

ECEN 5023-001, -001B, -740

Mobile Computing & IoT Security

Lecture #14

21 March 2017

Quiz 6 review

Select all the statements below that incorporate the Bluetooth Low Energy Asymmetric Design Philosophy.

- ☒ Master devices perform scanning
- ☒ The slave does not run the profile
- ☒ A device with more energy resources are given more to do
- ☒ Radio packets are small

Agenda

- Class Announcements
- Mid-Term format
- Bluetooth Smart / Low Energy

Class Announcements

- No quiz for the week of February 27th!
- Atmel tutorial assignment due before the March 7th
 - Questions regarding the tutorial will be on the mid-term
- Mid-term will be held in class on Tuesday, March 7th, at 6:30 in class
 - For on campus students, you must be in class for the exam
 - For distant learners, the mid-term will be due by 6:00pm on Thursday, March 9th

Mid-Term

- March 7th, 2017
 - For the distant learners, the Mid-Term will be available from March 7th at 6:30pm to Thursday the 9th at 6:30pm
- Will be administered by D2L
 - 75 minute time limit for the Mid-term
 - 5 minutes time limit for the bonus section
 - 1 attempt
- Open book, but not open people

Mid-Term

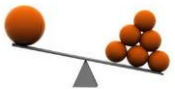
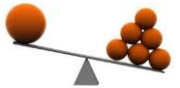
- Material covered will include:
 - All the readings from the first day of class
 - All the lectures through Thursday, March 2nd, 2017
 - All assignments
 - Atmel ATSAMB11 tutorial
- Questions:
 - 33 questions that will represent 100% of the mid-term
 - Question pool will be over 100 questions
 - 10 bonus questions each worth 1 point
 - Comprised of a random selection from the first 6 week quiz questions (roughly 150 questions in the question library)

BLE: Central (What does this device do?)

- After connecting to the peripheral, the central device will need to discover what the device does using the following four procedures:
 - Primary Services Discovery
 - Relationship Discovery
 - Characteristic Discovery
 - And, Descriptor Discovery
- The first process is the Primary Services Discovery
 - These are the services that describe what the device does
 - For example:
 - If the device has a battery, the primary services would expose the Battery Service
 - If the device has a temperature sensor, the primary services would expose the Temperature Service
 - If the device had a temperature sensor within the battery, this secondary service of the battery would not be exposed through Primary Services Discovery

BLE: Central (What does this device do?)

- Next, for each Primary Service that the central device knows could include another service (Relationship Discovery)
 - This is where the secondary service of a temperature sensor in a battery would be discovered
- The set of services that a device does not necessarily determine the set of profiles that the peripheral device supports.
 - Due to the complex algorithm to match the profile to the services provided, it is the more resource rich device, the central, that matches the profile to the peripheral
 - The benefit of this approach is that future client profile roles that use a set of services on a peripheral do not need to be designed into the peripheral when manufactured



BLE: Central (What does this device do?)

- After the primary and secondary services are discovered, the central devices enter the Characteristic and Descriptor phases
 - There is no revision numbers in BLE
 - Therefore, the only way to know if a given optional feature exists is to check for the exposure of a given characteristic that is linked to the optional feature

BLE: Central (Interacting with Services)

- Once the central device has completed its connection with the peripheral and discovered its services, it can begin to read and write characteristics and descriptors
- The protocol used to perform the reads and writes is the Attribute Protocol
 - The protocol has no state when connected or between one connection and the next
 - Any state that is maintain would be maintained in the “application layers of the BLE stack”
- **Read Characteristics:**
 - The most basic of all services is simply exposing a set of readable characteristics
 - For example, the temperature value of a temperature sensor

BLE: Central (Interacting with Services)

- **Writeable Characteristics:**

- The next level of complexity is a service that has characteristics that is both writeable and readable
- For example, the Link Loss Service is a writeable characteristic. An Alert Level can be written by the client to configure the behavior when the link between two devices is lost
 - If the client writes “No Alert” into the characteristic, then when the devices disconnect, the server will do nothing
 - If the client writes “Mild Alert” into the characteristic, then when the devices disconnect, the server will use a mild alert to notify the user
 - If the client writes “High Alert” into the characteristic, then when the devices disconnect, the server will alert the user all possible means that the server is configured

BLE: Central (Interacting with Services)

- **Control Points:**

- A control point is a type of service that the client can write to, but it has no “state”
- The control point has no “state” due to the service uses the written value immediately, and the server does not have any need to store that value after it has been consumed.
- For example:
 - A client may want a peripheral to sent out an alert immediately
 - The client could set the characteristic in the Link Loss Service to “Mild” or “High Alert” and disconnect which would then trigger the server to perform the alert
 - But, the client may not want to disconnect
 - Instead, the client could write to the Alert Level characteristic in the Immediate Alert Service
 - Immediate Alert Service is only writeable and causes an alert immediately
 - With the characteristic consumed immediately, the alert sounded, there is no reason to save state

BLE: Central (Interacting with Services)

- **State Machines:**

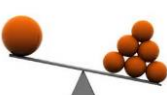
- The state machine exposes writeable control points and readable characteristics
- These expose the state of the state machine
- The difference between a state machine and a control point is that state is remembered in a state machine
- Example: A state machine for synchronization
 - It would have two states
 - The machine is doing nothing (Idle)
 - And, the machine is searching for an accurate time (Searching)
 - To control the state machine, control points are made available such as
 - Start Synchronization
 - Cancel Synchronization

BLE: Central (Interacting with Services)

- **State Machines:**

- Example: A state machine for synchronization (continued)
 - The state machine should be well defined
 - If in Idle and the control point Start Synchronization, go to Search
 - If in Search and the control point Cancel Synchronization, go to Idle
 - If in Idle and the control point Cancel Synchronization, stay in Idle (Do Not go into Error)
 - If in Search and control point Start Synchronization, stay in Search (Do Not go into Error)

BLE: Central (Notifications and Indications)

- Services expose state. Some change infrequently or random
- Polling these services would consume energy on the resource starved peripheral by transmitting state information that is not changing
 - Example: The battery state of a peripheral may change only once a day, but polling it every 15 minutes would increase the drain on the battery
 - If the battery is polled only once a day, the client may think the battery is full even though the battery may be out of charge
-  • A better process is to set up a notification
 - The client can configure the desired service's characteristics to send a notification as required
 - Most notifications are defined by the service, but some can be configured further by the client writing to additional characteristic descriptors
 - The notification will be sent during the next connection event between the client and peripheral
 - Not acknowledgement from the master is required

BLE: Central (Notifications and Indications)

- Indications are similar to notifications, but a confirmation message from the client is sent back to the server that it has received the data and that the application has received the data

BLE: Attributes background

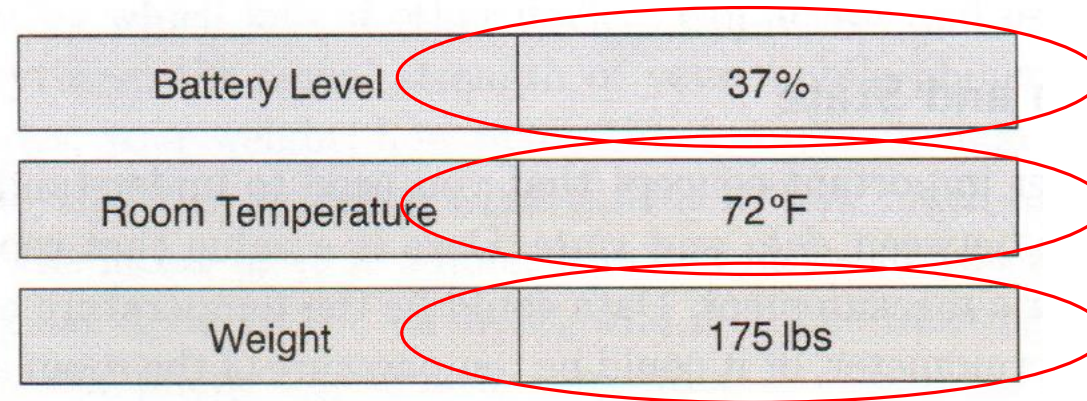
- Definition per Google: a piece of information that determines the properties of a field or tag in a database or a string of characters in a display
- In Bluetooth Smart, a basic concept is that some devices have data and others want to use it
- The device that has the data is the **server**, and the device that wants to use it is the **client**

BLE: Attributes background

- **Data** is a value that represents something such as fact or measurement
 - Examples include the temperature of a room as measured by a thermometer or read by a heating system. Multiple devices can “know” data
- **State** is a value that represents the status or condition of a device such as what it is doing or how it is operating
 - State is only know by one device, so only one device can hold state information
- For our Bluetooth Smart discussions, “state” will refer to information (data) that resides on the server and that “data” refers to information as it is in transit from the server to the client and held on the client
- The data on the client is **not** authoritative because the server’s state could have changed since the client last received it

BLE: Attributes background

- Bluetooth Smart uses three different kinds of state:
 - **External** state is measured by an external sensor and that every time that it is read, it may change such as a temperature sensor



Battery Level	37%
Room Temperature	72°F
Weight	175 lbs

Figure 10–3 Physical measurements

BLE: Attributes background

- Bluetooth Smart uses three different kinds of state:
 - **Internal** state corresponds to the state of a state machine that represents the internal state of the device

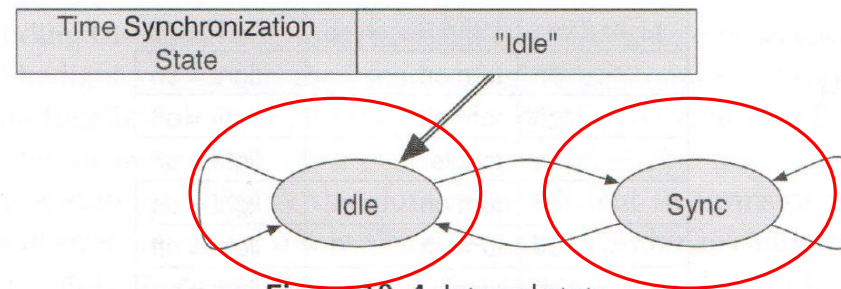


Figure 10-4 Internal state

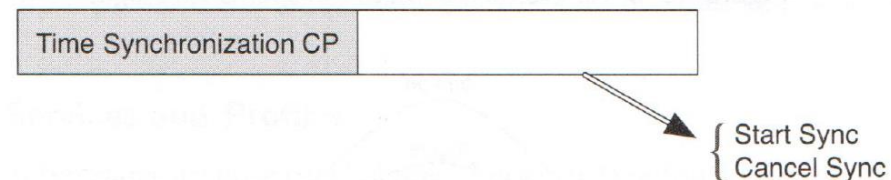


Figure 10-5 Abstract state

BLE: Attributes background

- Bluetooth Smart uses three different kinds of state:
 - **Abstract** state is state information that is only relevant at a momentary point in time and it does not represent external or internal state of the device

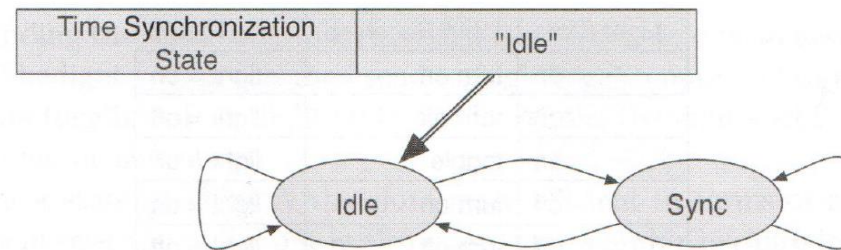


Figure 10-4 Internal state

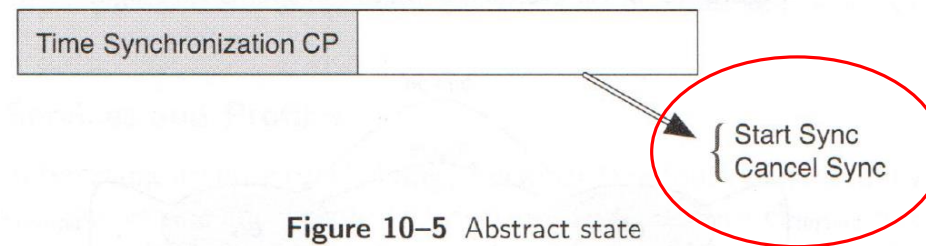


Figure 10-5 Abstract state

Examples of an abstract state are **control points**. They are commands that tell a device to perform an action, but in itself it has not state.

BLE: Attributes background

- A server's behavior is defined in a **service** specification
- The server specification defines the state that is exposed by the server using an attribute database
- Attributes on a service may be readable returning historical or current data while some attributes may be writeable to send commands (control points) to the server
- The profile specifications define how to use one or more services to enable a given use case

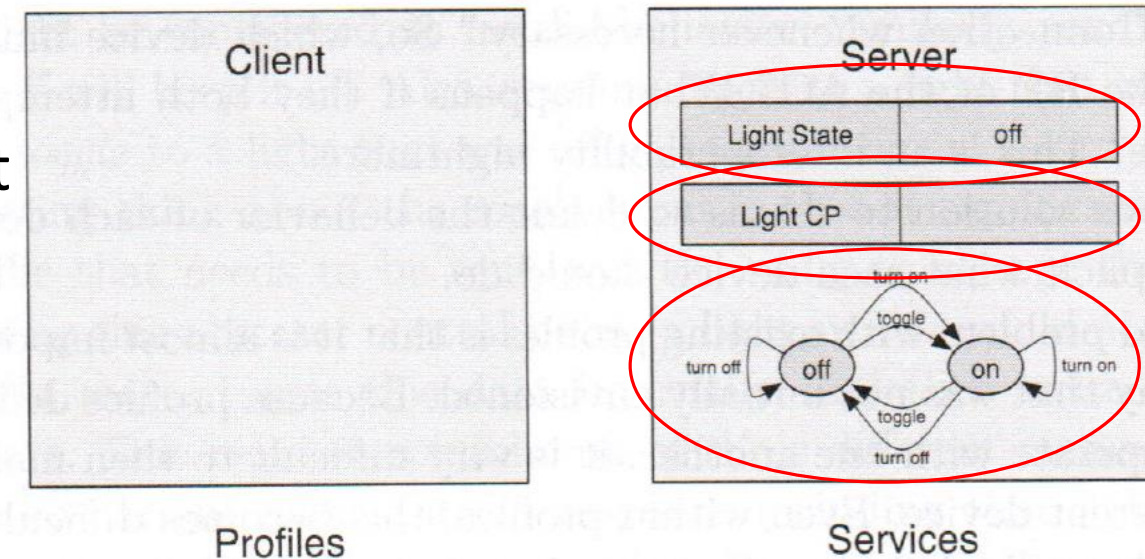


Figure 10–8 The profile/service architecture

BLE: Attributes background

- The profile specifications define how to use one or more services to enable a given use case
- As an example, how to configure the attributes exposed for the service in an attribute database on the server to ask the server to do something that the client needs it to do
- Defining specifically the server's services and independent of the client enables the service to be independently tested and independent of a client
- Any client that is given access, a connection, to the service, can use the service

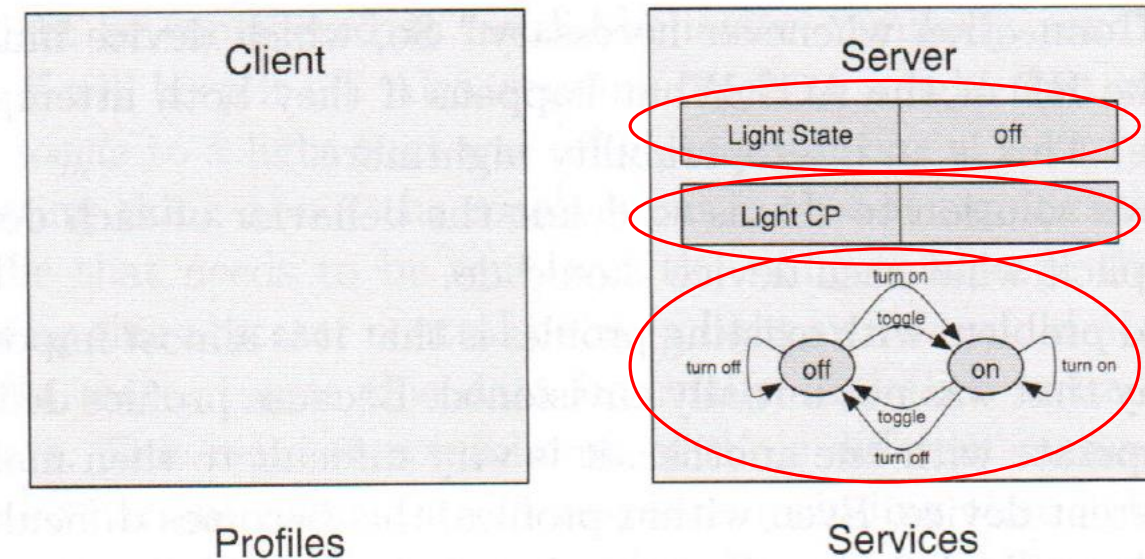


Figure 10–8 The profile/service architecture

BLE: Attributes (background)

- A client's behavior is defined in a **profile** specification
- As an example, a light switch would implement the light **profile** that knows how to control the light state through the server attributes and control points
- The light switch, the client, knows the behavior of the light service because the server specification, Light Service, to be the same for every instance
- The client profiles are essentially a set of rules for discovering, connecting, configuring and using a service

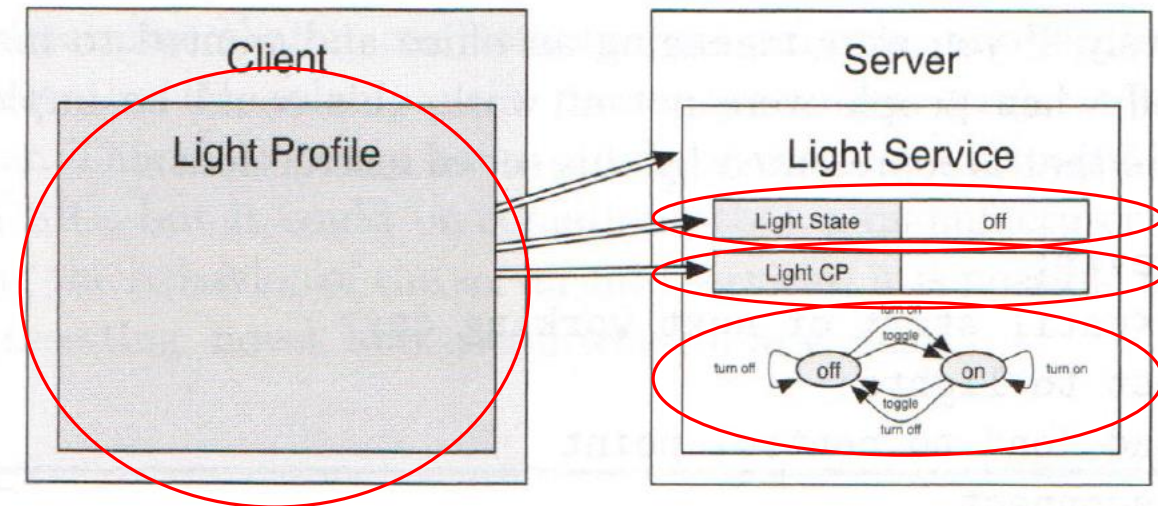


Figure 10-9 An example of a light profile and service

BLE: Attributes background

- Example 1: A home security system that knows that if the house is unoccupied and the homeowners would like the house look like it was occupied, a simple client profile could be written to the light service

Loop forever:

Wait <random period from 10 seconds to 3 hours>

Connect to a light:

Send "toggle" to control point
disconnect

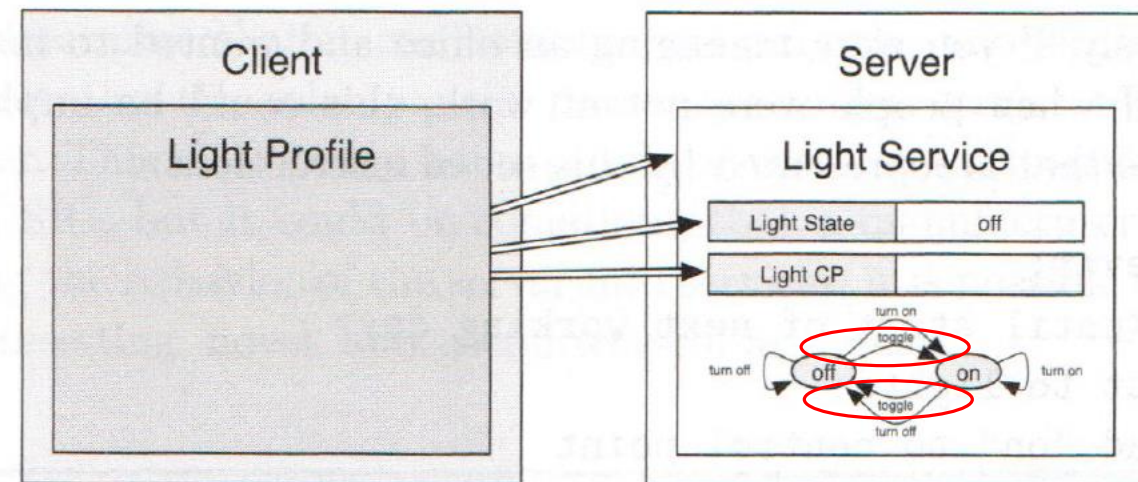


Figure 10-9 An example of a light profile and service

BLE: Attributes background

- Example 2: Take the same light service Server and install it in an office. In this example, the office managers would like the lights to be on while the staff is in the office

Loop forever:

Wait <until start of next working day>

Connect to lights:

Send “on” to control point

Disconnect

Wait <until end of work day>

Connect to lights:

Send “off” to control Point

Disconnect

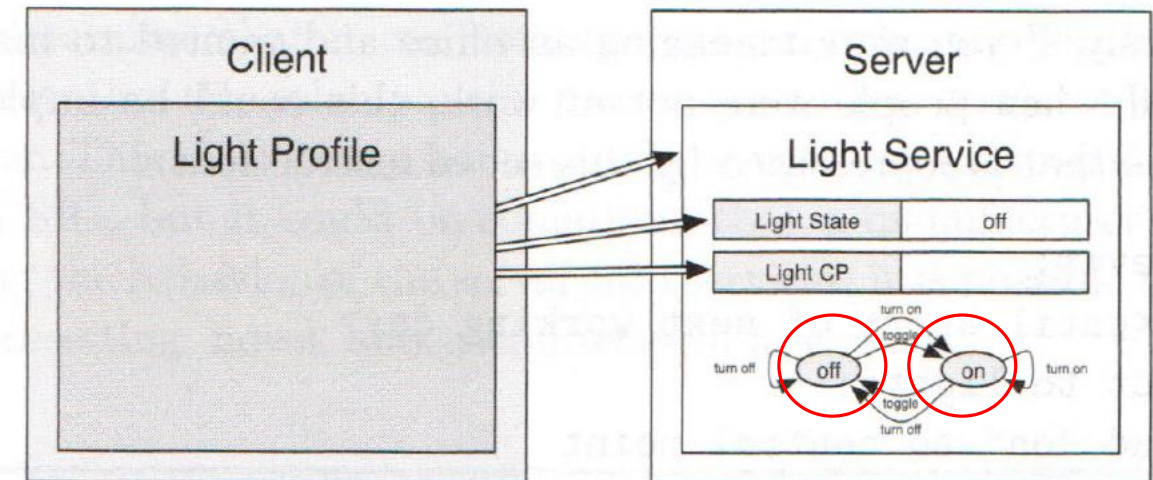


Figure 10–9 An example of a light profile and service

BLE: Attributes background

- ★ • The powerful combinations of the different services within the client profile make Bluetooth Low Energy. In the previous examples the client profile could be a mixture of services of occupancy sensors, time of day, and light services.
- ★ • Each individual service can be kept very simple. For this model to work, the services must be atomic
 - Atomic in this context means that services perform only one set of actions. Make the services atomic make the services available to different clients as well as different client profiles.
 - If services were not atomic, the client profile previously may not have had access to the occupancy or timer services available in a device
- ★ • Another key concept is that the services are not dependent on each other which means any combination of services is possible

BLE: Attributes

- An **Attribute** is a piece of labeled, addressable data
- The Bluetooth Smart attribute is composed of three values:
 - Attribute handle:
 - Attribute type:
 - Attribute value:

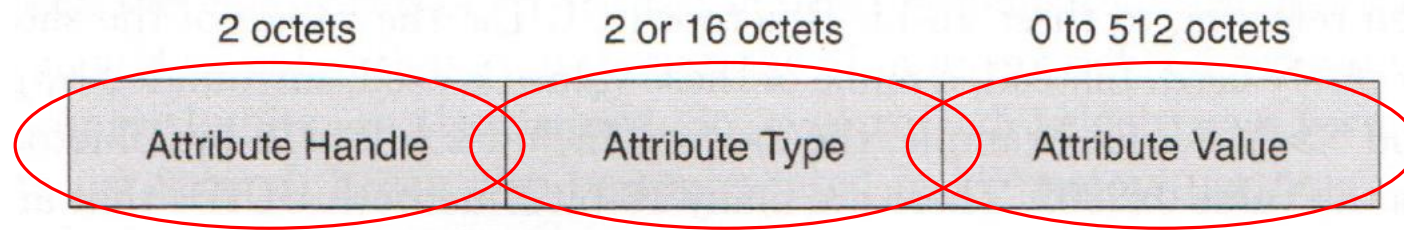


Figure 10–10 The structure of an attribute

BLE: Attributes

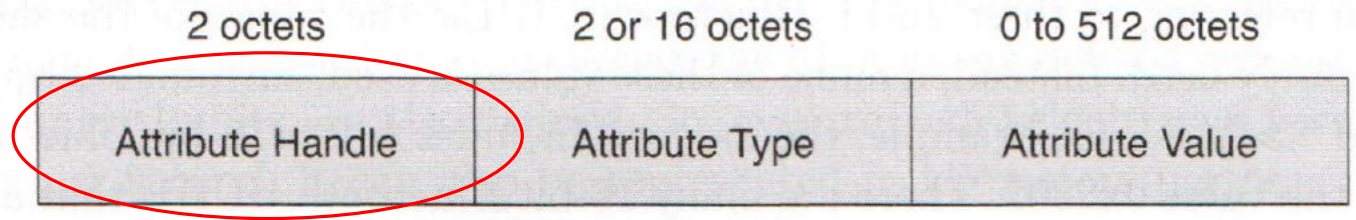


Figure 10–10 The structure of an attribute

- The attribute **handle** can be considered the memory address of the attribute
- For example, a device could have two temperature sensors which would have the same attribute type
 - To get the attribute value for the first temperature sensor you would need to access it through its attribute **handle**.
 - Similarly, to read the second temperature sensor you would need to read it through its own attribute **handle**.

BLE: Attributes

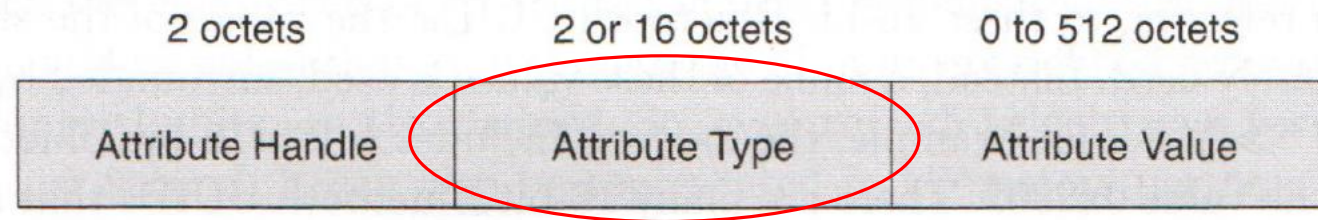


Figure 10–10 The structure of an attribute

- The attribute **type** is the type of data that the attribute value exposes such as temperature, pressure, power, time, etc.
- To address all the possible attribute types, a 128-bits is used for the attribute type which is called a Universally Unique Identifier (UUID)
- 128-bits, 16-bytes, to address every attribute type would be very energy expensive to transmit on the BLE radio. For the most common attributes, there is an abbreviated 16-bit, 2 byte, UUID which is combined with the Bluetooth Base UUID to provide the complete 128-bit UUID

00002A01 – 0000 – 1000 – 8000 – 00805F9B34FB

BLE: Attributes

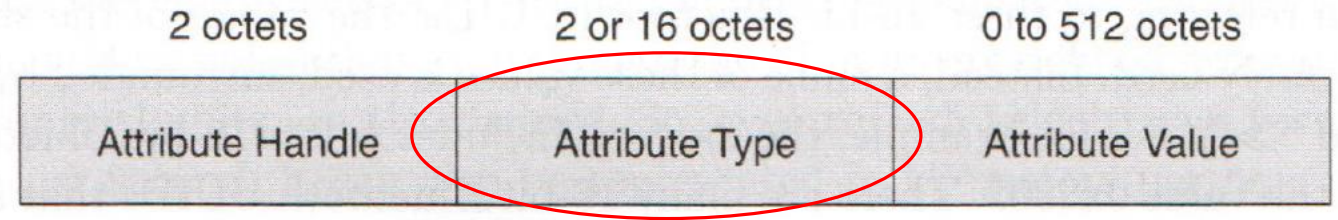


Figure 10–10 The structure of an attribute

- When referring to the 16-bit UUIDs, instead of the actual 16-bits, for readability, they are named inside these brackets << >>
 - Examples: <<Include>> refers to the BLE 16-bit UUID 0x2802
- The 16-bit UUIDs are arranged in the following groups for readability:

0x1800 through 0x26FF are for Service UUIDs

0x2700 through 0x27FF are for Units

0x2800 through 0x28FF are for Attribute Types

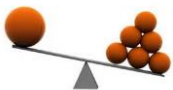
0x2900 through 0x29FF are for Characteristic Descriptors

0x2A00 through 0x7FFF are for Characteristic Types

BLE: Attributes

0x1800 through 0x26FF are for Service UUIDs
0x2700 through 0x27FF are for Units
0x2800 through 0x28FF are for Attribute Types
0x2900 through 0x29FF are for Characteristic Descriptors
0x2A00 through 0x7FFF are for Characteristic Types

- Each BLE **service** can be identified by a UUID. The 16-bit shortened UUID allows 3,840 unique services to be addressed
- Many of the attribute values can be represented in different units such as degree Celsius or Fahrenheit.
 - The **Units** 16-bit UUID are derived from the Bureau International des Poids et Mesures or otherwise known as the International System of Units
 - The **Unit** field enables the client to take the attribute value and appropriately apply it to the profile. For instance, if the attribute was the velocity of a car, the client profile may require the value in MPH or KPH. Knowing what the attribute value is, the client can convert it to the profile's input requirements if required



BLE: Attributes

0x1800 through 0x26FF are for Service UUIDs

0x2700 through 0x27FF are for Units

0x2800 through 0x28FF are for Attribute Types

0x2900 through 0x29FF are for Characteristic Descriptors

0x2A00 through 0x7FFF are for Characteristic Types

- The **Attribute Type** 16-bit UUIDs are the most fundamental attribute types.
 - These are typically used for the attributes types defined by the Generic Profile, and not a service.
 - Primary Service
 - Secondary Service
 - Include
 - Characteristic
- Some data exposed by a service may require additional data. The additional data is described by using **Characteristic Descriptors**

BLE: Attributes

0x1800 through 0x26FF are for Service UUIDs
0x2700 through 0x27FF are for Units
0x2800 through 0x28FF are for Attribute Types
0x2900 through 0x29FF are for Characteristic Descriptors
0x2A00 through 0x7FFF are for Characteristic Types

- Each unique type of value that is exposed to a service is allocated a **characteristic type**
 - The **characteristic types** 16-bit UUIDs enables 22,015 different characteristic types to be defined
 - This enables a client to discover all the different types of data that a server has
 - Each **characteristic type** has a defined format and representation

BLE: Attributes

- A collection of attributes is called a **database**
 - A database can be very small, a minimum of 6 attributes
 - Or very large and complex
 - The complexity is not at the attribute layer, but how those attributes are used in services and profile
 - The attribute database is always contained on the attribute server
 - An attribute client uses the Attribute Protocol to communicate with the attribute server
 - With only one attribute server on a device, there is only one attribute **database**
 - For a BLE device, the attribute **database** includes a Generic Access Profile service that is mandatory to support
 - Since both the client and server require the Generic Access Profile, every BLE device includes an attribute server and **database**

BLE: Attributes

- Attribute database example
 - GAP Service Attribute
 - TX Power Service
 - Battery Service

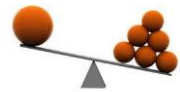
Attribute Handle	Attribute Type	Attribute Value
0x0001	Primary Service	GAP Service
0x0002	Characteristic	Device Name
0x0003	Device Name	"Proximity Tag"
0x0004	Characteristic	Appearance
0x0005	Appearance	Tag
0x0006	Primary Service	GATT Service
0x0007	Primary Service	Tx Power Service
0x0008	Characteristic	Tx Power
0x0009	Tx Power	-4dBm
0x000A	Primary Service	Immediate Alert Service
0x000B	Characteristic	Alert Level
0x000C	Alert Level	
0x000D	Primary Service	Link Loss Service
0x000E	Characteristic	Alert Level
0x000F	Alert Level	"high"
0x0010	Primary Service	Battery Service
0x0011	Characteristic	Battery Level
0x0012	Battery Level	75%
0x0013	Characteristic Presentation Format	uint8, 0, percent
0x0014	Characteristic	Battery Level State
0x0015	Battery Level State	75%, discharging
0x0016	Client Characteristic Configuration	0x0001

Figure 10–11 An example of an attribute database

BLE: Attributes

- Some attributes contain information that can only be read or written
- To support these restrictions upon access, each and every attribute in an attribute database has a **permission**
 - Access permission
 - Determines what types of requests are permitted: Readable, Writeable, or Readable/Writeable
 - If a request is attempted without the proper access permission, an error will be returned stating that the client cannot read this attribute or write to this attribute
 - Authentication permission
 - Determines whether Authentication is required or no authentication is required
 - Upon access to an attribute value, in addition to checking whether proper access permission, the attribute server checks with authentication is required. If authentication is required, the client that sent the request is authenticated with the device
 - If the client has not been authenticated, an error stating that there is insufficient authentication will be returned

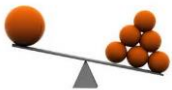
BLE: Attributes



- If the client receives the insufficient authentication error, it can do one of two things:
 - Ignore the request and pass the error up to the application
 - Or, attempt to authenticate by using its Security Manager and resend the request
- The complexity of authentication and reissuing the request is up to the client, and the server is not required to hold state of the request
- Authorization permission
 - Upon access to an attribute value, in addition to checking whether proper access permission, the attribute server checks whether the client has been authorized. If authorization is required, the client that sent the request is determined whether it has been authorized
 - If the client has not been authorized and receives the error of insufficient authorization, it is an error that the client can not resolve
 - Authorization is a property of the server, and the server either authorizes the client or it does not
- Note: Attribute permissions only pertain to the Attribute value

BLE: Attributes (accessing)

- Each attribute in the attribute database can be accessed using the following types of messages:
 - **Find Requests:** a client can find attributes in an attribute database so that the more efficient handle-based requests can be used
 - **Read Requests:** A request to read one or more attribute values using one or more handles or a range of handles to read
 - **Write Requests:** These requests always use an attribute handle and the value to write
 - These three requests always cause the attribute server to send a single response. If more data is required than what can fit in a single response, the client will need to resend the request by adjusting the attribute handles
- To minimize complexity of the server, only one request can be sent at a time, and another request can only be sent after the previous response has been received



BLE: Attributes (accessing)

- Each attribute in the attribute database can be accessed using the following types of messages:
 - **Write Command** does not require a response which makes it different than the write request
 - The **notification** which is sent by the server to the client without a request from the client does not require a response as well
 - Since the **Write Command** and **Notifications** do not require a response, they are considered unreliable, but they can be very useful in the correct use case
 - The last way to access an attribute is the **Indication**. An indication is similar to the notification in that the server provides this information without the request of the client, but an indication does require a response, so it is considered reliable

BLE: Attributes (Prepare Write Requests and Execute Write Requests)

- These messages enable a device to prepare a whole sequence of writes and then executes them as a single transaction
- Each **Prepare Write Request** include:
 - The handle of the attribute to be written
 - The value of the attribute
 - An offset into the attribute's value where this value will be written
- Enabling each **Prepare Write Request** to write a portion of a large attribute value
- **Prepare Write Response** include:
 - The handle of the attribute to be written
 - The value of the attribute
 - An offset into the attribute's value where this value will be written
- The **Prepare Write Response** enables the client to verify that the server has the correct write request before issuing the **Execute Write Request**
- If the **Prepare Write Response** did not match the Prepare Write Request, the client to issue the "cancel" code in the **Execute Write Request**

BLE: Attributes (Grouping)

- The Generic Attribute Protocol only defines a flat structure of attributes
 - Each Attribute has its handle as an address
- The Attribute Protocol defines groups as in object oriented programming of attributes
- Definitions:
 - An **interface** is a description of external behavior
 - A **class** is an implementation of that interface
 - An **object** is an instantiation of that class

BLE: Attributes (Grouping)

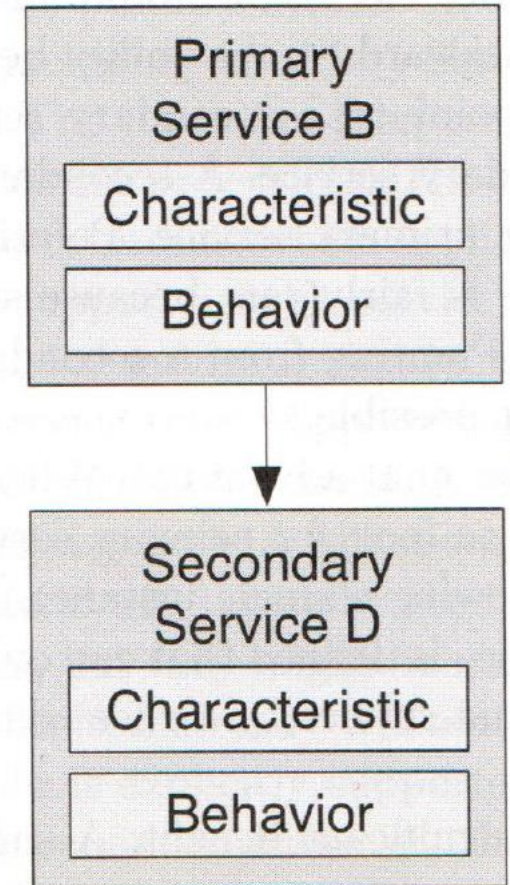
- Example:
 - A car is an **instance** of an automobile **class** that implements the driving **interface**
 - Not all car objects look the same, they can be implemented differently, but critically, they all have the same basic driving **interface**
 - Driving wheel, the accelerator pedal, and brake pedal
 - The driving interface is the same, but the class that implements this interface can be different
 - General Motors, BMWs, Hondas, etc.
 - A service is grouped by using a service declaration
 - Grouping of one or more characteristics
 - And, a characteristic is grouped using a characteristic declaration
 - Grouping of one or more attributes

BLE: Services

- The Generic Attribute Profile defines two basic forms of groupings:
 - Services
 - And, Characteristics
- Service
 - Equivalent of an object that has immutable interface
 - Includes one or more characteristics
 - Can reference other services
 - The set of characteristics and their associated behavior encompasses the immutable interface for the service
- Characteristic
 - A unit of data or exposed behavior

BLE: Services

- Primary and secondary services
 - A primary service exposes what a device typically does
 - Example: Battery gas gauge's primary service is to provide how much charge is in the battery
 - A device can have both a primary and secondary services
 - If a device supports a Service B, Service B would be instantiated as a primary service
 - If the device has additional information, but not associated with the what the device does, this secondary service, Service D, would be instantiated as a secondary service.
 - A secondary service is an encapsulation of behavior and characteristics that are not something that a user would necessarily need to understand
 - Example: The battery gas gauge providing the temperature of the battery



BLE: Services

- **Secondary** services can only be found by reference and must always have another service that points to them
- A **primary** service can have more than one **secondary** service which can create a tree of services
- **Primary** services can easily be located by a client that is looking for a particular service
 - This enables a simple client to locate whether a desired service is on a device with minimal effort
- Once the client builds up the “tree” of services or multiple “trees,” a “forest,” the client can pass this information to the application to determine how to use the services

BLE: Services

- Service Declaration
 - A service is grouped by using a service declaration which is an attribute type of Primary Service or Secondary Service
 - Simple clients are devices that have no user interface but can still use services on a peer device
 - A simple device can just search for the primary services and find the services that it requires
 - Since the Primary services are directly accessible, the simple client does not need to walk through the “tree” of services to locate it

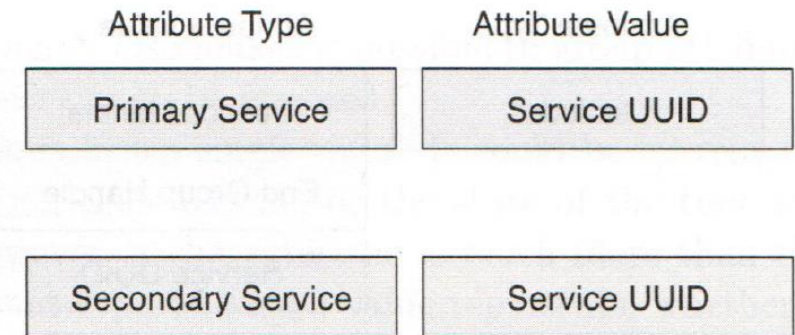


Figure 10–21 Primary and secondary service declaration

BLE: Services

- Including Services
 - Secondary services must be discovered separately
 - Each service can have zero or more Include attributes
 - Include attributes always immediately follow the service declaration and come before any other attributes for the service
 - The Include attribute definitions encompass the handle range for the referenced service along with the Service UUID
 - Enabling quick discovery of the referenced services, their group attributes, and type of services

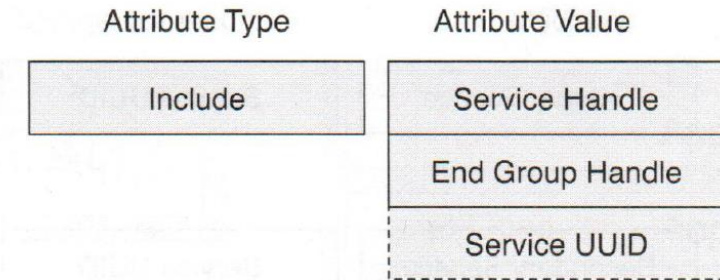


Figure 10–22 The structure of the Include declaration

Handle	Type	Value
0x0001	Primary Service	Service B
0x0002	Include	0x0200, 0x0209, Service D
...
0x0100	Primary Service	Service A
0x0102	Include	0x0300, 0x0317, Service C
...
0x0200	Secondary Service	Service D
...
0x0300	Primary Service	Service C
...

Figure 10–23 An example of an attribute database of Services A(C), B(D)

BLE: Characteristics

- A characteristic is just a single value
- However, a characteristic needs to expose the following:
 - What type of data a value represents
 - Whether a value can be read or written
 - How to configure the value to be indicated or notified or broadcast
 - And what the value means
- To expose this additional information, a characteristic is composed of three basic elements:
 - **Declaration**: start of the characteristic and groups all the other attributes for the characteristic
 - **Value**: the actual value of the characteristic
 - **Descriptors**: the additional information or configuration of the characteristic

BLE: Characteristics

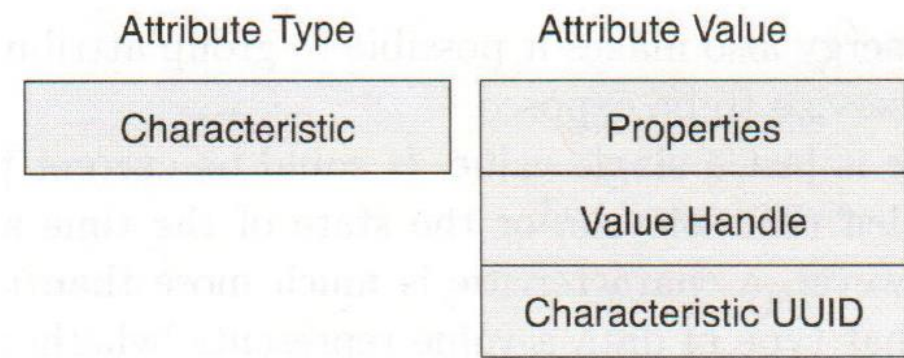


Figure 10–24 Characteristic Declaration

- Characteristic Declaration starts a characteristic and contains three fields:
 - **Characteristic properties:** specifies if the characteristic value attribute can be read, written, notified, indicated, broadcasted, commanded, or authenticated in a signed write
 - **Handle of the value attribute:** the handle of the attribute that contains the value for the characteristic
 - **Type of characteristic:** The characteristic UUID is used to identify the type of characteristic value

BLE: Characteristics

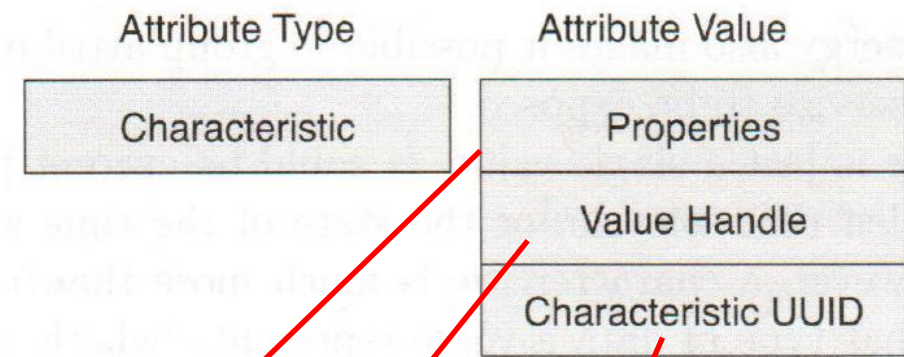


Figure 10-24 Characteristic Declaration

Handle	Type	Value
0x0001	Primary Service	GAP Service
0x0002	Characteristic	read write, 0x0003, Device Name
0x0003	Device Name	"Proximity Tag"
0x0004	Characteristic	read, 0x0005, Appearance
0x0005	Appearance	Tag

Figure 10-25 Characteristic example

BLE: Characteristics

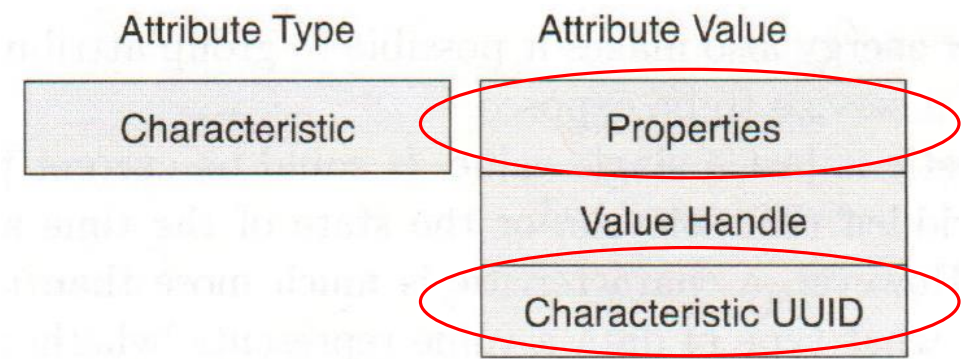


Figure 10–24 Characteristic Declaration

- **Characteristic Value**
 - The characteristic value is an attribute with the type that must match the characteristic declaration's characteristic UUID field.
 - Types of actions that can be performed on the characteristic value is exposed by the characteristic declaration or in the characteristic extended properties descriptor
 - Characteristics have no behavior, so the service specification with which this characteristic is grouped should specify the behavior exposed by this instance of the characteristic

BLE: Characteristics

Characteristic Value

- Descriptors
 - Examples of descriptors that may be included with a characteristic
 - **Extended Properties**: at this time, only two have been defined: perform reliable writes on the value and the ability to write to Characteristic User Description descriptor
 - **User Description**: an associate text string to go with the server such as a description where the light is located
 - **Client Characteristic Configuration**: only required if specifying whether the characteristic is notifiable or indicatable
 - **Server Characteristic Configuration**: a single bit that causes some data associated with the service to be broadcasted which the characteristic is grouped
 - Example: “Room Temperature Service: 20.5C

BLE: Characteristics

Displayed value = characteristic value X 10^{exponent}

- Descriptors

- Examples of descriptors that may be included with a characteristic
 - **Characteristic Presentation Format:** a multiple-field value that contains the following information:
 - **Format:** similar to variable declarations in C such as int, unsigned_16, float, etc.
 - **Exponent:** base 10 exponent that is a **signed** integer
 - **Unit:** UUID defined in the assigned numbers document such as Temperature in Celsius
 - **Namespace:** single byte that determines which organization controls the descriptor field
 - **Description:** a 16-bit unsigned number that is like an adjective. For example, if the thermometer exposes two temperature characteristics, this field could specify whether that particular characteristic is “inside” or “outside”
 - **Characteristic Aggregation Format:** used to reference two characteristic values that are concatenated together into a “single” value
 - Example: GPS coordinates would have a complex characteristic single value that is actually composed of the latitude and longitude values

BLE: Attribute Protocol

- A simple protocol by which an attribute client can find and access attributes on an attribute server
- The six structured basic operations are:
 - Request
 - Response
 - Command
 - Indication
 - Confirmation
 - Notification

BLE: Attribute Protocol

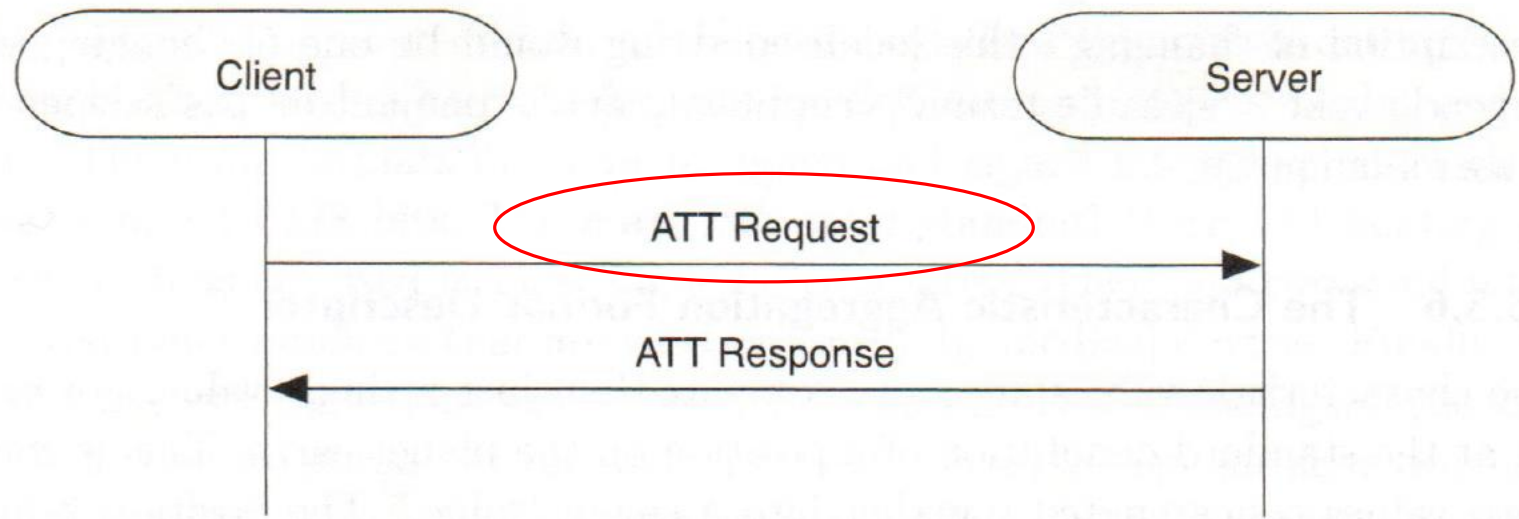


Figure 10–26 An Attribute Protocol request

- The client can send only one request at a time
- The server only has two options for the response
 - A response directly associated with the request
 - An error message

BLE: Attribute Protocol

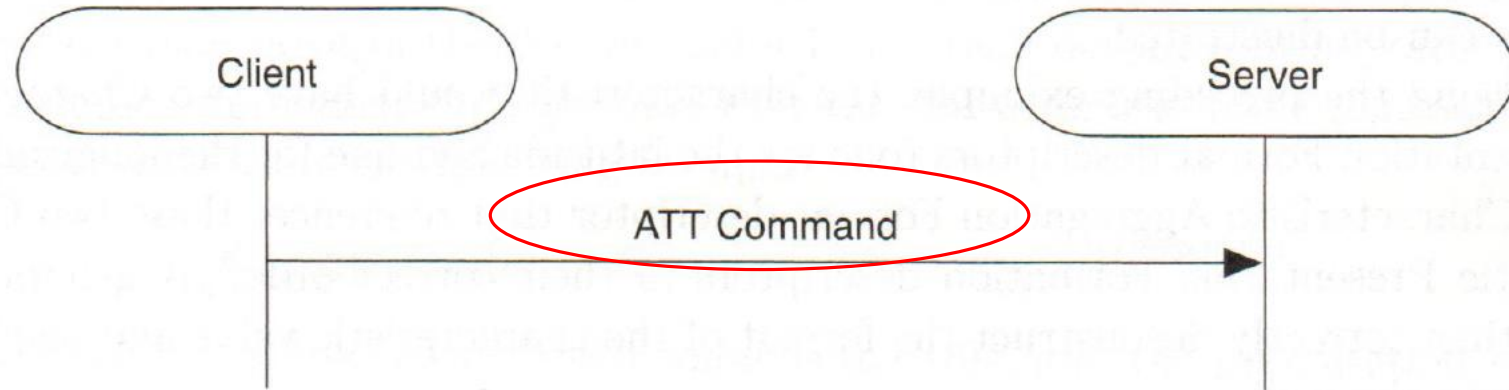


Figure 10–27 An Attribute Protocol command

- The client sends a command to a server but receives no response
- The client sends a command when it wants the server to perform an action with no requirement for an immediate response

BLE: Attribute Protocol

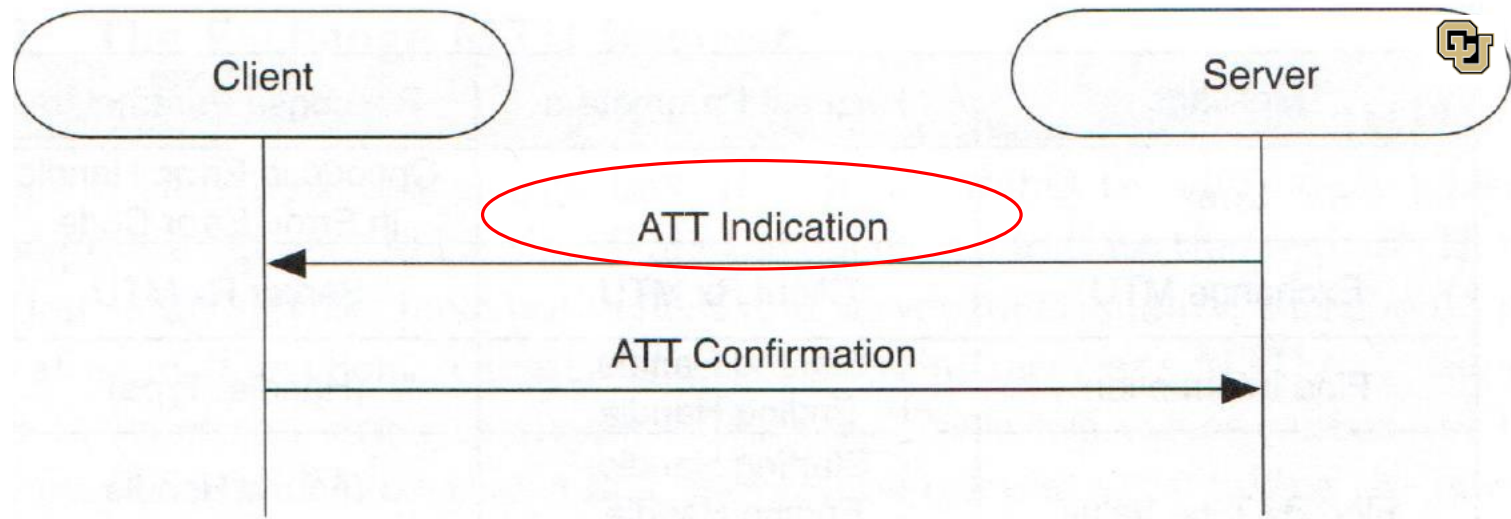
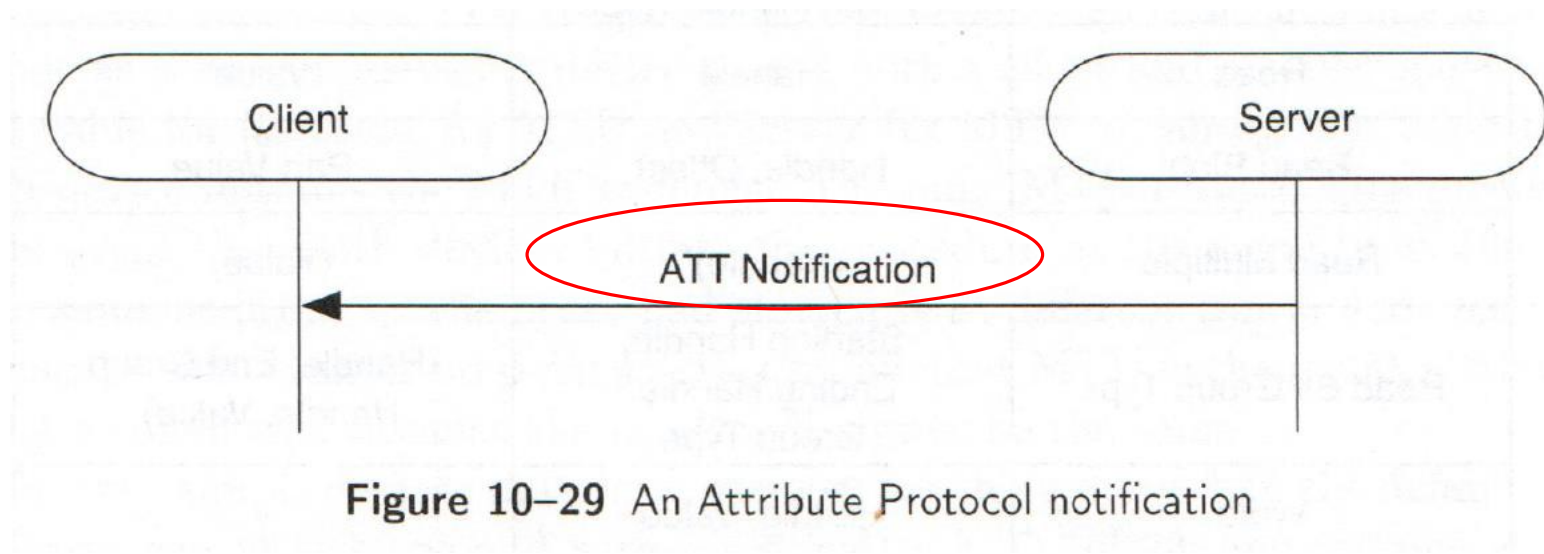


Figure 10–28 An Attribute Protocol indication

- Indications are sent by a server to a client to inform the client that a particular attribute has given a value
- Indications are similar to requests in that only one indication can be sent and a confirmation is required by the client

BLE: Attribute Protocol



- The server can send notification to a client to inform a client that a value of a particular attribute has changed, but does not require a confirmation

BLE: Attribute Protocol

- Find Information Request and Response
 - Find handle and type information for a sequence of attributes
 - The requests includes two handles: a starting handle and an ending handle
 - Typically, the response packet is too short to complete this request, so the client must repeat the operation with incrementing the starting handle to be one more than the last response handle

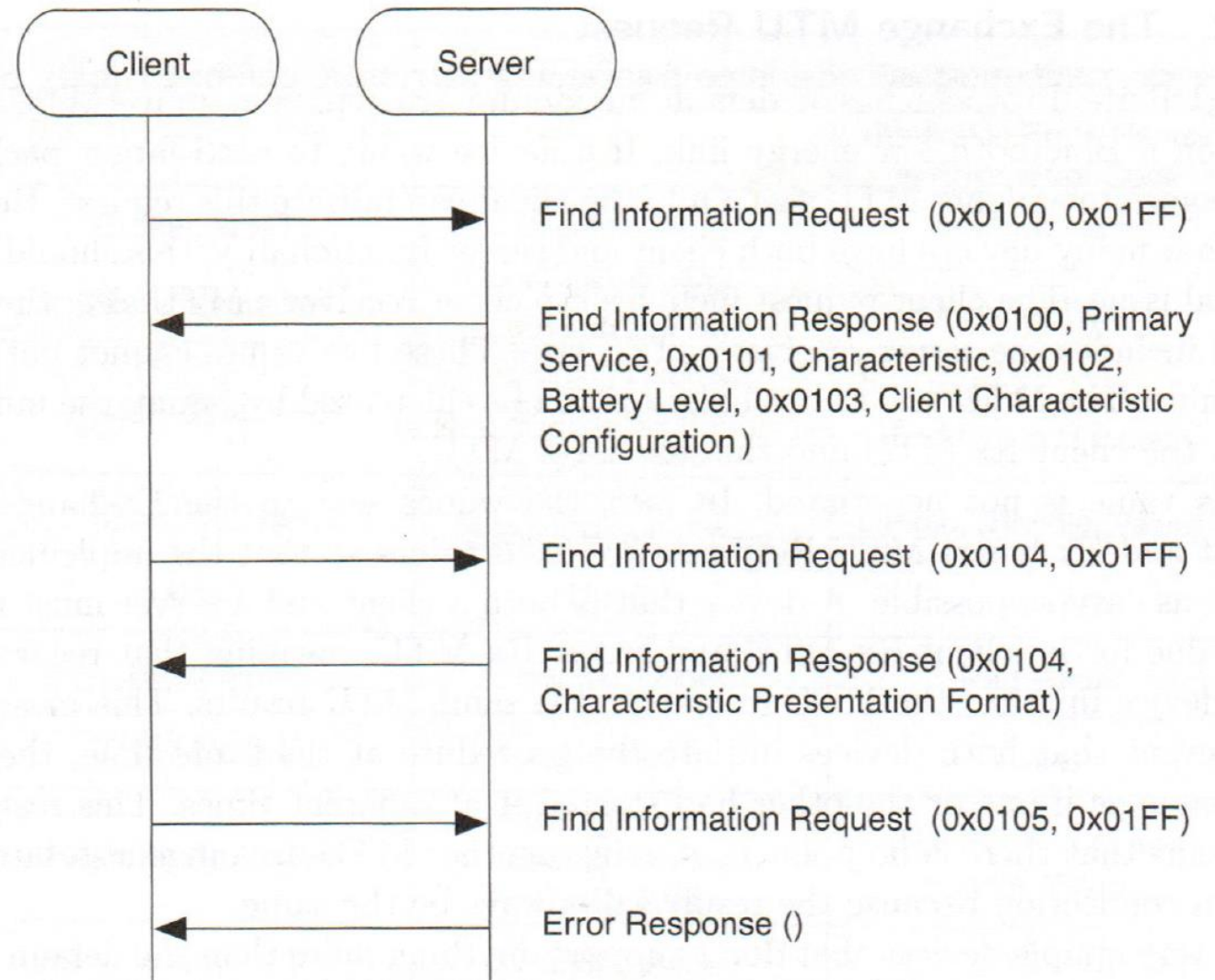


Figure 10–31 The Find Information Request