー機能を無効化します。逆に値が 0x01 場合、プライバシー機能を有効化します。

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	UUIDPeripheral Privacy Flag	Peripheral Privacy Flag	読み取り可 オプションで書き込みも可能

図 3-121 Peripheral Privacy Flag 値の Attribute の構造

Reconnection Address 値

Reconnection Address 値は Reconnection Address を記述する Characteristic です。Attribute の記述を図 3-122 に示します。Attribute Handle はデータベース上のアドレスを示します。Attribute Type は UUID_{Reconnection Address} (= 0x2A03) を持ちます。Attribute Value は GAP によって生成される non-resolvable address が記述されます。non-resolvable address は uint48 型を取ります。non-resolvable address については ver4.0 Vol.3 Section 10.8.2.1 を参照ください。

※13 <https://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicViewer.aspx?u=org.bluetooth.characteristic.gap.appearance.xml>

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	UUIDReconnection Address	Reconnection Address	書き込み専用

図 3-122 Reconnection Address 値の Attribute の構造

Peripheral Preferred Connection Parameters Characteristic (PPCP) 値

PPCPはPeripheralの推奨する接続設定を記述するCharacteristicです。PPCP値のAttributeの記述を図3-123に示します。Attribute Handleはデータベース上のアドレスを示します。Attribute TypeはUUID_{PPCP} (= 0x2A04) を持ちます。

Attribute Valueは4つの接続設定に関する領域で構成されます。

Attribute Handle	Attribute Type	Attribute Value				Attribute Permission
0xNNNN	UUID _{PPCP}	Minimum Connection Interval	Maximum Connection Interval	Slave Latency	Connection Supervision Timeout Multiplier	読み取り可

図 3-123 PPCP 値の Attribute の構造

Minimum Connection Interval

Minimum Connection IntervalはConnection Intervalの最小値Conn_Interval_Minを定義します。Connection Intervalの最小値の実値connIntervalminは次の式に従います。Conn_Interval_Minの値域は0x0006～0x0c80となっており、値域外の値はサポートされません。

$$\text{connIntervalmin} = \text{Conn_Interval_Min} \times 1.25\text{ms}$$

Maximum Connection Interval

Maximum Connection IntervalはConnection Intervalの最大値Conn_Interval_Maxを定義します。Connection Intervalの最大値の実値 connIntervalmaxは次の式に従います。Conn_Interval_Maxの値域は0x0006～0x0c80となっており、値域外の値はサポートされません。また、Conn_Interval_Maxの値は必ずConn_Interval_Min以上の値をとります。

$$\text{connIntervalmax} = \text{Conn_Interval_Max} \times 1.25\text{ms}$$

Slave Latency

Connection イベント数で示される Slave Latency を定義します。Slave Latency の値域は 0x0000 ～ 0x01F3 となっており、値域外の値はサポートされません。

Connection Supervision Timeout Multiplier

接続監視におけるタイムアウトの値 N を定義します。このタイムアウト Time の実値は 10ms の整数倍で計算され、以下の式で表されます。

$$\text{Time} = N \times 10\text{ms}$$

N の値域は 0x000A ～ 0x0c80 のため、Time の値域は 100ms ～ 32s の範囲になります。なお、地域外の値はサポートされません。

3-10-5. GATT プロファイルの Attribute Type の一覧

以上が GATT プロファイル中の Service、Characteristic の定義になります。本節で取り上げた Service、Characteristic に関する Attribute Type の一覧を表 3-28 に示します。

1

2

3

4

5

6

7

8

9

10

11

12

Attribute Type	UUID
<<GAP Service>>	0x1800
<<GATT Service>>	0x1801
<<Primary Service>>	0x2800
<<Secondary Service>>	0x2801
<<Include>>	0x2802
<<Characteristic>>	0x2803
<<Characteristic Extended Properties>>	0x2900
<<Characteristic User Description>>	0x2901
<<Client Characteristic Configuration>>	0x2902
<<Server Characteristic Configuration>>	0x2903
<<Characteristic Format>>	0x2904
<<Characteristic Aggregate Format>>	0x2905
<<Device Name >>	0x2A00
<<Appearance>>	0x2A01
<<Peripheral Privacy Flag>>	0x2A02
<<Reconnection Address>>	0x2A03
<<PPCP>>	0x2A04
<<Service Changed>>	0x2A05

表 3-28 GATT プロファイルの Attribute Type の一覧

3-10-6. GATT プロファイルで利用できる機能

GATT プロファイルは、本節の冒頭より述べていた Attribute をベースとしたデータベースによる Service、Characteristic と、本項で解説するそれらを利用するサーバ/クライアントの機能の2つによって構成されています。これは本章冒頭での市場のたとえで考えると、前者の Attribute ベースのデータベースは「市場の食材の陳列ブース」であり、後者の Service、Characteristic を利用するサーバ/クライアント機能は「ブースで食材を売買する店員である」と考えることができます。

本章はここまでで前者に相当する GATT によって提供される Service、Characteristic について解説しました。本項では、後者のそれらをベースとした GATT プロファイルによって提供される機能について解説していきたいと思います。

GATT プロファイルで利用できる機能と、GATT クライアント/サーバでの対応/非対応を表 3-29 に示します。表からもわかるように、GATT プロファイルの機能は 11 の Procedure と 21 の Sub-Procedure が存在します。ここで、Sub-Procedure とは Procedure で定義される一連の手順を利用した副次的な Procedure を指します。

No.	GATT プロファイルの機能		対応 / 非対応	
	Procedure	Sub-Procedure	クライアント	サーバ
1	Server Configuration	Exchange MTU		Option
2	Primary Service Discovery	Discover All Primary Services		○
		Discover Primary Services by Service UUID		○
3	Relationship Discovery	Find Included Services		○
4	Characteristic Discovery	Discover All Characteristic of a Service		○
		Discover Characteristic by UUID		○
5	Characteristic Descriptor Discovery	Discover All Characteristic Descriptors		○
6	Reading a Characteristic Value	Read Characteristic Value	Option	○
		Read Using Characteristic UUID		○
		Read Long Characteristic Values		
		Read Multiple Characteristic Values		
7	Writing a Characteristic Value	Write Without Response		
		Signed Write Without Response		
		Write Characteristic Value		
		Write Long Characteristic Values		
8	Notification of a Characteristic Value	Notifications		Option
9	Indication of a Characteristic Value	Indications	○	
10	Reading a Characteristic Descriptor	Read Characteristic Descriptors		
		Read Long Characteristic Descriptors		
11	Writing a Characteristic Descriptor	Write Characteristic Descriptors	Option	
		Write Long Characteristic Descriptors		

表 3-29 GATT プロファイルで利用できる機能と GATT クライアント / サーバの対応 / 非対応

それでは、さっそく Server Configuration から GATT プロファイルの機能を見ていきましょう。

Server Configuration

このProcedureはGATTクライアントによってATTの設定を行うためのProcedureになります。このProcedureでは、MTU容量を変更するsub-ProcedureであるExchange MTUがサポートされています。

MTUの容量を変更する（Exchange MTU）

MTU (ATT_MTU) の容量を変更するSub-ProcedureをExchange MTUと呼びます。GATTクライアントがデフォルト値よりも大きな容量のATT_MTUを要求する際に利用します。変更できる容量は接続された双方のデバイスがサポートする最大容量まで変更することができます。このSub-Procedureは、接続中に一度だけ実行することができます。サーバ/クライアント間の通信の概略を図3-124に示します。ここで利用しているAttribute Protocol PDUのExchange MTU Request、Exchange MTU Responseについては3-9-3項を参照ください。

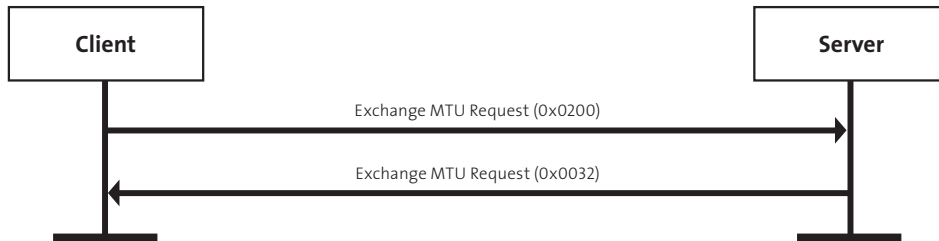


図 3-124 Exchange MTU

Primary Service Discovery

GATTサーバ上のPrimary Serviceを発見するProcedureをPrimary Service Discoveryと呼びます。このProcedureによってGATTサーバ上にPrimary Serviceを発見することで、それにひも付いたIncludeやCharacteristicなどの情報に、他のProcedureからアクセスすることもできます（このProcedureが次に示すRelationship Discoveryとなります）。このProcedureでは、GATTサーバ上のすべてのPrimary Serviceを発見するDiscover All Primary Servicesと、ServiceUUIDからPrimary Serviceを発見するDiscover Primary Services by ServiceUUIDがサポートされています。以降では、この2つのsub-Procedureについて見ていきましょう。

すべてのサービスを発見する (Discover All Primary Service)

GATTサーバ上のサービスを発見するSub-ProcedureをDiscover All Primary Serviceと呼びます。GATTサーバ上のAttributeを、Attribute Handleが0x0001から0xFFFFまで、Attribute Typeが<<Primary Service>> (=0x2800)であるサービスを探索します。クライアントはGroup Type Requestを利用することで、サーバからGroup Type Responseが返却されます。サーバ/クライアント間の通信の概略を図3-125に示します。ここで利用しているAttribute Protocol PDUについては3-9-3節の「Read By Group Type Request/Response」、138ページを参照ください。

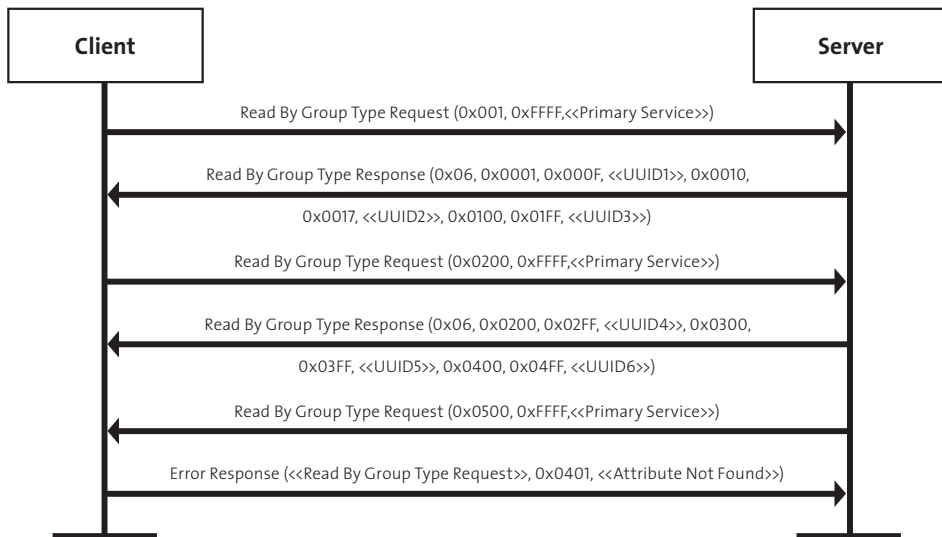


図 3-125 Discover All Primary Service

ServiceUUIDからサービスを発見する (Discover Primary Services by ServiceUUID)

先の方法とは別に、ServiceUUIDから直接GATT上のサービスを発見するSub-Procedureも存在します。これをDiscover Primary Services by ServiceUUIDと呼びます。クライアントはFind By Type Value Requestを利用することで、サーバからFind By Type Value Responseが返却されます。サーバ/クライアント間の通信の概略を図3-126に示します。ここで利用しているAttribute Protocol PDUについては3-9-3節の「Find By Type Value Request/Response」、131ページを参照ください。なお、Find By Type Value Requestで指定しているAttribute Handleの終了条件よりも早く、与えたServiceUUIDが発見された場合、Serviceの探索は終了します。

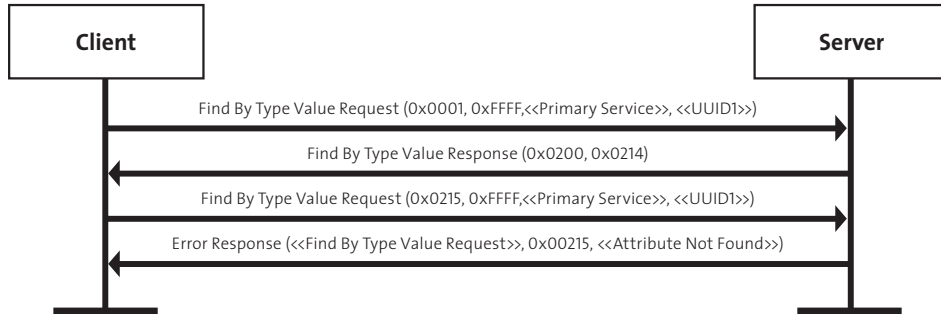


図 3-126 Discover Primary Services by ServiceUUID

Relationship Discovery

GATTサーバ上のServiceは、Service内で他のServiceをIncludeすることでServiceの再利用性を高めることができます。このIncludeされている他のServiceなどを検索し、Include元のServiceとの関係性を調べるProcedureをRelationship Discoveryと呼び、Sub-ProcedureとしてFind Included Servicesをサポートします。

Service上のIncludeを発見する (Find Included Services)

Service内のIncludeの発見するSub-ProcedureをFind Included Servicesと呼びます。サーバ/クライアント間の通信の概略を図3-127に示します。クライアントはRead By Type Requestを利用することで、サーバからRead By Type Responseが返却されます。ここで利用しているAttribute Protocol PDUについては3-9-3節の「Read By Type Request/Response」(133ページ)、「Read Request/Response」(135ページ)項を参照ください。

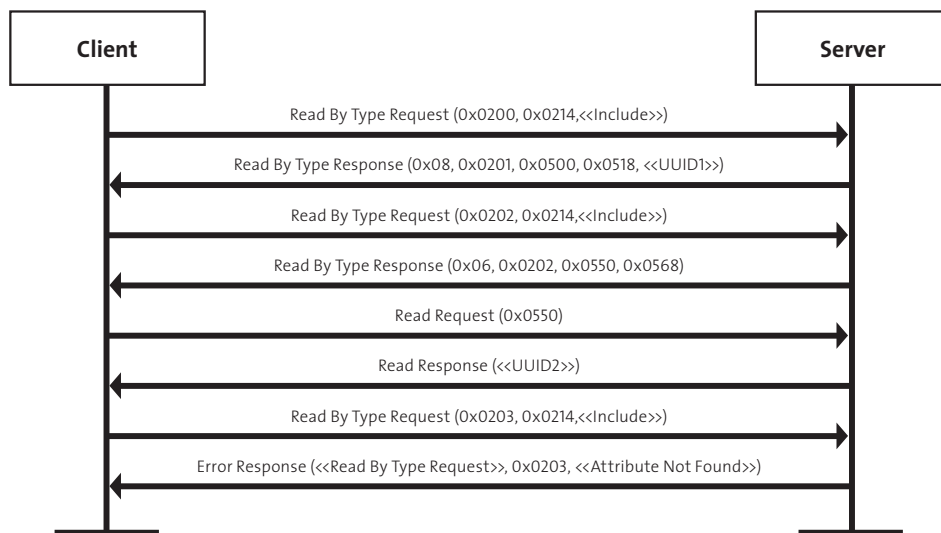


図 3-127 Find Included Services

Characteristic Discovery

Serviceの発見と同様、Characteristicの発見もサポートされています。Characteristicを発見することにより、他のProcedureからそれにひも付いた情報にアクセスすることができます。このProcedureでは、発見したService上のすべてのCharacteristicを発見するDiscover All Characteristics of a Serviceと、UUIDから直接Characteristicを発見するDiscover Characteristic by UUIDがSub-Procedureとしてサポートされています。

Service上のCharacteristicをすべて発見する (Discover All Characteristics of a Service)

先のProcedureでServiceの発見を行いました。この発見したService上のCharacteristicをすべて発見するSub-ProcedureをDiscover All Characteristics of a Serviceと呼びます。サーバ/クライアント間の通信の概略を図3-128に示します。クライアントがRead By Type RequestでUUIDとして<<Characteristic>>を指定して利用することで、サーバからRead By Type Responseが返却されます。ここで利用しているAttribute Protocol PDUについては3-9-3節の「Read By Type Request/Response」、133ページを参照ください。

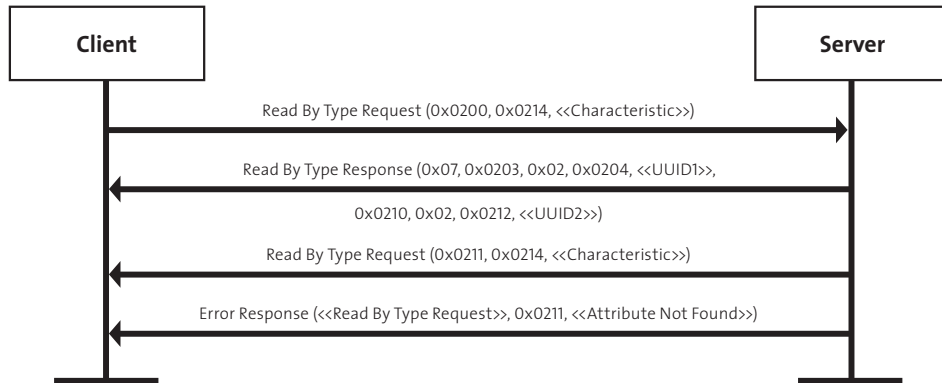


図 3-128 Discover All Characteristics of a Service

UUID から Characteristic を発見する (Discover Characteristics by UUID)

Service の Attribute Handle の範囲が既知、かつ Characteristic UUID が既知の場合に利用する Sub-Procedure を Discover All Characteristic of a Service と呼びます。手続きや利用する Attribute Protocol PDU については、基本的に Discover All Characteristic の場合と同様です。サーバ/クライアント間の通信の概略を図 3-129 に示します。

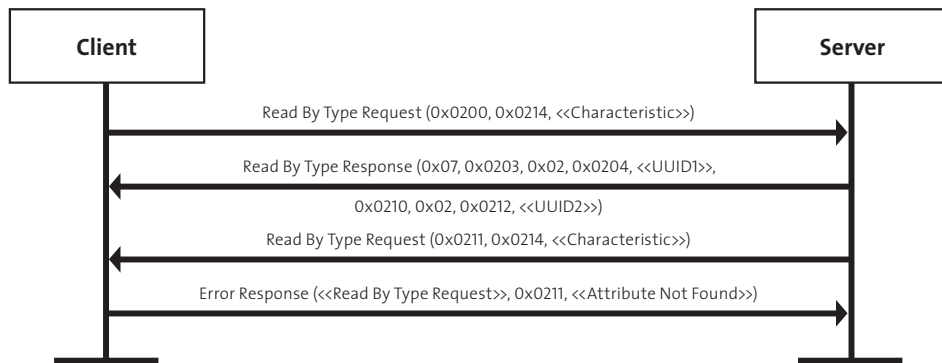


図 3-129 Discover Characteristics by UUID

Characteristic Descriptor Discovery

当然、発見した Characteristic からその Characteristic ディスクリプタを発見することもできます。この Procedure を Characteristic Descriptor Discovery と呼んでいます。これによって

Characteristic ディスクリプタを発見した後、後述する他の Procedure を使ってディスクリプタ内の情報を引き出すことも可能です。この Procedure では、ディスクリプタの発見する Sub-Procedure、Characteristic Descriptor Discovery がサポートされています。

Characteristic ディスクリプタを発見する (Discover All Characteristic Descriptors)

ある Characteristic の Attribute Handle が既知であるとして、その範囲内の Characteristic ディスクリプタの Attribute Handle と Attribute Type を発見する Sub-Procedure は、Discover All Characteristic Descriptors と呼ばれています。サーバ/クライアント間の通信の概略を図 3-130 に示します。クライアントが Find Information Request で調べたい Characteristic の Attribute Handle の範囲を指定することで、サーバから Find Information Response が返却されます。ここで利用している Attribute Protocol PDU については 3-9-3 節の「Find Information Request/Response」、129 ページを参照ください。

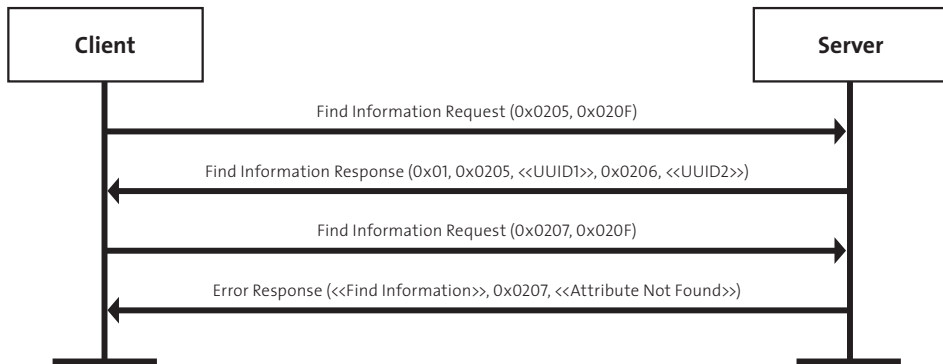


図 3-130 Discover All Characteristic Descriptors

Characteristic Value Read

GATT サーバから Characteristic 値を読み出す Procedure は Characteristic Value Read と呼ばれています。この Procedure には、純粹に Characteristic 値を読み出す Read Characteristic Value、Characteristic UUID から値を読み出す Read Using Characteristic UUID、Read Characteristic Value では読み出しきれない長さの Characteristic 値を読み出す Read Long Characteristic Value、そして複数の Attribute Handle のセットから Characteristic 値のセットを読み出す Read Multiple Characteristic Values、以上の 4 種類の Sub-Procedure が定義されています。

以降で、それら Sub-Procedure について詳細を見ていきましょう。

Characteristic 値を読み出す (Read Characteristic Value)

GATT サーバ上に存在する Characteristic 値の Attribute Handle が既知、かつ Characteristic 値の容量が $[ATT_MTU - 1]$ である場合、Characteristic 値を直接読み出すことができます。この処理の手順を示している Sub-Procedure が Read Characteristic Value です。 $[ATT_MTU - 1]$ 以上の長さを持つ Characteristic 値を読み出す場合は、後述する Characteristic Long Characteristic Value を利用します。サーバ/クライアント間の通信の概略を図 3-131 に示します。この Sub-Procedure では、クライアントが Read Request で読み出したい Characteristic の Attribute Handle を指定することで、サーバから Read Response として Characteristic 値が返却されます。ここで利用している Attribute Protocol PDU については 3-9-3 節の「Read Request/Response」、135 ページを参照ください。

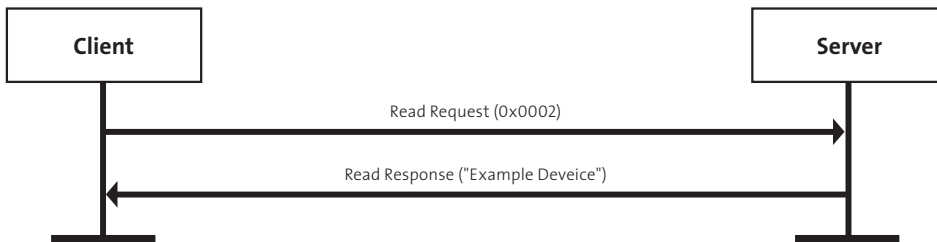


図 3-131 Read Characteristic Value

Characteristic UUID から Characteristic 値を読み出す (Read Using Characteristic UUID)

GATT サーバ上の Characteristic UUID が既知である場合、Characteristic UUID から Characteristic 値を直接読み出すことができます。この処理の手順を示したのが Read Using Characteristic UUID です。サーバ/クライアント間の通信の概略を図 3-132 に示します。この Sub-Procedure では、クライアントが Read By Type Request で GATT サーバ上のメモリ空間 (Attribute Handle: 0x0001 ~ 0xFFFF) で所望の Characteristic UUID を直接検索します。検索の結果はサーバから Read By Type Response として Characteristic 値が返却されます。ここで利用している Attribute Protocol PDU については 3-9-3 節の「Read By Type Request/Response」、133 ページを参照ください。

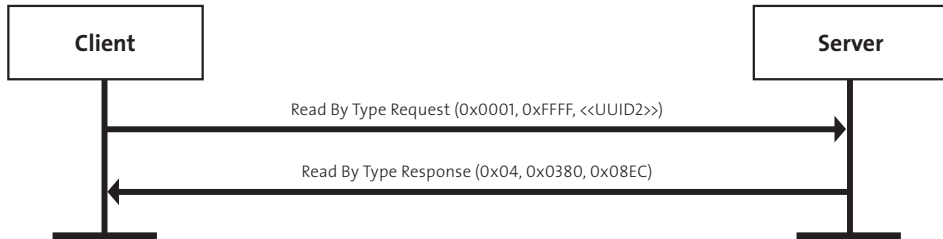


図 3-132 Read Using Characteristic UUID

〔ATT_MTU-1〕以上の長さのCharacteristic値を読み出す(Read Long Characteristic Values)

Characteristic値のAttribute Handleと値の長さが既知である場合、通常はRead Characteristic Valueを利用しますが、値の長さが〔ATT_MTU-1〕を超える場合、一度のRead Characteristic Valueではプロトコル上、値の読出しに対処できません。このような〔ATT_MTU-1〕を超えた値の読み出しを行う場合に利用するSub-ProcedureがRead Long Characteristicです。サーバ/クライアント間の通信の概略を図3-133に示します。このSub-Procedureでは、クライアントはRead RequestでGATTサーバ上へ読み出すのに加え、Read Blob Requestを利用してさらに値を読み出します。Read Blob Requestに与える引数Offset(読み出し対象のCharacteristic値のLSBからのオフセット)は当然、〔ATT_MTU-1〕となります。Read Blob RequestでOffsetを〔ATT_MTU-1〕× n (n:繰り返し回数)として連続で読み出すことで、所望の長さのCharacteristic値を読み出します。ここで利用しているAttribute Protocol PDUについては3-9-3節の「Read Request/Response」、および「Read Brob Request/Response」、135ページを参照ください。

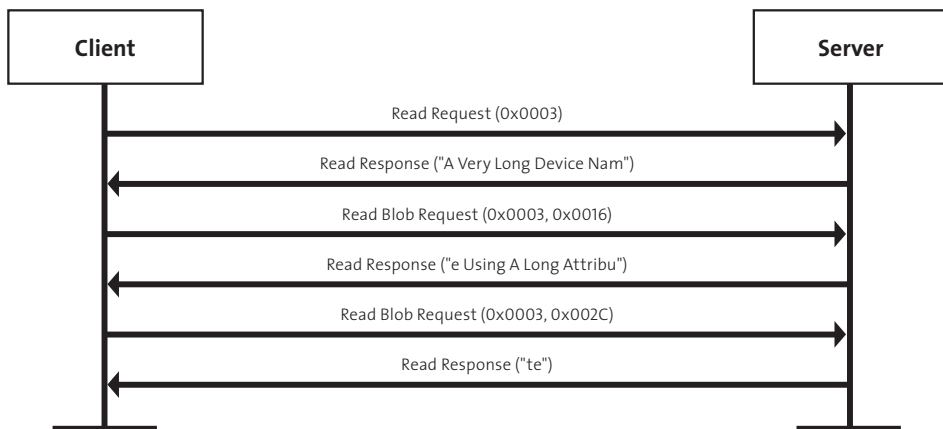


図 3-133 Read Long Characteristic Values

Attribute Handle のセットから Characteristic 値のセットを読み出す (Read Multiple Characteristic Values)

Characteristic 値の Attribute Handle のセットが既知である場合、これら Attribute Handle に対応する Characteristic 値をセットで確認することができます。このような Sub Procedure を Read Multiple Charactersistic と呼びます。サーバ/クライアント間の通信の概略を図 3-134 に示します。この Sub-Procedure では、クライアントは Read Multiple Response を用いて GATT サーバ上から値を読み出します。セットとなっている Characteristic 値の長さは〔ATT_MTU-1〕以下となり、〔ATT_MTU-1〕以上の長さを持つ Characteristic 値を読み出した場合、最初の〔ATT_MTU-1〕の長さ分だけの値が読み出されます。ここで利用している Attribute Protocol PDU については 3-9-3 節の「Read Multiple Request/Response」、137 ページを参照ください。

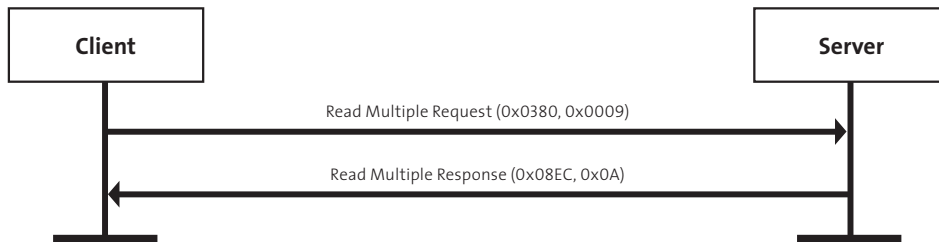


図 3-134 Read Multiple Characteristic Values

Characteristic Value Write

読み込みと対になる Procedure として、GATT サーバ上の Characteristic に値を書き込む Procedure が定義されており、Characteristic Value Write と呼ばれています。この Procedure には、確認応答 (Acknowledgement、ACK などと呼びます) を必要としない書き込みを行う Write Without Response、それに加えて暗号化を利用して書き込みを行う Signed Write Without Response、書き込みに対して確認応答を受け取る Write Characteristic Value、Write Characteristic Value で扱う以上の長さの Characteristic 値を書き込むための Write Long Characteristic Value、そして複数の Characteristic に同時に Characteristic 値を反映する Reliable Write、以上の 5 種類の Sub-Procedure がサポートされています。

Characteristic 値を書き込む (ACK なし) (Write Without Response)

無線通信において、通信相手に関する情報は通信によってしか知ることはできません。したがって、特に無線通信で相手側のデバイスに情報を書き込む際、その操作によって通信相

手から「結果、どうなったのか」を知ることは非常に重要となります。この確認応答をACK (Acknowledgement)と呼びます。Characteristic値をGATTサーバに書き込む際、このACKを受けるか否かで利用するSub-Procedureが異なります。表3-30に、Characteristic Value WriteのSub-Procedureの違いを示します。

Sub-Procedure名	ACKの有無
Write Without Response	×
Signed Write Without Response	×
Write Characteristic Value	○
Write Long Characteristic Value	○
Reliable Write	○

表 3-30 Characteristic Value WriteのSub-Procedureの違い

表3-30からわかるように、Write Without ResponseはACKを利用せずに書き込みを行います。サーバ/クライアント間の通信の概略を図3-135に示します。ACKを利用せずに書き込むため、非常にシンプルにクライアントからWrite Commandを指定のAttribute Handleに対して行うのみになります。Write Commandについては3-9-3節の「Write Command」、140ページを参照ください。

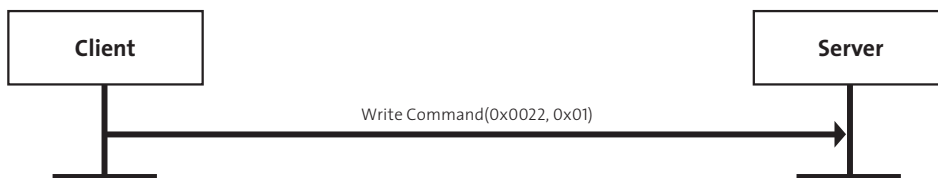


図 3-135 Write Without Response

暗号化を有効化してCharacteristic値を書き込む(ACKなし)(Signed Write Without Response)

Signed Write Without Responseは暗号化を有効化してCharacteristicを書き込むSub-Procedureです。Write Without Responseと同様、ACKは利用しません。このSub-Procedureは

- ① Characteristic PropertiesにおいてAuthenticated Signed Writeが有効化されている（参照3-10-2項の「Characteristic Propertiesで定義される属性」、148ページ）
- ② クライアントとサーバがBondingされている（参照3-8-4項）

場合にのみ利用することができます。

またこの Sub-Procedure は (ATT_MTU-15) octet までの書き込みに対応し、一度の操作でのそれ以上の書き込みには対応していません。

サーバ/クライアント間の通信の概略を図3-136に示します。ACKを利用しないため、Write Without Responseと同様、サーバからのレスポンスはありません。書き込みについては Signed Write Command を利用します。引数である Authentication Signature には、SMP によって発行された認証署名が与えられます。ちなみに Characteristic 値の書き込み幅が (ATT_MTU-15) octet に制限されている理由は、Signed Write Command の引数において Attribute Opcode で 1octet、Attribute Handle で 2octet、Authentication Signature で 12octet が必要なためです (ATT - MTU - (1-2-12) = ATT_MTU - 15)。Signed Write Command については 3-9-3 項の「Signed Write Command」、140 ページを参照ください。



図 3-136 Signed Write Without Response

Characteristic 値を書き込む (ACK あり) (Write Characteristic Value)

Write Without Response とは異なり、書き込みに GATT サーバ側からの ACK を要求する Sub-Procedure を Write Characteristic Value と呼びます。ここで、一度に書き込むことができる Characteristic 値は (ATT_MTU-3) となります。(ATT_MTU-3) であるのは、引数の Attribute opcode と Attribute Handle で 3octet 消費してしまうためです。これ以上の書き込みが必要な場合は、後述する Write Long Characteristic Value を利用します。

サーバ/クライアント間の通信の概略を図3-137に示します。書き込みには Write Request を利用します。書き込みが成功していた場合、サーバ側から Write Response が ACK として返却されます。ここで利用している Attribute Protocol PDU については 3-9-3 節の「Write Request/Response」(139 ページ) を参照ください。

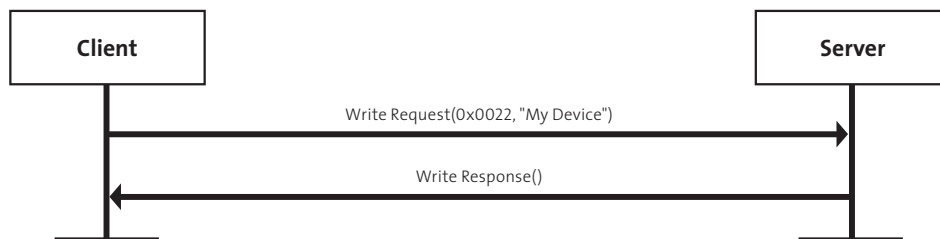


図 3-137 Write Characteristic Value

〔ATT_MTU-3〕octet 以上の値を書き込む（ACK あり）（Write Long Characteristic Value）

〔ATT_MTU-3〕octet 以上の Characteristic 値を GATT サーバ上に書き込む場合の Sub-Procedure を Write Long Characteristic Value と呼びます。

サーバ/クライアント間の通信の概略を図 3-138 に示します。この Sub-Procedure で、クライアント側は Prepare Write Request と Execute Write Request の 2 種類の Attribute Protocol PDU を利用して書き込みを行います。Prepare Write Request を利用して書き込む内容をサーバ上にキューとしてためることができ、目的のキューがたまったタイミングで Execute Write Request でサーバ上の Characteristic 値に反映させることができます。なお、Prepare Write Request で値を書き込む場合、Attribute Opcode で 1octet、Attribute Handle で 2octet、そして書き込む値のオフセット量を指定する Value Offset で 2octet 消費してしまうため、一度に書き込むことができる Characteristic 値は〔ATT_MTU-5〕octet となります。ここで利用している Attribute Protocol PDU については 3-9-3 節の「Prepare Write Request/Response」（141 ページ）および「Execute Write Request/Response」（142 ページ）を参照してください。

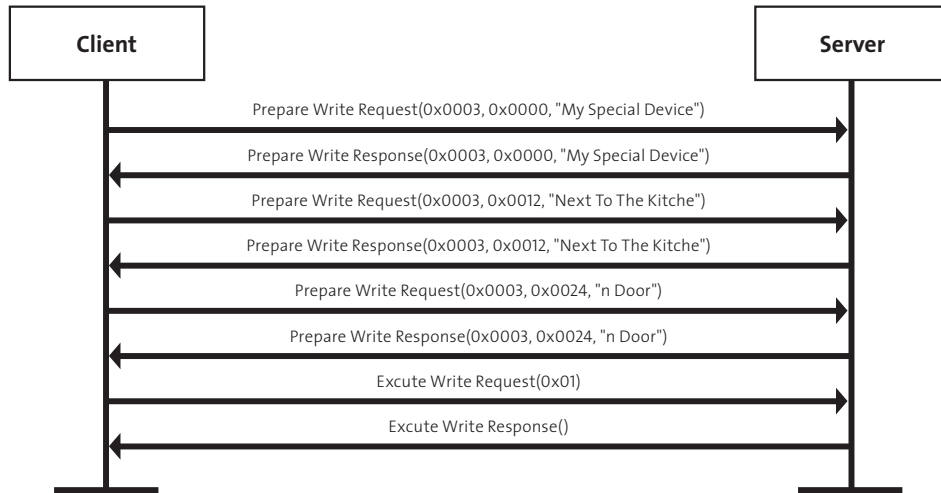


図 3-138 Write Long Characteristic Value

複数の Characteristic に同時に Characteristic 値を書き込む (Reliable Writes)

先ほどの Prepare Write Request/Excecute Write Request を活用することで、複数の Characteristic に同時に Characteristic 値を書き込むこと Sub-Procedure もサポートされており、これを Reliable Writes と呼びます。

サーバ/クライアント間の通信の概略を図 3-139 に示します。この Sub-Procedure では、Write Long Characteristic Value と同様、Prepare Write Request/Excecute Write Request を利用します。Write Long Characteristic Value との違いは Value Offset は 0x00 として、書き込みたい Characteristic への Attribute Handle を逐次指定しつつ、反映したい Characteristic 値を送信します。値の反映は Write Long Characteristic Value と同様、Excecute Write Request で同時に反映させることができます。ここで利用している Attribute Protocol PDU については 3-9-3 節の「Prepare Write Request/Response」(141 ページ) および「Excecute Write Request/Response」(142 ページ) を参照ください。

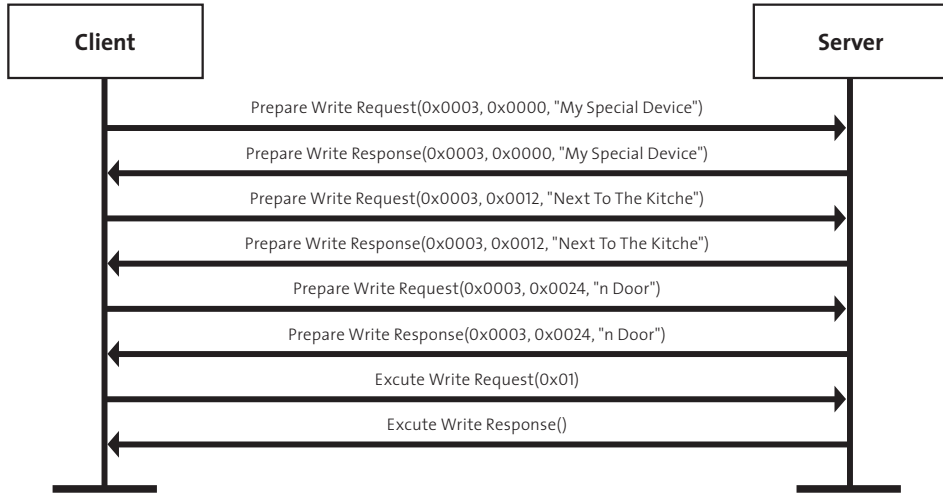


図 3-139 Reliable Writes

Characteristic Value Notification

GATTサーバからのNotification（通知機能）をクライアントが受ける場合のProcedureをCharacteristic Value Notificationと呼びます。このProcedureでは、サーバから値を通知するNotificationと呼ばれるSub-Procedureがサポートされています。なお、この通知機能はClient Charactersitic Configuration ディスクリプタ（参照 3-10-2 項の「Client Characteristic Properties」、151 ページ）によって設定することができます。

GATT サーバからの通知を受け取る（ACKなし）（Notifications）

GATTサーバから通知を受け取る際のSub-ProcedureをNotificationsと呼びます。Notificationではサーバからの通知に対してクライアントからACKの返信は行いません。ちなみに、後述するNotificationsとIndicationsの違いはクライアントがACKを返信するか否かです。したがって、Notificationsは信頼性が求められない情報の、Indicationsには信頼性が求められる情報の伝達に利用するとよいでしょう。

サーバ/クライアント間の通信の概略を図3-140に示します。処理手順としては非常にシンプルで、サーバ側がHandle Value Notificationを利用するのみです。Handle Value Notificationについては3-9-3 項の「Handle Value Notification」、143 ページを参照してください。

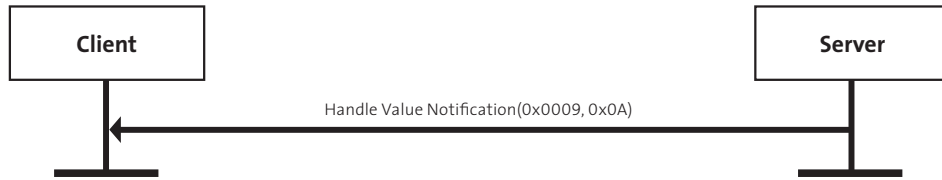


図 3-140 Notifications

Characteristic Value Indications

GATTサーバからの通知に対してACKを返信する通信処理をCharacteristic Value Indicationと呼びます。先のNotificationとは異なり、サーバからの通知に対してクライアントからACKの返信を行います。このProcedureではIndicationsと呼ばれるSub-Procedureがサポートされています。なお、この通知機能はCharacteristic Value Notificationsの場合と同様、Client Characteristic Configurationディスクリプタ（参照3-10-2項の「Client Characteristic Configuration」、151ページ）によって設定することができます。

GATTサーバからの通知を受け取る（ACKあり）（Indications）

GATTサーバから通知を受け取り、クライアントからACKの返信を要求するSub-ProcedureをIndicationsと呼びます。サーバ/クライアント間の通信の概略を図3-140に示します。処理手順としてはこちらも非常にシンプルで、サーバ側がHandle Value Indicationsを利用するのみです。クライアントはIndicationを受信後、このAttribute Protocol PDUに対するACKとしてHandle Value Confirmationを返信します。これらのAttribute Protocol PDUについては3-9-3節の「Handle Value Indication」、143ページを参照してください。

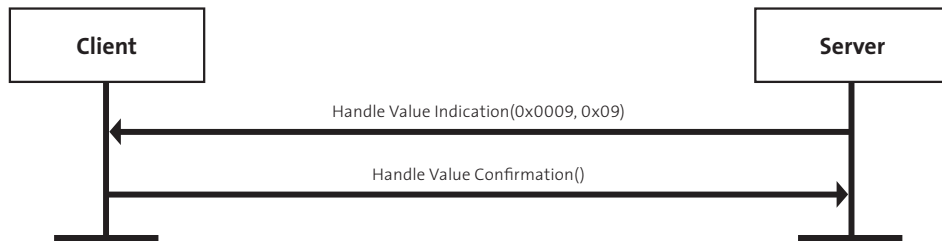


図 3-141 Indications

Characteristic Descriptors

GATTサーバ上のCharacteristicディスク립タの読み込み/書き込みを行う処理を定めたProcedureをCharacteristic Descriptorsと呼びます。このProcedureでは、Characteristicディスク립タの読み込みを行うRead Characteristic Descriptors/Read Long Characteristic Descriptors、書き込みを行うWrite Characteristic Descriptors/Write Long Characteristic Descriptors、以上の4種類のSub-Procedureが定められています。

Characteristicディスク립タを読み込む (Read Characteristic Descriptors)

GATTサーバからCharacteristicディスク립タを読み込むSub-ProcedureをRead Characteristic Descriptorsと呼びます。基本的には、処理手順としてはRead Characteristic Valueと同様で、Read Requestを利用してCharacteristicディスク립タのAttribute Handleを指定して読み込みます。サーバ/クライアント間の通信の概略を図3-141に示します。Attribute Protocol PDUについては、3-9-3項の「Read Request/Response」、135ページを参照してください。

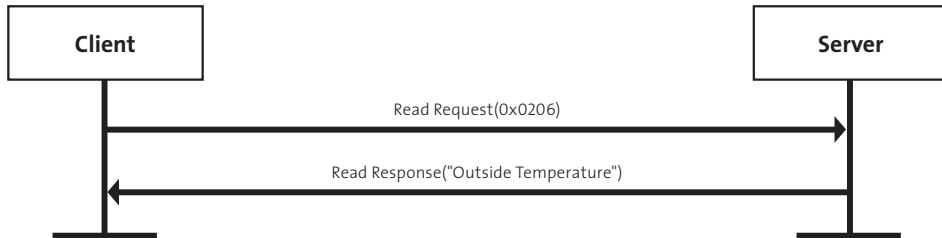


図 3-142 Read Characteristic Descriptors

〔ATT_MTU-1〕octet以上のCharacteristicディスク립タを読み込む (Read Long Characteristic Descriptors)

Read Characteristic DescriptorsではRead Requestと呼ばれるAttribute Protocol PDUを利用しているため、一度に読み込むことができるディスク립タの値は〔ATT_MTU-1〕octetに制限されてしまいます。〔ATT_MTU-1〕octet以上のディスク립タを読み込む場合の処理を記述したsub-Procedureが別途要されており、Read Long Characteristic Descriptorsと呼びます。

サーバ/クライアント間の通信の概略を図3-142に示します。基本的にはRead Long Characteristic Valueと手順は同様となるので、Sub-Procedureの挙動については図3-132を参照ください。また、Attribute Protocol PDUについては、3-9-3節の「Read Request/Response」および「Read Brob Request/Response」、135ページを参照してください。

Characteristic ディスクリプタを書き込む (Write Characteristic Descriptors)

GATT サーバから Characteristic ディスクリプタを書き込む Sub-Procedure を Write Characteristic Descriptors と呼びます。基本的には、処理手順としては Write Characteristic Value と同様に、Write Request を利用して Characteristic ディスクリプタの Attribute Handle を指定して書き込みます。サーバ/クライアント間の通信の概略を図3-143に示します。Attribute Protocol PDU については、3-9-3 節の「Write Request/Response」(139 ページ) を参照ください。

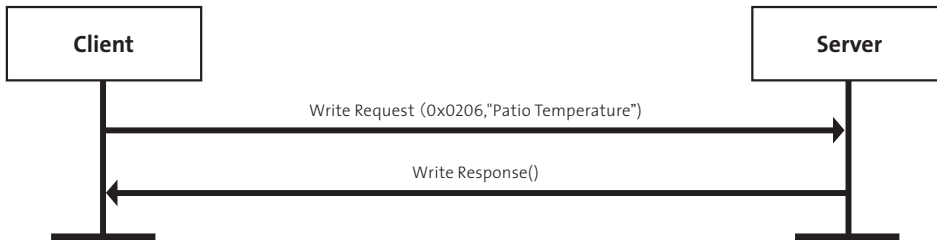


図 3-143 Write Characteristic Descriptors

〔ATT_MTU-3〕octet 以上の Characteristic ディスクリプタを書き込む (Write Long Characteristic Descriptors)

Write Characteristic Descriptors では Write Request と呼ばれる Attribute Protocol PDU を利用しているため、一度に書き込むことができる ディスクリプタの値は 〔ATT_MTU-3〕octet に制限されてしまいます。〔ATT_MTU-3〕octet 以上の Characteristic ディスクリプタを書き込む場合、Sub-Procedure として Write Long Characteristic Descriptors が用意されています。

サーバ/クライアント間の通信は、基本的には Write Long Characteristic Value と同様の手順となるので、Sub-Procedure の挙動については図 3-137 (175 ページ) を参照ください。また、Attribute Protocol PDU については、3-9-3 節の「Prepare Write Request/Response」(141 ページ) および「Execute Write Request/Response」(142 ページ) を参照ください。

3-11. iOS エンジニアの BLE あんちょこ

以上で BLE の仕様について確認してきました。Core Bluetooth プログラミングについての Tips は Part 2、4 章以降で筆者 堤が解説していますが、本項では iOS エンジニアと関連する規格上のポイントをまとめます。

3-11-1. Bluetooth Accessory Design Guidelines for Apple Products

iOS デバイスと Bluetooth デバイスを連携させるにあたり Apple からガイドラインが提供されています。このようなデバイスをアクセサリと呼んでいますが、BLE を利用したアクセサリについても Bluetooth Accessory Design Guidelines for Apple Products の 3 章に、そのガイドラインが記載されています。^{※14} アクセサリを開発するためには、アクセサリがこのガイドラインを満たしておく必要があるため、iOS 開発者側もそのガイドラインを理解しておくことが望ましいでしょう。

アクセサリが実装すべき役割 (Role)

Apple は iOS と連携するアクセサリにおいて、Peripheral もしくは Broadcaster のいずれか一方が最低限で実装されているべきだとしています。これは iOS デバイスがアクセサリと形成するネットワーク中で、基本的に Central となることを指向しているためと考えます。役割 (Role) については、「3-8-3. Mode と Procedure」を参照してください。

※14 <https://developer.apple.com/hardware/drivers/BluetoothDesignGuidelines.pdf>

Advertising チャンネル

アクセサリは iOS デバイスに対して 3ch の Advertising チャンネル (37ch、38ch、そして 39ch) の全てを利用して Advertising を行うべきだとしています。Advertising チャンネルについては 3-4-1 項の「BLE の無線通信の物理チャンネル」(62 ページ) を参照してください。

サポートする Advertising PDU

デバイスは 4 種類の Advertising PDU (ADV_IND、ADV_NOCONN_IND、ADV_SCAN_IND) の内、いずれか 1 つを利用します。残る Advertising PDU である ADV_DIRECT_IND について、Apple はガイドライン上でアクセサリで利用するべきではない、としています。Advertising PDU については、「3-6-3. Advertising PDU」を参照してください。

Advertising Packet (Advertising Data)

アクセサリが Advertising で送出する Advertising Packet は、Flags、TX Power Level、Local Name、Services の情報を少なくとも 1 つ含まなければなりません。

なお、Advertising Packet に含める情報については Supplement to the Bluetooth Core Specification、および 3-4-5 項の「Advertising Packet と Scan Response Packet」(73 ページ) を参照してください。

Advertising Packet の情報が納まらない場合はどうすべきなのか

アクセサリが省電力を意識して Advertising PDU のサイズをコンパクトに設定している、もしくは Advertising PDU に対して上記の情報が収まりきらないこともありえます。この場合、アクセサリ側は代わりに SCAN_RSP PDU に Local Name および TX Power Level などの情報を配置することが許されています。しかしこの場合、Apple の製品の全てが SCAN_RSP PDU に対応する Active Scanning に対応しているわけではない点に注意する必要があります。

また Advertising Packet 中に Services 情報を含める場合、Apple は Primary Service は常に Advertising Packet に含めて Advertising しなければなりませんが、他方、Secondary Service は含めるべきではない、としています。ただし Advertising PDU の容量に制限がある場合、アクセサリの主要な使用目的に対し重要でない Service の情報については、アクセサリ側の判断で省略され

る可能性があります。

なお、Advertising Packet と Scan Response Data のフォーマットについては、Bluetooth Core Specification Supplement, Part C Section18 に従います。

Advertising Interval の推奨設定

アクセサリの Advertising Interval については注意して設定する必要があります。これは Advertising Interval が iOS デバイスからのアクセサリの発見のしやすさや、接続のしやすさに大きく影響を与え、また、アクセサリがバッテリー駆動の場合はバッテリーの容量についても影響を与えるためです。

アクセサリが Apple 製品から発見されるためには、最初の 30 秒間の Advertising Interval の設定値 $\text{advInterval}=20\text{ms}$ とすることが推奨とされています。この 30 秒間にアクセサリが発見できなかった場合、Apple 製品から発見しやすくするために、Apple は下記の長周期な Advertising Interval の利用を推奨しています。ただし、この長周期の Advertising Interval を利用する場合、発見や接続により時間がかかる場合があり、注意する必要があります。Advertising Interval については 3-4-5 の項「Advertising Interval」(69 ページ)を参照してください。

なお、長周期の Advertising Interval (152.5ms、211.25ms、318.75ms、417.5ms、546.25ms、760.0ms、852.5ms、1022.5ms、1285.0ms) はいずれも 0.625ms の整数倍となっており、BLE の仕様を満たす値に設定されていることがわかります。

Connection Parameter の推奨設定

アクセサリが Peripheral の場合、Central との接続においていくつかの Connection Parameter を設定する必要があります。アクセサリは、その利用状態に合わせた適切な Connection Parameter を L2CAP Connection Parameter Update Request を送信することによって、適切なタイミングでリクエストするしなければなりません。詳細は Bluetooth 4.0 Specification, Volume3, Part A, Section 4.20 に記述されています。

なお、アクセサリの Connection Parameter のリクエストは下記の条件を満たさない場合、棄却されます。

- $\text{Intraval Max} \times (\text{Slave Latency} + 1) \leq 2 \text{ s}$
- $\text{Interval Min} \geq 20\text{ms}$

- $\text{Interval Min} + 20\text{ms} \leq \text{Interval Max}$
- $\text{Slave Latency} \leq 4 \text{ times}$
- $\text{connSupervisionTimeout} \leq 6\text{s}$
- $\text{Interval Max} \times (\text{Slave Latency} + 1) \times 3 < \text{connSupervisionTimeout}$

BLEを利用したHIDデバイス（これをHID over GATTとも呼びます）は、接続型のアクセサリの一つですが、この用途の場合、Apple製品として承認されるためにはConnectionにおけるインターバルは11.25m以下にする必要があります。

なお、Apple製品側でGAPサービス上のPeripheral Preferred Connection Parameters Characteristicについて、読み出したり利用したりすることはありません。

Privacy

アクセサリはいかなる状況でもResolvable Private Addressのキーが解除できるように実装されるべきです。Apple製品はプライバシーに配慮する見地からRandom Device Addressを利用します。

Permission

アクセサリ側はServiceおよびCharacteristicの発見のために特別なPermissionたとえばペアリング、認証、もしくは暗号化を要求しません。唯一、要求する場合はCharacteristic値やCharacteristic Descriptor値にアクセスする場合のみです。Permissionについては3-9-2項の「Attribute Permission」（122ページ）を参照ください。

Pairing

アクセサリ側からペアリングを要求するべきではありません（ただし、認証要件を満たさない認証コードによるリクエストをアクセサリが棄却した場合を除きます）。

BLEの仕様上、セキュリティ上の理由からPeripheralがCentralとBonding処理おこなったのペアリングを要求する場合、Peripheralは認証要件を満たさない認証コードによるリクエストを適切に棄却しなければいけません。

同様に、iOSデバイスがCentralかつGATTサーバとして動作する場合、そのような認証コード

によるリクエスト受けた場合は適切に棄却します。したがって、ペアリングを要求する iOS デバイスとやりとりするためには、アクセサリ側はペアリングの手続きを開始しなければなりません。

ペアリングについては Apple 製品上でのユーザ認証に依存します。一度、アクセサリが Apple 製品とペアリングすると、持続的な利用のために Central、Peripheral の両者で分散鍵 (DK: Distributed Keys) を保持する必要があります。もしペアリングが必要とされなくなった場合、アクセサリは鍵を削除しなければなりません。

なお、アクセサリの設計において認証機能を組み込むためには、BLE を利用する場合であっても、アクセサリは MFi プログラムを締結し、Authentication Specification をサポートする必要があり、注意が必要です。MFi についてはコラム「Bluetooth と MFi プログラム」(48 ページ) を参照してください。

MTU の容量

iOS デバイスはデフォルトの容量を超える MTU と、そのための MTU の拡張に関するリクエスト (Exchange MTU Request) をサポートしており、アクセサリ側はより大きい容量の MTU への拡張を要求することができます。MTU については 3-9-2 の「MTU (Maximum Transfer Unit)」(122 ページ) や後述する 3-11-3 項を参照してください。

Services

GAP Service

アクセサリは GAP Service 中の Device Name Characteristic を Permission が書き込み可能な状態で実装しなければなりません。Device Name Characteristic については、3-10-4 の「Device Name Characteristic 値」(158 ページ) を参照してください。

GATT Service

アクセサリがその動作中に自身の GATT データベースを変更する場合、そのアクセサリは Service Changed Characteristic を実装していなければなりません。この Service Changed Characteristic はファームウェア・アップデートなど、動的に GATT を変更する際に利用する Characteristic です。逆に考えると、アクセサリ上の Service Changed Characteristic の有無を確認することで、Apple 製品はアクセサリから読みだした Attribute の情報をキャッシュすることで、アクセサリとの通信における Service や Characteristic の発見などの操作の効率化を図ることができ

ます。

Device Information Service

アクセサリは Device Information Service を実装しなければなりません。ただし Device Information Service の UUID は Advertising Packet 中でアドバタイズされてはいけません。アクセサリは以下の Characteristic をサポートしなければなりません。以下の Characteristic の詳細は Bluetooth SIG のサイト^{※15}でも確認することができます。

- Manufacture Name String
- Model Number String
- Firmware Revision String
- Software Revision String

Available Services

iOS 7.0 を利用することで、いずれの iOS 製品も Battery Service、Time Service そして Apple Notification Center Service (ANCS) のサービスに対応することができ、アクセサリ側でそれらを利用できるようになります。

Time Service は現在時刻と現地時刻の情報を保持する Characteristic を持ちます。ただし、現在時刻を変更した時にそれらの時刻を調整する機能は提供されていません。また、ANCS は UUIDANCS=7905F431-B5CE-4E99-A40F-4B1E122D00D0 で提供されています。

これらのサービスはアクセサリと接続した直後にその機能が保証されているものではありません。またアクセサリは Service が利用可能になった場合に Service Changed Characteristic による Indication によってその旨を通知しなければなりません。iOS デバイスは接続が維持されている限り、利用可能であると通知された機能を使用することができます。

GATT サーバ

iOS6.0 以降で、アプリケーション側から GATT サーバに対して Service や Characteristic を提供することで、iOS デバイスがアクセサリを利用できるようになります。本項の推奨事項はそのような場合のアクセサリに適用します。

以下の Service は iOS によって内部で実装され、サードパーティである iOS アプリケーションに対して公開されることはありません。

※15 <https://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicsHome.aspx>

- GATT (Generic Attribute Profile) Service
- GAP (Generic Access Profile) Service
- Bluetooth Low Energy HID Service
- Battery Service
- Time Service
- Apple Notification Center Service

iOS デバイスは GAP Service 上の Service Changed Characteristic を実装しています。これは、GATT 上のデータベースの内容を任意のタイミングで変更できるようにするためです。したがって、アクセサリは Service Changed Characteristic の値の Indication をサポートし、Indication を受信したら、それに応じてデータベースのキャッシュを無効にしなければなりません。

アクセサリは ATT/GATT リクエストおよびコマンドの利用を最小限にし、必要な情報のみを送信しなければなりません。たとえば、アクセサリが特定のサービスを検索する場合、Attribute Protocol PDU で Discover All Services は利用してはいけません。この場合は Attribute Protocol PDU の Use Discover Primary Service By Service UUID で代替します。これはパケットのやりとりを削減することで電力消費を抑えアクセサリ、Apple 製品双方のパフォーマンスの向上につながるためです。

サードパーティの iOS アプリケーションがアクセサリ上の Service を検索する場合、以下の Service が iOS によって内部で利用され、発見されたアクセサリ上の Service がフィルタされた状態で Core Bluetooth によって提供されます。

- GATT Service
- GAP Service
- Bluetooth LE HID Service
- Apple Notification Center Service

サービスを所有するアプリケーションがフォアグラウンドではなく、バックグラウンドで実行する権利を持たない場合、アクセサリとのペアリングや Characteristic 値の読み書きが失敗する可能性があります。一方で、アクセサリはこのようなエラーを含んだいずれのエラーに対しても、充分に対応できるように頑健に設計されていなければなりません。

Attribute Protocol PDU において Prepare Write Request が利用された場合、キューとして積まれた全ての Attribute は同一の GATT サービスのインスタンス中に保持されます。

3-11-2. BLE はなぜ低消費電力なのか

BLE の仕様については表 3-3 でまとめました。表 3-3 を見ただけでは、BLE の何が電力の低消費化につながっているのかわかりにくいので、ここで改めてクラシック BT と BLE を比較してその仕様の違いについて確認していきましょう。クラシック BT の仕様と BLE の仕様の比較を表 3-31 に示します。

	クラシック BT	BLE
周波数帯域	2.4GHz	2.4GHz (2.400~2.4835GHz)
変調方式	GFSK	GFSK
データレート	24Mbps(Bluetooth 3.0 High Speed 時)	1Mbps
チャンネル数	79ch.	40ch (Advertising ch. =3ch. / Data ch.=37ch.)
最大出力	100.00mW	10.00mW (+10dBm)
ペアリング	必須	オプション
パケット長	1021 octet	47 octet

表 3-31 クラシック BT の仕様と BLE の仕様の比較

表 3-31 を見るとわかるように、変調方式や理論上のデータレート自体はクラシック BT と BLE 間で違いはありません。したがって、同じように通信を利用した場合、理論上は両者の消費電力に差は生まれないはずです。それでは、ここで BLE の低消費電力化に寄与している要素は何なのでしょう。この要素をまとめると、次の 3 点に尽きます。

- 送信パケット長の小型化
- 間欠動作による低消費電力化
- 通信チャンネル数の削減

送信パケット長の小型化

BLE では、送信パケット長はクラシック BLE の 1021 octet から大幅に削減され 47 octet となっています。これは 2 つ目の項目にも関係しますが、パケット長が小型化すれば送信に要する時間も相対的に短くなります。この観点から、後述する MTU についてもやりとりする Attribute の値のサイズを不用意に拡張すると通信に要する時間が増加し、アクセサリの低消費電力化を妨げる要因と

なるため、電力要件が厳しい場合は注意する必要があります。

間欠動作による低消費電力化

LED の調光（光の明るさを調節すること）をおこなう場合、PWM（Pulse Width Modulation）という手法を利用します。この手法は単純に言えばスイッチの ON/OFF を繰り返し、ON と OFF の区間の比率を調整することで LED に与える電力を調整する、というものです。BLE における間欠動作による低消費電力化も基本的には同様で、通信の区間をできるかぎり削減し、通信していない区間ではスリープをすることで、全体としての電力消費を押さえる工夫をおこなっています。

3-4-5 項の「Passive Scanning」(71 ページ)で取り扱った Passive Scanning などその工夫の一つです。Passive Scanning では、Scan 状態のデバイス（たとえば iOS デバイス）は常に周囲のデバイスからの Advertising を待ち続ける一方、周辺の BLE デバイスは自身のタイミングで Advertising をおこないます。重要なのは BLE デバイス側が Advertising を自身のタイミングで行うという点です。仮に、これが逆だった場合、周辺の BLE デバイスは iOS デバイスなどからの Advertising に変わる何らかの packets を待ち続けなければならない、BLE デバイス側は「どのタイミングでスリープし」「どのタイミングで起動していれば良いのか」を図ることができなくなってしまいます。

これと関連して、Advertising Interval の設定なども深く影響します。たとえば、Advertising Interval の間隔を短くした場合、BLE デバイスは単位時間内でより多く何らかの通信を行うことになるため、電力消費に大きく影響をうけるでしょう。したがって、上述したように Advertising Interval の設計の際は、実現したいサービスに対して Interval が適切か否かを議論することが重要だと考えます。

通信チャンネル数の削減

クラシック BT および BLE とともに、無線通信の際には周波数ホッピング・スペクトラム拡散 (FHSS) を利用しています。この際、通信時にホップする周波数の Channel 数ですが、BLE では先述した Advertising のために Advertising 専用のチャンネルを 37ch、38ch、39ch の 3 種類に限定し、少ないチャンネルのホップでも発見が容易になるよう配慮されています。発見を容易にすることで、上述した間欠動作による低電力動作につながります。

以上の項目について、アプリケーションレベルで実際にどのように対応できるか、については 6 章で堤が解説しています。低電力化を意識する必要がある場合はそちらも参照ください。

3-11-3. Core Bluetooth における 20octet が何を示すのか

Core Bluetooth の書き込みにおいて、書き込むデータはデフォルトでは 20octet (=20byte) となっていますが、これは何を意味しているのでしょうか。本項では、その理由を BLE の仕様から紐解いていきたいと思います。

GATT とデータのやりとりを行う際、下流の Controller から Host に至るまでにパケットは分解され、L2CAP に到達した時点で Host 側が利用する Information Data はデフォルトで 23octet となっています。このパケットの流れは「3-5. L2CAP (Logical Link Control and Adaption Protocol) による パケットの制御」、そして図 3-29、図 3-30 を確認できます。この Information Data のサイズが ATT_MTU であり、Information Payload が 23octet であることから必然的に $ATT_MTU_{\text{default}} = 23\text{octet}$ となります。

ここで、この Information Payload に Attribute Protocol PDU において実際のデータの他にメソッドで利用する情報も含まれている点が 20octet である理由であり、答えです。たとえば、Attribute Protocol PDU における Write Request、Write Command、Handle Value Notifications、Handle Value Indications では、Attribute Value の他に、Attribute Opcode (1octet)、Attribute Handle (2octet) が含まれます。したがって、実際に Attribute Value としてデフォルトで利用できるサイズは、

$$ATT_MTU_{\text{default}} - (\text{Attribute Opcode} + \text{Attribute Handle}) = 23\text{octet} - (1\text{octet} + 2\text{octet}) = 20\text{ octet}$$

となり、これが Core Bluetooth 上での書き込みの上限サイズ 20octet となります。

それでは Attribute に書き込める値が 20octet なのか、というとそういうわけではありません。3-9-2 項の「MTU (Maximum Transfer Unit)」(122 ページ) でも示しているように Attribute のサイズの最大値は 512octet であり、20octet はあくまでデバイス間のデフォルトでの通信量ということを示しています。

なお仕様上、読み込みの場合と書き込みの場合では一度に通信できるデータのサイズは異なります。これは Attribute Protocol PDU における Read Request をみれば明らかです。この場合、Information Data 中で Attribute Value の他は Attribute Opcode (1octet) のみのため、実際に利用できるデフォルトでのサイズは、仕様上は

$$ATT_MTU_{\text{default}} - (\text{Attribute Opcode}) = 23\text{octet} - (1\text{octet}) = 22\text{octet}$$

となります。

また、認証証明付きの書き込みを行う場合も通常の書き込み時のデータサイズとは異なり、Attribute Value の他に、Attribute Opcode (1octet)、Attribute Handle (2octet)、Authentication Signature (12octet) が含まれます。したがって、仕様上は

$$\begin{aligned} & \text{ATT_MTU}_{\text{default}} - (\text{Attribute Opcode} + \text{Attribute Handle} + \text{Authentication Signature}) \\ &= 23\text{octet} - (1\text{octet} + 2\text{octet} + 12\text{octet}) = 8\text{octet} \end{aligned}$$

となります。

MTU 拡張の使いどころ

〔ATT_MTU-1〕octet 以上のサイズの Attribute は Long Attribute と呼ばれており、当然、Long Attribute を扱うための Attribute Protocol PDU も用意されています。これに相当する Attribute Protocol PDU は Prepare Write Request や Read Blob Request であり、Long Attribute のデータを複数回にわけて送受信するために利用します (3-9-3、135 ページの「Read Blob Request/Response」を参照)。

では、MTU 拡張^{※16} はどのような場合に利用するのでしょうか？ MTU 拡張の目的は以下の 2 つに尽きます。

- ・パケットのやりとりの回数を減らす
- ・Indication や Notification のように複数回に分けて送受信を行わないようなメソッドにおいて ATT_MTU_{default} 以上のデータを送信する

パケットのやりとりの回数を減らすことは、通信の回数を減らすことで電力の削減などに寄与します。ただし、本質的にはやりとりする Attribute が少なくなるようにしておく方がよく、サービスやハードウェア、そしてアプリケーションのバランスを総合的にみて設計して利用するべきでしょう。

Indication や Notification で GATT サーバ上から送信されるデータをデフォルト値以上に増やす場合は MTU 拡張をする以外に方法はないため、どちらかといえば後者の目的での利用が MTU 拡張の大きな目的となると考えます。

いずれにせよ、ATT_MTU のサイズや MTU 拡張については、サービス、ハードウェアも含めて

※16 3-9-3 項の「Exchange MTU Request/Response」(126 ページ) および 3-10-6 項の「MTU の容量を変更する (Exchange MTU)」(164 ページ) を参照してください。

考慮する必要があり、アプリケーションのみで最適な値が決定するものではありません。この点は心の片隅にとどめておくといいいでしょう。