sending a pairing request to Side 2. This is shown in Figure 11.14. The main points to note are:

- IO capabilities are set to NoInputNoOutput.
- The secure connections pairing flag is set to Yes.
- Keypress notifications are set to Yes. If both sides set this field, then keypress notifications will be generated and sent.
- Key distribution is set to No. This means that no keys will be distributed by the master in Phase 3.

Side 2 responds with a pairing response as shown in Figure 11.15. The main points to note are:

- IO capabilities are set to KeyboardDisplay.
- Secure connections pairing is set to Yes.
- Keypress notification is set to No. Because one side has set this to Yes and the other side has set this to No, keypress notifications will not be used.
- Key distribution is set to No. This means that no keys will be distributed by the Slave in Phase 3.

11.10.2.2 Pairing Phase 2

The sequence of steps that happens after the two sides exchange feature information is shown in Figure 11.16. These steps are:

1. Pairing public key rxchange

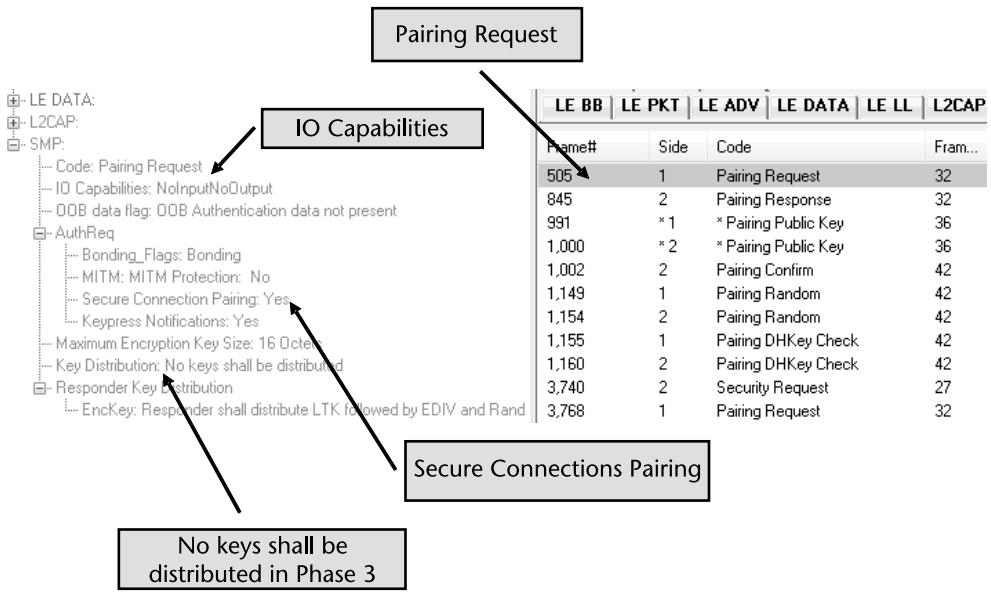

Figure 11.14 Example of a Pairing Request.

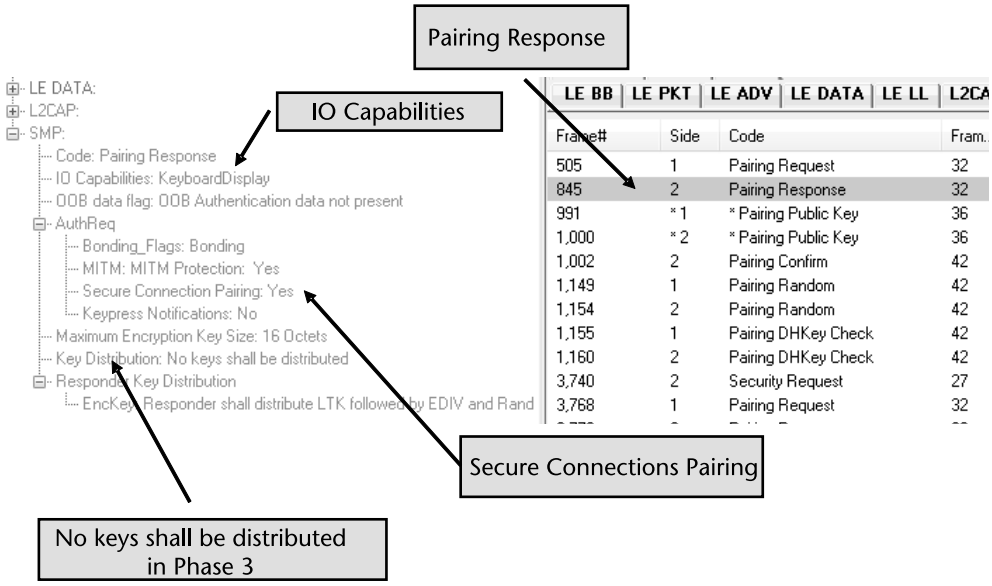

Figure 11.15 Example of a Pairing Response.

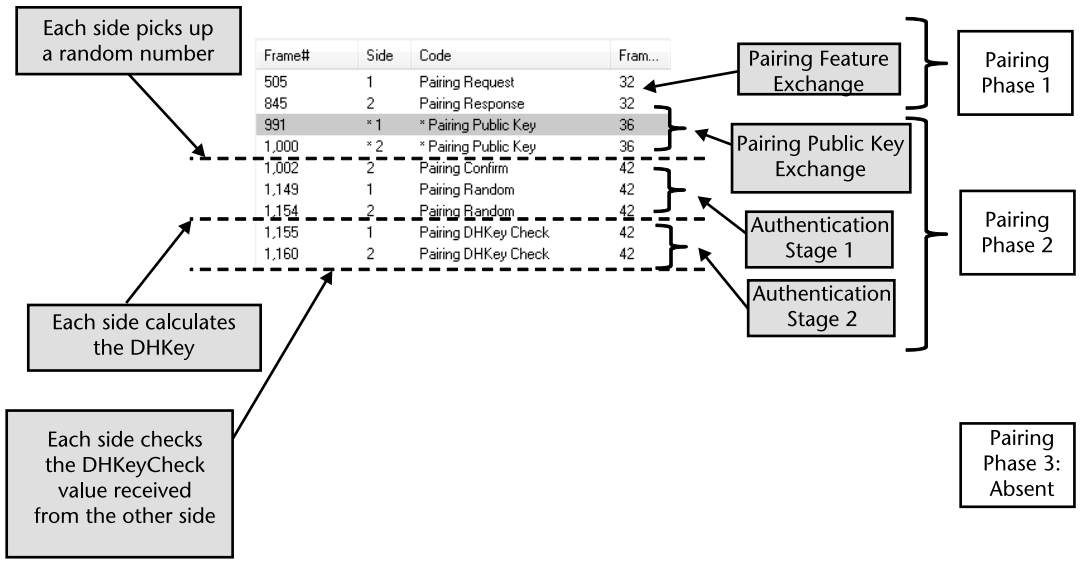

Figure 11.16 Example of Pairing Phase 2.

- The public key exchange happens when the devices exchange public keys. The Master sends its public key to the Slave, followed by the Slave sending its public key to the Master.

2. After public key exchange, each side generates a random number that will be used in subsequent stages.
3. Authentication Stage 1
   - As seen in Frame #1002, the Slave calculates a confirmation value and sends it to the Master using pairing confirm. This is generated using the

previously-generated random numbers and the public key received from the Master.
   - The Master sends the random number that it generated to the Slave in Frame #1149, and the Slave sends its random number to the Master in Frame #1154.
4. Each side calculates the DHKey
5. Authentication Stage 2
   - The Master sends the DHKey check value to the Slave in Frame #1155, and the Slave sends the DHKey check value to the Master in Frame #1160
6. Each side checks the DHKey check value that it received from the remote side. If is the DHKey check values are correct, an LE secure connection is established.

11.10.2.3 Pairing Phase 3

The pairing Phase 3, transport-specific key distribution does not happen in this particular sequence. This is because both master and slave indicated during Phase 1 that no keys will be distributed in Phase 3.

11.11 Summary

The security manager is responsible for carrying out security related procedures like pairing, authentication, and encryption. The security manager architecture in LE introduces several enhancements as compared to BR/EDR. This includes moving the security manager block into the host in the controller to keep the controller requirements to a minimum as well as using asymmetrical architecture to perform less processing in one device at the cost of performing more processing on the peer device. This is done because generally in LE communication one device (like a mobile phone) has abundant resources while the other device (like a sensor) has very restricted resources and the device with limited resources tries to conserve battery power by doing as little processing as possible.

The pairing process is one of the important processes defined by security manager. It uses a three phase process. The first phase broadly includes exchange of capabilities, the second phase includes exchanging of short term keys, and the third phase involves the exchange of actual keys that will be used in subsequent procedures.

Bibliography

Bluetooth Core Specification 4.0 http://www.bluetooth.org.
NIST Publication FIPS-197 (http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf).
IETF RFC 4493 (https://tools.ietf.org/html/rfc4493).

CHAPTER 12

Attribute Protocol (ATT)

12.1 Introduction

The attribute protocol provides a mechanism for discovering attributes of a remote device, reading the attributes and writing the attributes. As shown in Figure 12.1 ATT sits above L2CAP layer and uses L2CAP as the transport mechanism for transferring data. In turn, ATT provides services to the Generic Attribute Profile.

Attribute protocol follow a client server model. The server exposes a set of attributes to the client. The client can discover, read and write those attributes. The server can also notify or indicate the client about any of the attributes.

A device can implement only a client role, only a server role, or both client and server roles. At any given time, only one server can be active on a device.

12.2 Attribute

An attribute is something that represents data. It could be thought of as just any data at any given time when the device is in any given state. It could be the position, size, mass, temperature, speed, or any other data that the device wants to share with other devices. ATT protocol is designed to push or pull that data to or from a remote device. Besides that ATT protocol also supports setting notifications and indications so that the remote devices can be alerted when that data changes.

Some examples of attributes are:

- The temperature provided by a thermometer.
- The unit in which the temperature is provided.
- The name of a device.
- The manufacturer name and model number of a device.

Besides containing the value of the data, an attribute has three properties associated with it:

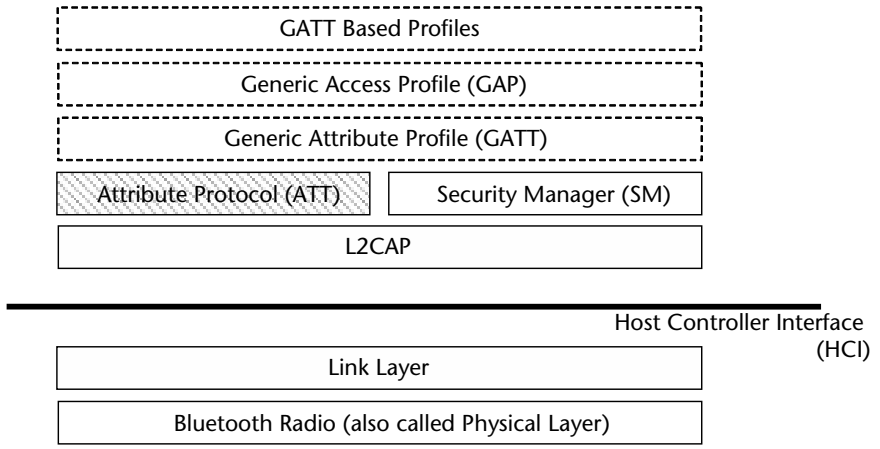

Figure 12.1 ATT protocol in LE protocol stack.

1. Attribute Type.
2. Attribute Handle.
3. Access Permissions.

The structure of an attribute is show in Figure 12.2.

The clients can discover the handles of the various attributes on the server and can read or write these attributes provided they have sufficient permissions to do so. The server can also inform the client of the value of the attribute at any particular time. The various operations that can be carried out between the client and the server will be explained in detail in further sections.

12.2.1 Attribute Type

The attribute type specifies what that particular attribute represents. This allows the client to understand the meaning of that particular attribute. Some examples of attribute type are:

- <<Primary Service>>
- <<Health Thermometer Service>>
- <<Manufacturer Name>>
- <<Serial Number>>

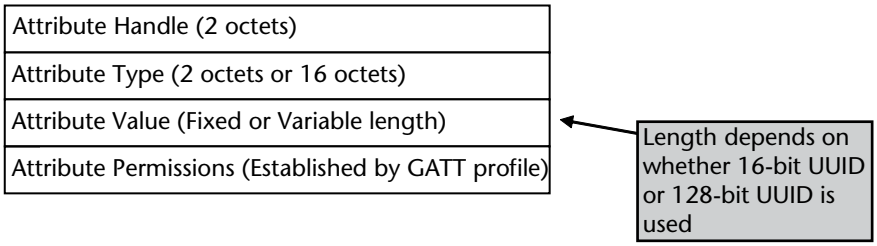

Figure 12.2 Attribute structure.

The attribute type is identified by a Universally Unique Identifier (UUID). A UUID is a 128-bit value which is considered to be unique over space and time. The implementation could either use the set of predefined UUIDs defined by the Bluetooth SIG or define its own UUIDs.

In general, a shorter form of UUIDs is used. This shorter form is 16-bit. The 16-bit UUIDs are assigned by the Bluetooth SIG and share the same namespace as the 16-bit SDP UUIDs. These are published on the Bluetooth Assigned Numbers page on the Bluetooth SIG website.

The 128-bit value can be derived from the 16-bit value by combining it with the Bluetooth_Base_UUID as follows:

128-bit UUID = 16-bit UUID * 2^{96} + Bluetooth_Base_UUID

Note: This can also be derived by replacing the xxxx in the following with the hexadecimal value of the 16-bit UUID 0000xxxx-0000-1000-8000-00805F9B34FB.

12.2.2 Attribute Handle

All attributes on the server are assigned a unique, non-zero attribute handle. This handle is used by the client in all operations with the server to identify the attribute.

It is allowed to dynamically add or remove attributes on the server as long as the new attributes are not assigned a handle which has already been used by any other attribute in the past (even if that attribute has been deleted). This ensures that clients always get a unique attribute handle.

Once an attribute has been assigned an attribute handle, it should not change over time. This ensures that the clients can keep accessing that attribute with the same handle.

The attributes on the server are ordered by the attribute handle.

An attribute handle of 0x0000 is reserved and an attribute handle of 0xFFFF is known as the maximum attribute handle.

12.2.3 Attribute Permissions

Each attribute has a set of permissions associated with it which determines the level of access that is permitted for that particular attribute. The attribute permissions are used by a server to determine whether a client is allowed to read or write an attribute value and whether Authentication or Authorization is required to access that particular attribute.

Attribute permissions are a combination of following four permissions:

1. Access Permissions: This could be set to one of the following:
   - Readable.
   - Writable.
   - Readable and Writable.
2. Encryption Permissions: This could be set to one of the following:
   - Encryption Required.
   - No Encryption Required.
3. Authentication Permissions: These are used by the server to determine if an authenticated physical link is required when a client attempts to access

that attribute or when the server has to send a notification or indication to client. This could be set to one of the following:
   - Authentication Required.
   - No Authentication required.
4. Authorization Permissions: These are used by the server to determine if a client needs to be authorized before accessing an attribute value. This could be set to one of the following:
   - Authorization Required.
   - No Authorization Required.

If the client does not have sufficient permissions an error code is returned. Some of the error codes are shown below:

- Read Not Permitted: The attribute cannot be read.
- Write Not Permitted: The attribute cannot be written.
- Insufficient Authentication: The attribute requires authentication before it can be read or written.
- Insufficient Authorization: The attribute requires authorization before it can be read or written.
- Insufficient Encryption Key Size: The Encryption Key Size used for encrypting this link is insufficient.
- Insufficient Encryption: The attribute requires encryption before it can be read or written.

12.2.4 Attribute Value

An attribute value is an octet array that contains the actual value of the attribute. The length of the attribute can be either fixed or variable:

- Fixed length—The length can be one octet, two octet or four octet.
- Variable Length—The attribute can be a variable length string.

To simplify things, ATT does not allow multiple attribute values to be transmitted in a single PDU. A PDU contains only one attribute value and if the attribute value is too long to transmit in a single PDU, it can be split across multiple PDUs. There are some exceptions to this though—when a client requests for multiple attributes to be read and the attributes have a fixed length, then the response can contain multiple attributes.

12.2.5 Control Point Attributes

The attributes that cannot be read, but can only be written, notified or indicated are called control point attributes. These are generally used to control the behavior of a device and therefore called control point attributes.

One example of Control Point Attribute is the Alert Level characteristic defined by the Immediate Alert Service. The remote devices can write the alert level into

this attribute (It can be one of “No Alert, “Mid Alert”, or “High Alert”). Once the alert level is written the device may take some specific action like flashing an LED or sounding an alarm. This characteristic can only be written but cannot be read.

12.2.6 Grouping of Attribute Handles

ATT allows a set of attributes to be grouped together. In this case, a specific attribute is placed at the beginning of the other attributes that form the group. The clients can request the first and the last handle associated with the group of attributes.

The groups are defined by the higher layer profile, GATT. GATT defines grouping of attributes for three attribute types:

- <<Primary Service>>
- <<Secondary Service>>
- <<Characteristic>>

Once the attributes are grouped together, operations like Read By Group Type can be done on those attributes.

These will be explained in detail in the next chapter.

12.2.7 Atomic Operations

At any point of time, multiple clients may be connected to a server. The server treats each operation from the client as an atomic operation which is not impacted by any other client initiating an operation at the same time.

For example if a client is writing a large amount of data by splitting it into chunks, then the server treats this as an atomic operation. The commands to do this, viz Prepare Write Request and Execute Write Request, will be explained in further detail later in this chapter. This operation is not impacted if another client starts doing another write operation in parallel.

12.3 Attribute Protocol

The attribute protocol defines how a device will discover, read and write the attributes of another device. The following two roles are defined.

1. Attribute Server—The attribute server exposes a set of attributes and their associated values to a peer device.
2. Attribute Client—The attribute client can discover, read and write the attributes on a server. In addition to this it can be indicated and notified by the server.

The attribute protocol supports data to be transferred both from the client to server and vice versa. The client can read or write the attributes of the server. In addition to this the server can also notify the client about any attribute.

The PDUs supported by ATT can be of following six method types:

1. Request—This PDU is sent by the client to the server and the server responds with a response.
2. Response—This PDU is sent by the server in response to a request from the client.
3. Command—This PDU is sent by the client to the server. No response is received for this PDU.
4. Notification—This PDU is sent by the server to the client. The client does not send anything back.
5. Indication—This PDU is sent by the server to the client and the client responds with a confirmation.
6. Confirmation—This PDU is sent by the client to confirm the receipt of an indication.

These are shown in Figure 12.3.

12.3.1 PDU Format

The format of Attribute protocol PDUs is shown in Figure 12.4.

The Opcode contains the following:

- Method—Identifies the method for which this PDU is sent.
- Command Flag—Indicates whether this is a command. There is no acknowledgement from the server for a command.

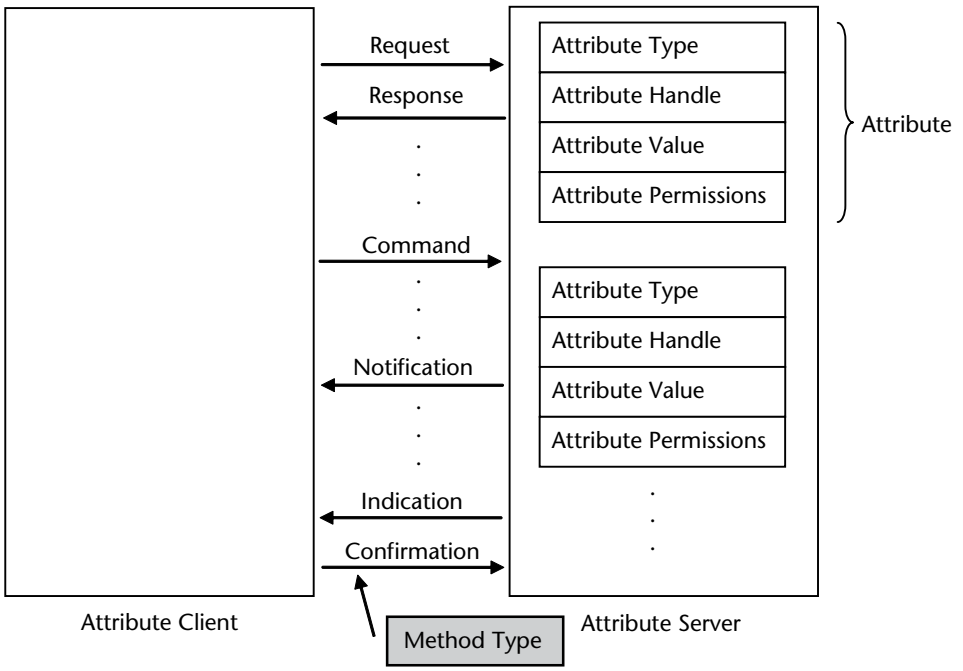

Figure 12.3 Attribute Client and Server.

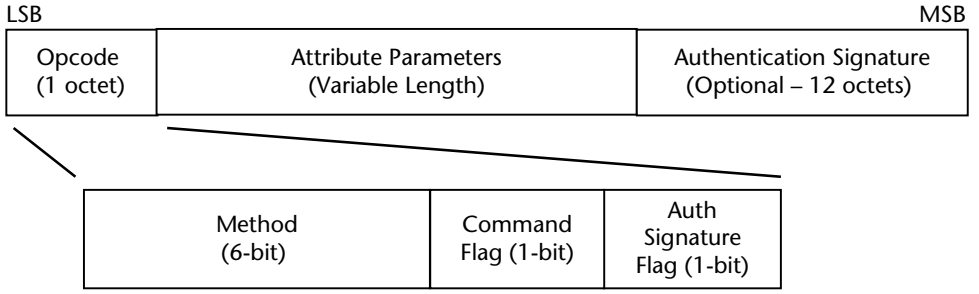

Figure 12.4 Format of attribute protocol PDU.

- Authentication Signature Flag—If this is 1, then an Authentication Signature is appended to the PDU. (Authentication Signature will be explained shortly.)

The Attribute Parameters contain the data as per the specified Method.

The Authentication Signature is an optional field and is used for signing the data that is present in this PDU. If this field is present, then it provides a signature for the Opcode and Attribute Parameters fields. This field may be used if the link is not encrypted but security is required.

12.3.2 Sequential Transactions

The Attribute protocol is a sequential request-response, indication-confirmation protocol.

- For commands and responses—The client sends one request to the server and then waits for the response before sending the next request.
- For indications and confirmations—The server sends one indication to the client and then waits for the confirmation from the client before sending the next indication.

This introduces a flow control mechanism for request-response and indication-response transactions. The flow control mechanism is valid only for request-response pairs and indication-confirmation pairs. So it is possible to have sequences like request-indication-response-confirmation or request-notification-response.

Flow control of each client and server is independent.

A request-response pair or an indication-confirmation pair is called a transaction. ATT defines a transaction timeout of 30 seconds. This means that if a request is sent from the client to the server and the response is not received within 30 seconds, then the transaction is considered to have timed out and the client can inform the higher layers which initiated the transaction about the failure.

For notifications and commands, there is no response from the other side. These can be sent at any time and there is no flow control. The notifications and commands that cannot be processed are just discarded. So these PDUs are considered to be unreliable.

One example of notifications is the intermediate temperature measurements sent by a server acting as a thermometer. If the client receives those notifications, then it displays the intermediate temperature value, otherwise those values just get discarded.

12.4 Methods

As mentioned previously, the PDUs supported by ATT can be of following six method types:

1. Request: Invokes a Response;
2. Response;
3. Command;
4. Notification;
5. Indication: Invokes a Confirmation;
6. Confirmation.

12.4.1 Request and Response Type Methods

The different PDUs for Request and Response type method are shown in Table 12.1. These PDUs are sent from the client to the server. The response that can be sent by the server for each request is also shown. Besides that response, an Error Response can also be sent from the server to the client.

12.4.1.1 Exchange MTU Request

The Exchange MTU Request is sent by the client to the server to:

- Inform the server of the client's maximum receive MTU size.
- Request the server to respond with the server's maximum MTU size.

Table 12.1 Methods of Request and Response Type

Request	*Response*
Exchange MTU Request	Exchange MTU Response
Find Information Request	Find Information Response
Find By Type Value Request	Find By Type Value Response
Read By Type Request	Read By Type Response
Read Request	Read Response
Read Blob Request	Read Blob Response
Read Multiple Request	Read Multiple Response
Read By Group Type Request	Read By Group Type Response
Write Request	Write Response
Prepare Write Request	Prepare Write Response
Execute Write Request	Execute Write Response

This request has the following parameter:

- Client Rx MTU: Client Receive MTU Size.

This request is sent only by the client to the server. It is sent only once during the connection.

The default value of ATT_MTU is 23 octets for LE. The exchange MTU request is used by the client if it wants to use an ATT_MTU bigger than this value. A bigger value will finally be used if both client and server support that bigger value.

The message sequence chart of the Exchange MTU Request is shown in Figure 12.5.

12.4.1.2 Exchange MTU Response

The Exchange MTU Response is sent by the server to the client in response to the Exchange MTU Request. It contains the server's receive MTU size

This response has the following parameter:

- Server Rx MTU: Server Receive MTU Size.

The message sequence chart of the Exchange MTU Response is shown in Figure 12.5.

After Exchange MTU request and response is complete, both the client and server set their MTU_SIZE (ATT_MTU) to the minimum of server and client receive MTU sizes. Both of them use the same size for all further requests and responses.

A practical example of Exchange MTU Request and Response is shown in Figure 12.19 towards the end of this chapter.

12.4.1.3 Find Information Request

The Find Information Request is sent by the client to the server to get a list of attribute handles and their types. The structure of an attribute was shown in Figure

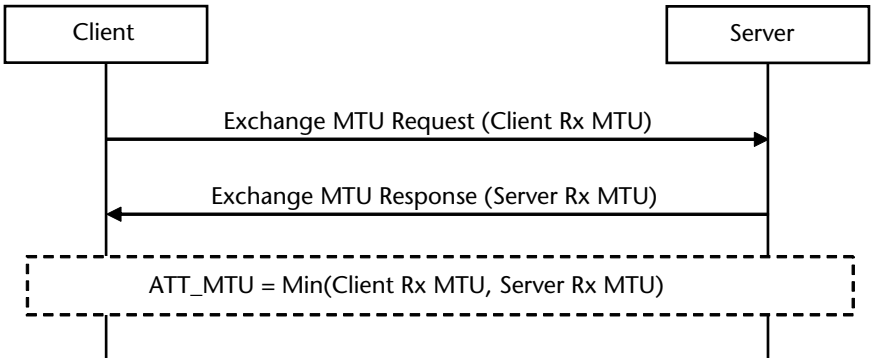

Figure 12.5 Exchange MTU Request and Response.

12.2. This request is used to get the first two fields (Attribute Handle, Attribute Type) of that structure for a range of attributes.

The range of attributes is specified by two parameters:

- Starting Handle: Starting handle of the range of handle numbers to search.
- Ending Handle: Ending handle of the range of handle numbers to search.

The message sequence chart of the Find Information Request is shown in Figure 12.6.

12.4.1.4 Find Information Response

The Find Information Response is sent by the server to the client in response to the Find Information Request. It contains the handle-UUID pairs (Attribute Handle, Attribute Type) for the range of Attribute Handles that was provided by the client.

The parameters of this response are:

- Format—Specifies the format of information data—Whether the handles contain 16-bit UUID or 128-bit UUID.
- Attribute Data List—handle-value pairs containing Attribute Handle and Attribute Value.

The UUID can be either 16-bit or 128-bit. The type that is returned is contained in the Format field.

Since the range of attributes can be long, this response can be split across multiple response packets but the handle-UUID pairs are not split across packets. The handle-UUID pairs are sent in ascending order of the handle value.

The message sequence chart of the Find Information Response is shown in Figure 12.6.

12.4.1.5 Find By Type Value Request

The Find By Type Value Request is sent by the client to the server to get a list of attributes which contain a particular Attribute Type and Attribute Value.

The parameters of this request are:

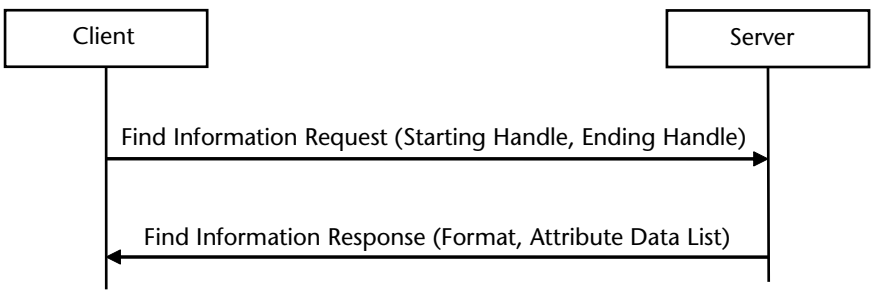

Figure 12.6 Find information Request and Response.

- Starting Handle: Starting handle of the range of handle numbers to search.
- Ending Handle: Ending handle of the range of handle numbers to search.
- Attribute Type: UUID that specifies the Attribute Type to find.
- Attribute Value: Attribute Value to find.

The UUID can be specified as only 16-bit.

The message sequence chart of the Find By Type Value Request is shown in Figure 12.7.

12.4.1.6 Find By Type Value Response

The Find By Type Value Response is sent by the server to the client in response to the Find By Type Value Request. It contains the list of handles that correspond to attributes which exactly match the Attribute Type and Attribute Value provided by the client in the Find By Type Value Request.

The parameters of this response are:

- Handle Information List: List of handles that match the Attribute Type and Attribute Value.

The message sequence chart of the Find By Type Value Response is shown in Figure 12.7.

12.4.1.7 Read By Type Request

The Read By Type Request is sent by the client to the server to get a list of attributes of a particular Attribute Type.

The parameters of this request are:

- Starting Handle: Starting handle of the range of handle numbers to search.
- Ending Handle: Ending handle of the range of handle numbers to search.
- Attribute Type: UUID that specifies the Attribute Type to find.

The UUID can be specified as either 16-bit or 128-bit.

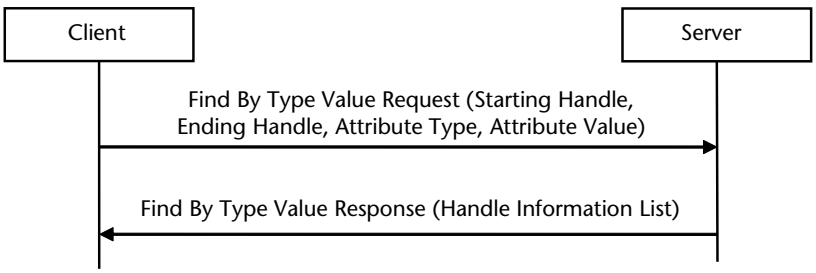

Figure 12.7 Find by type value Request and Response.

The message sequence chart of the Read By Type Request is shown in Figure 12.8.

12.4.1.8 Read By Type Response

The Read By Type Response is sent by the server to the client in response to the Read By Type Request. It contains the list of handle-value pairs (Attribute Handle, Attribute Value) that correspond to attributes which exactly match the Attribute Type provided by the client in the Read By Type Request.

The parameters of this response are:

- Length: The size of each attribute handle-value pair.
- Attribute Data List: handle-value pairs containing Attribute Handle and Attribute Value.

Since the range of attributes can be long, this response can be split across multiple response packets but the handle-value pairs are not split across packets. The handle-value pairs are sent in ascending order of the handle value.

The message sequence chart of the Read By Type Response is shown in Figure 12.8.

12.4.1.9 Read Request

The Read Request is sent by the client to the server to read the Attribute Value of an attribute.

This request has the following parameter:

- Attribute Handle: The handle of the attribute that is to be read.

The message sequence chart of the Read Request is shown in Figure 12.9.

12.4.1.10 Read Response

The Read Response is sent by the server to the client in response to the Read Request. It contains the Attribute Value corresponding to the attribute specified by the Attribute Handle provided by the client in the Read Request.

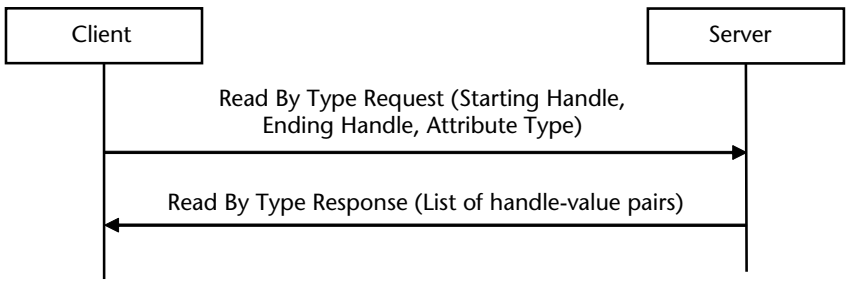

Figure 12.8 Read by type Request and Response.

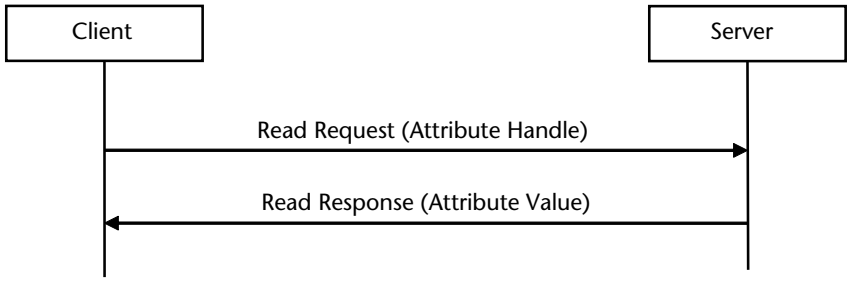

Figure 12.9 Read Request and Response.

The parameters of this response are:

- Attribute Value: Attribute Value corresponding to the Attribute Handle.

If the Attribute Value is too big to fit in the ATT_MTU -1 size, then only the ATT_MTU-1 octets are returned and the remaining octets can be read by the Read Blob request.

The message sequence chart of the Read Response is shown in Figure 12.9.

12.4.1.11 Read Blob Request

The Read Blob Request is sent by the client to the server to read the Attribute Value of an attribute starting from a particular offset. This request is useful in scenarios where the Attribute Value is long and cannot fit ATT_MTU–1 octets. In that case the Read Response reads only the first ATT_MTU-1 octets and the remaining octets can be read using the Read Blob Request

The parameters of this request are:

- Attribute Handle: The handle of the attribute that is to be read.
- Value Offset: The offset from where the part of Attribute Value should be read. The Value Offset parameter is based from zero. This means that the first octet has an offset of zero.

The message sequence chart of the Read Blob Request is shown in Figure 12.10.

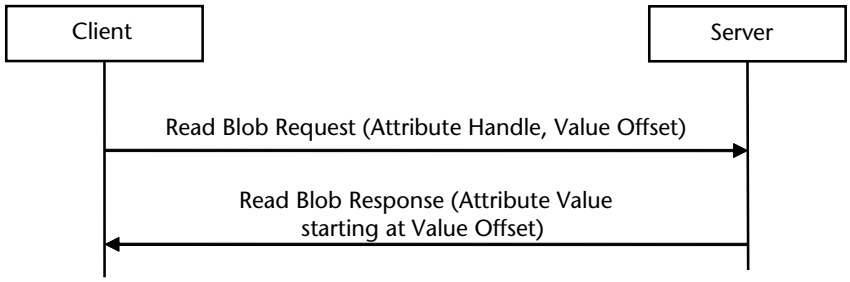

Figure 12.10 Read blob Request and Response.

12.4.1.12 Read Blob Response

The Read Blob Response is sent by the server to the client in response to the Read Blob Request. It contains a part of the Attribute Value corresponding to the attribute specified by the Attribute Handle provided by the client in the Read Blob request. The starting octet of the part of the Attribute Value corresponds to the Value Offset that was provided in the Read Blob Request.

The parameter of this response is:

- Part Attribute Value: octets of the Attribute Value starting at Value Offset.

If the part of the Attribute Value that is to be returned is too big to fit in the ATT_MTU -1 size, then only the ATT_MTU-1 octets are returned and the remaining octets can be read by another Read Blob Request by adjusting the offset.

The message sequence chart of the Read Blob Response is shown in Figure 12.10.

12.4.1.13 Read Multiple Request

The Read Multiple Request is sent by the client to the server to read the Attribute Values of two or more attributes. This request is useful in scenarios where multiple attributes are to be read using one single request.

This request has the following parameter:

- Set of Handles: A set of two or more attribute handles.

The message sequence chart of the Read Multiple Request is shown in Figure 12.11.

12.4.1.14 Read Multiple Response

The Read Multiple Response is sent by the server to the client in response to the Read Multiple Request. It contains a set of Attribute Values corresponding to the attributes specified by the Attribute Handles provided by the client in the Read Multiple Request. The set of values are just concatenated in the same order in which the Attribute Handles were provided in the Read Multiple Request.

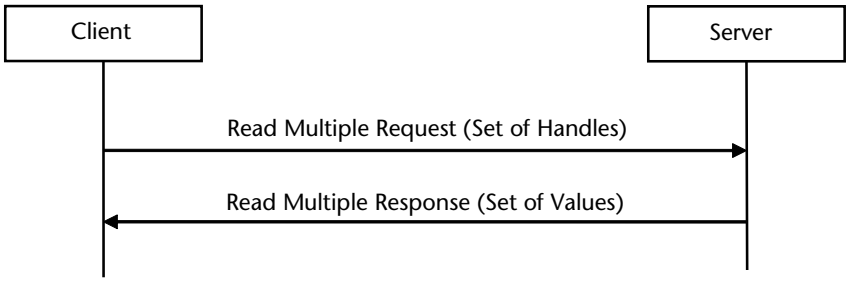

Figure 12.11 Read multiple Request and Response.

The parameter of this response is:

- Set of Values: A set of two or more values corresponding to the Set of Handles.

The message sequence chart of the Read Multiple Response is shown in Figure 12.11.

12.4.1.15 Read By Group Type Request

The Read By Group Type Request is sent by the client to the server to read the Attribute Values of a particular group of services. This request is similar to the Read By Type Request. The only difference is that it specifies an Attribute Group Type instead of an Attribute Type.

The GATT Profile provides support for grouping attributes into various types. This request is used to read the attributes of the <<Primary Service>> and <<Secondary Service>> groups. This will be explained in detail in the next chapter.

The parameters of this request are:

- Starting Handle: Starting handle of the range of handle numbers to search.
- Ending Handle: Ending handle of the range of handle numbers to search.
- Attribute Group Type: UUID that specifies the Attribute Group Type to read.

The message sequence chart of the Read By Group Type Request is shown in Figure 12.12.

A practical example of Read By Group Type Request that is used to retrieve the Primary Services from a server is shown in Figure 12.20 towards the end of this chapter.

12.4.1.16 Read By Group Type Response

The Read By Group Type Response is sent by the server to the client in response to the Read By Group Type Request. It contains the handles and values of the attributes with the specified Attribute Group Type.

The parameters of this response are:

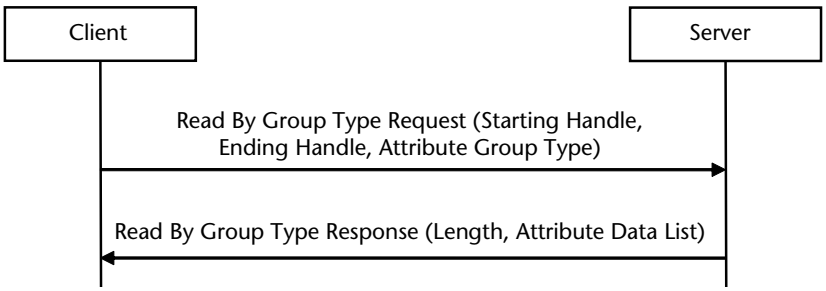

Figure 12.12 Read by group type Request and Response.

- Length: Size of each attribute data.
- Attribute Data List: This contains the Attribute Handle, the End Group Handle and the Attribute Value.

The End Group Handle is the handle of the last attribute within a group.

The message sequence chart of the Read By Group Type Response is shown in Figure 12.12.

A practical example of Read By Group Type Response that is used to retrieve the Primary Services from a server is shown in Figure 12.21 towards the end of this chapter.

12.4.1.17 Write Request

The Write Request is sent by the client to the server to request the server to write the value of an attribute and acknowledge that it has written it by a Write Response. If there is any error while writing (For example insufficient permissions), then an Error response is sent by the server to the client.

The parameters of this request are:

- Attribute Handle: The handle of the attribute.
- Attribute Value: The value to be written

The message sequence chart of the Write Request is shown in Figure 12.13.

12.4.1.18 Write Response

The Write Response is sent by the server to the client in response to the Write Request. It confirms that the Attribute Value has been successfully written.

This response does not have any parameters.

The message sequence chart of the Write Response is shown in Figure 12.13.

12.4.1.19 Prepare Write Request

The Prepare Write Request is sent by the client to the server to prepare to write the value of an attribute. This is useful in scenarios where the Attribute Value is long

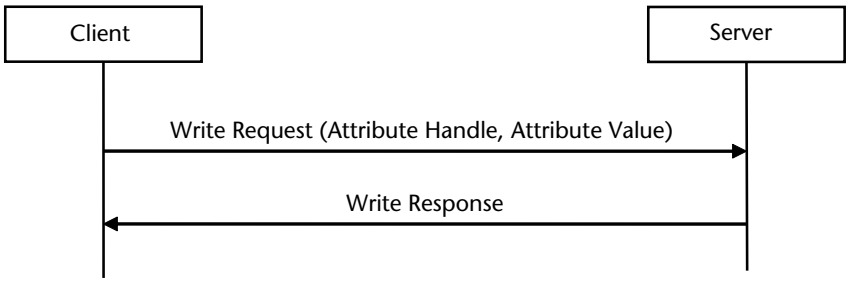

Figure 12.13 Write Request and Response.

and cannot fit one PDU. In this case the client sends multiple Prepare Write Request PDUs to the server and the server queues those requests. Once all Prepare Write Request PDUs have been sent, the client sends an Execute Write Request PDU. After the Execute Write Request PDU is received, the server writes all the queued octets of the Attribute Value that were received in the Prepare Write Request.

The new Attribute Values take effect only after the Execute Write Request has been executed on the server.

If a server is serving multiple clients, then it queues the Prepare Write Requests from each of the clients separately. The execution of queue from one client does not affect the queue from another client.

The parameters of this request are:

- Attribute Handle: The handle of the attribute.
- Value Offset: The offset of the first octet to be written.
- Part Attribute Value: The octets of the Attribute Value to be written starting at Value Offset.

The message sequence chart of the Prepare Write Request is shown in Figure 12.14.

12.4.1.20 Prepare Write Response

The Prepare Write Response is sent by the server to the client in response to the Prepare Write Request. It is used to acknowledge that the value has been received by the server and placed in the write queue.

The parameters of this request are the same as those of the Prepare Write Request. These are:

- Attribute Handle: Same as the one provided in Prepare Write request
- Value Offset: Same as the one provided in the Prepare Write request.
- Part Attribute Value: Same as the one provided in the Prepare Write request.

The message sequence chart of the Prepare Write Response is shown in Figure 12.14.

12.4.1.21 Execute Write Request

The Execute Write Request is sent by the client to the server to either execute the write or cancel the write of all prepared values that were inserted in the write queue of the server using Prepare Write Requests.

If a server is serving multiple clients, then it queues the Prepare Write Requests from each of the clients separately. The execution of queue from one client does not affect the queue from another client.

This request has the following parameter:

- Flags – 0x00: Cancel all prepared writes. 0x01: Write all pending prepared values.

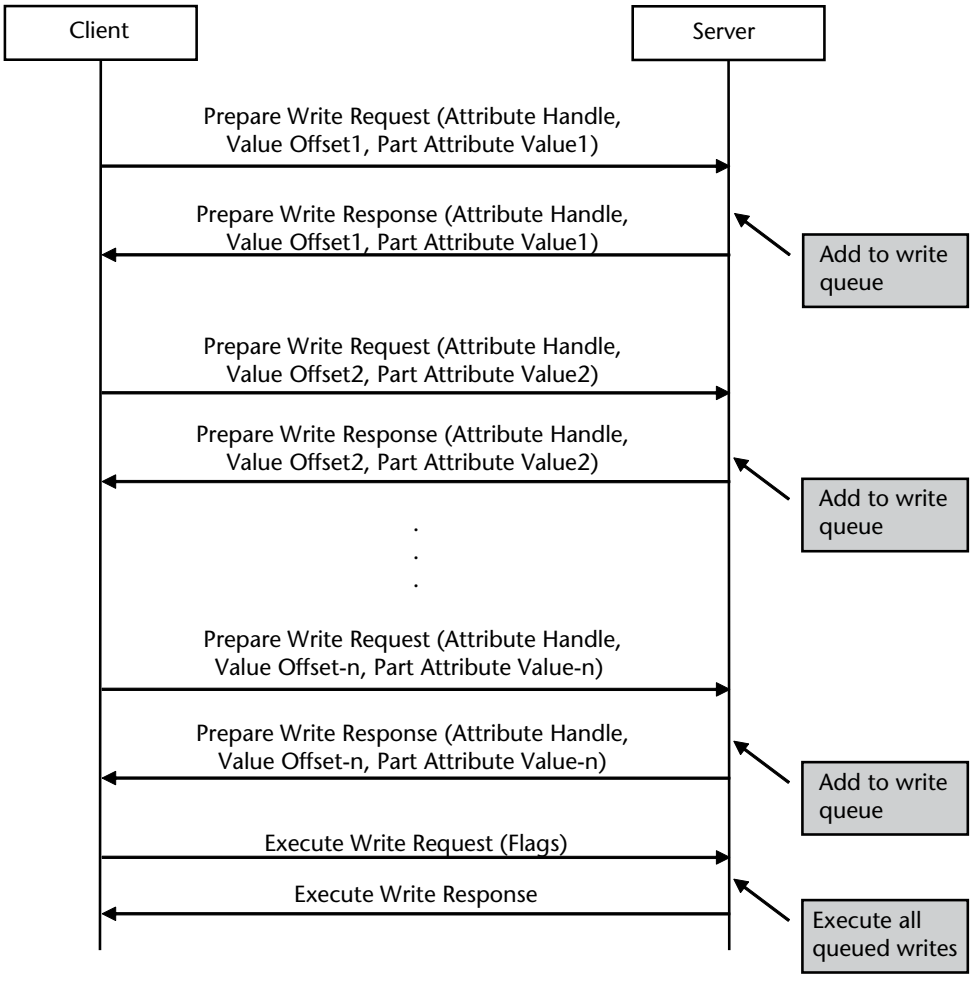

Figure 12.14 Prepare write Request and Response, execute write Request and Response.

If the Flags parameter contains 0x01, then all pending writes that were queued earlier are written in the same order in which they were queued.

The message sequence chart of the Execute Write Request is shown in Figure 12.14.

12.4.1.22 Execute Write Response

The Execute Write Response is sent by the server to the client in response to the Execute Write Request. It is used to confirm that all the values that were earlier queued have been successfully written.

This response does not have any parameters.

The message sequence chart of the Execute Write Response is shown in Figure 12.14.

12.4.2 Command Type Methods

The different PDUs for Command type are shown in Table 12.2. These PDUs are sent from the client to the server. There is no response or Error Response from the server in response to these PDUs.

12.4.2.1 Write Command

The Write Command is sent by the client to the server to request the server to write the value of an attribute. Generally this command is used for control-point attributes. The server does not acknowledge that it has written the value. Even if there is an error, the server does not send an Error Response.

The parameters of this command are:

- Attribute Handle: The handle of the attribute.
- Attribute Value: The value to be written.

The message sequence chart of the Write Command is shown in Figure 12.15.

12.4.2.2 Signed Write Command

The Signed Write Command is sent by the client to the server to request the server to write the value of an attribute after including an authentication signature with the data PDU. Generally this command is used for control-point attributes. The server does not acknowledge that it has written the value. Even if there is an error (like authentication signature verification fails), the server does not send an Error Response.

The parameters of this command are:

- Attribute Handle: The handle of the attribute.
- Attribute Value: The value to be written.
- Authentication Signature: 12-octet authentication signature.

Table 12.2 Methods of Command Type

Write Command
Signed Write Command

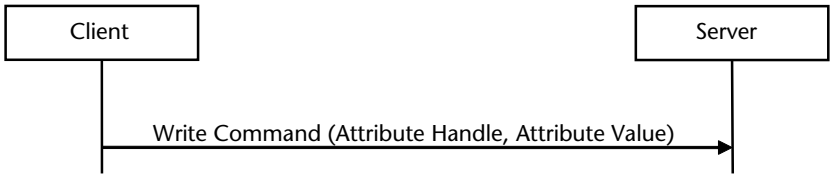

Figure 12.15 Write Command.

The Authentication Signature provides the feature of data signing. The server verifies the authentication signature to check the authenticity of the client before actually writing that value.

The message sequence chart of the Signed Write Command is shown in Figure 12.16.

12.4.3 Notification Type Methods

The different PDUs for Notification type are shown in Table 12.3. These PDUs are sent from the server to the client. There is no response from the client for these PDUs.

12.4.3.1 Handle Value Notification

The Handle Value Notification is sent by the server to the client to provide information about an attribute at any time.

The parameters of this notification are:

- Attribute Handle: The handle of the attribute.
- Attribute Value: The current value of the attribute.

The message sequence chart of the Handle Value Notification is shown in Figure 12.17.

12.4.4 Indication and Confirmation Type Methods

The different PDUs for Indication and Confirmation type are shown in Table 12.4. The Indication PDUs are sent from the server to the client. The client responds with a Confirmation PDU.

12.4.4.1 Handle Value Indication

The Handle Value Indication is sent by the server to the client to provide information about an attribute at any time.

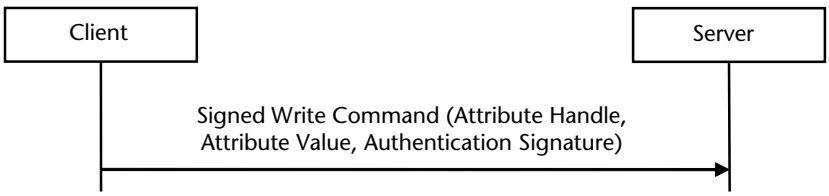

Figure 12.16 Signed write command.

Table 12.3 Methods of Notification Type

Handle Value Notification

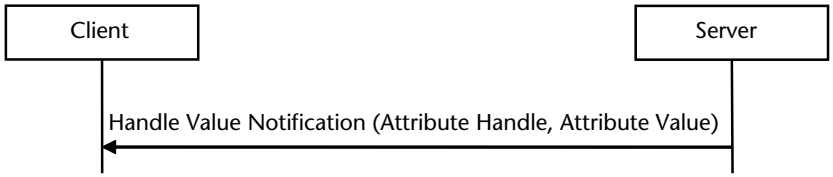

Figure 12.17 Handle value notification.

Table 12.4 Methods of Indication and Confirmation Type

Indication	*Confirmation*
Handle Value Indication	Handle Value Confirmation

The only difference from Handle Value Notification is that in this case the client responds with a Handle Value Confirmation. In case the value of the attribute is long, the client can use a Read Blob Request to get the entire value of the attribute after receiving the notification

The parameters of this indication are:

- Attribute Handle: The handle of the attribute.
- Attribute Value: The current value of the attribute.

The message sequence chart of the Handle Value Notification is shown in Figure 12.18.

12.4.4.2 Handle Value Confirmation

The Handle Value Confirmation is sent by the client to the server in response of a Handle Value Indication to confirm that it has received the Handle Value Indication.

This confirmation does not have any parameters.

The message sequence chart of the Handle Value Confirmation is shown in Figure 12.18.

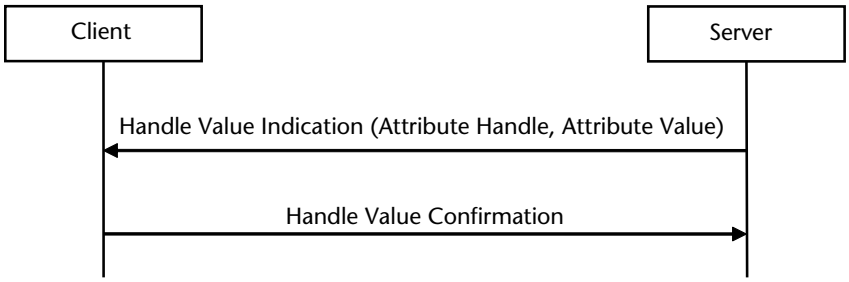

Figure 12.18 Handle value indication and confirmation.

12.5 Practical Examples

12.5.1 Exchange MTU

The air log captures of Exchange MTU Request and Response are shown in Figure 12.19. It shows that the client sends an Rx MTU size of 525 octets while the server responds with the Server Rx MTU of 23 octets. After these two PDUs are exchanged, both client and server will set the MTU to the minimum of the two values. This means that both the client and the server will set the MTU size to 23 octets and will not be transmitting any packets with data more than 23 octets.

12.5.2 Reading Primary Services of a Device

The list of primary devices supported by a remote device can be read by using the Read By Group Type Request with the UUID of <<Primary Service>>.

This is done by the client first sending the Read By Group Type Request to the server with the following parameters.

- Starting Attribute Handle = 1
- Ending Attribute Handle = 65535
- Attribute Group Type = Primary Service

This is shown in Figure 12.20.

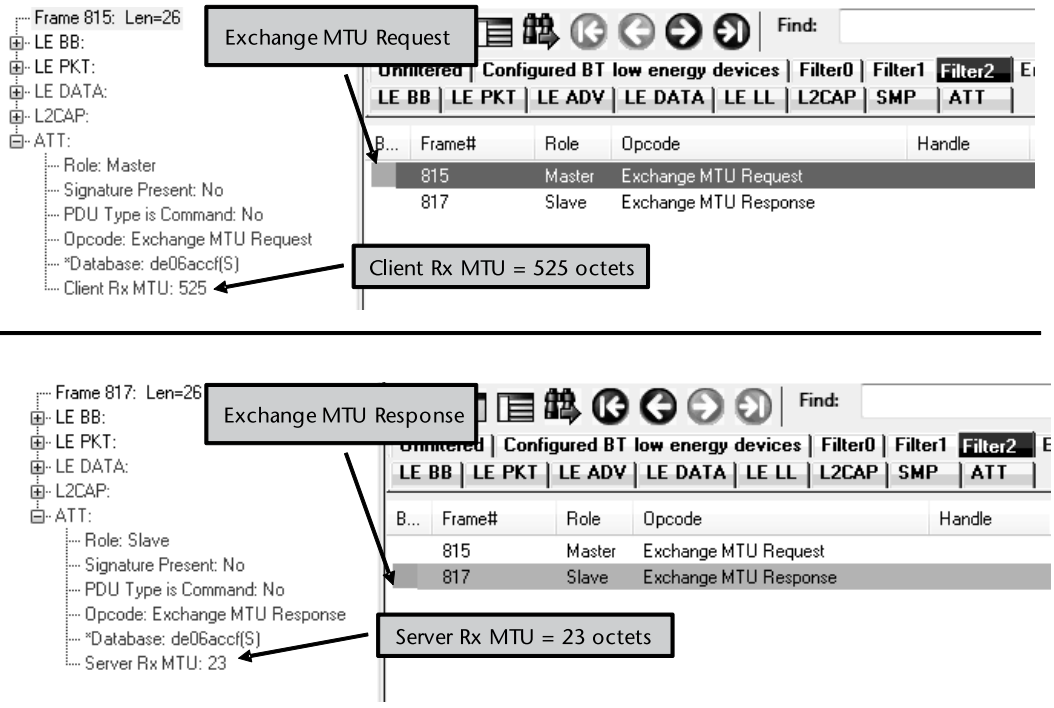

Figure 12.19 Example of exchange MTU Request and exchange MTU Response.

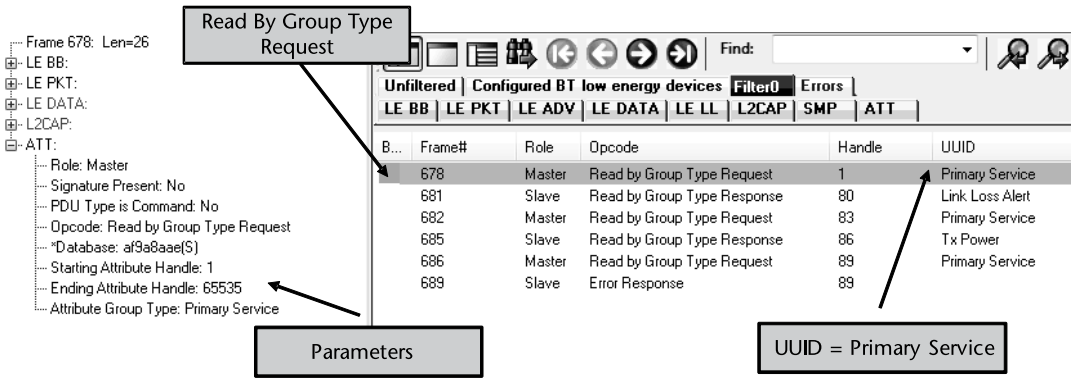

Figure 12.20 First read by group type Request for getting primary services.

The server responds to this request with a Read By Group Type Response which contains the Attribute Data List. In this example, the server returns the length as 6 octets for each handle-value pair and three service declarations in the response:

- Length: The size of each attribute handle-value pair = 6 octets.
- Attribute Data List: handle-value pairs containing Attribute Handle and Attribute Value.

1. Generic Access Profile with Starting Attribute Handle 0x01 and Ending Attribute Handle 0x07.
2. Generic Attribute Profile with Starting Attribute Handle 0x16 and Ending Attribute Handle 0x19.
3. Link Loss Alert with Starting Attribute Handle 0x80 and Ending Attribute Handle 0x82.

This is shown in Figure 12.21. It may be noted that the last handle that is returned is 0x82. This indicates to the client that if it wants to retrieve further services, then it should start with the next higher attribute handle, i.e., 0x83.

The client then continues to read more primary services. This time it sends the Starting Attribute Handle as 0x83 and all other parameters same as the previous request. This is shown in Figure 12.22.

The client responds with the set of services contained between the Attribute Handles 0x83 and 0x65535. This includes:

- Length: The size of each attribute handle-value pair = 6 octets.
- Attribute Data List: handle-value pairs containing Attribute Handle and Attribute Value.
    1. Immediate Alert Service with Starting Attribute Handle 0x83 and Ending Attribute Handle 0x85.
    2. Tx Power Service with Starting Attribute Handle 0x86 and Ending Attribute Handle 0x88.

This is shown in Figure 12.23.

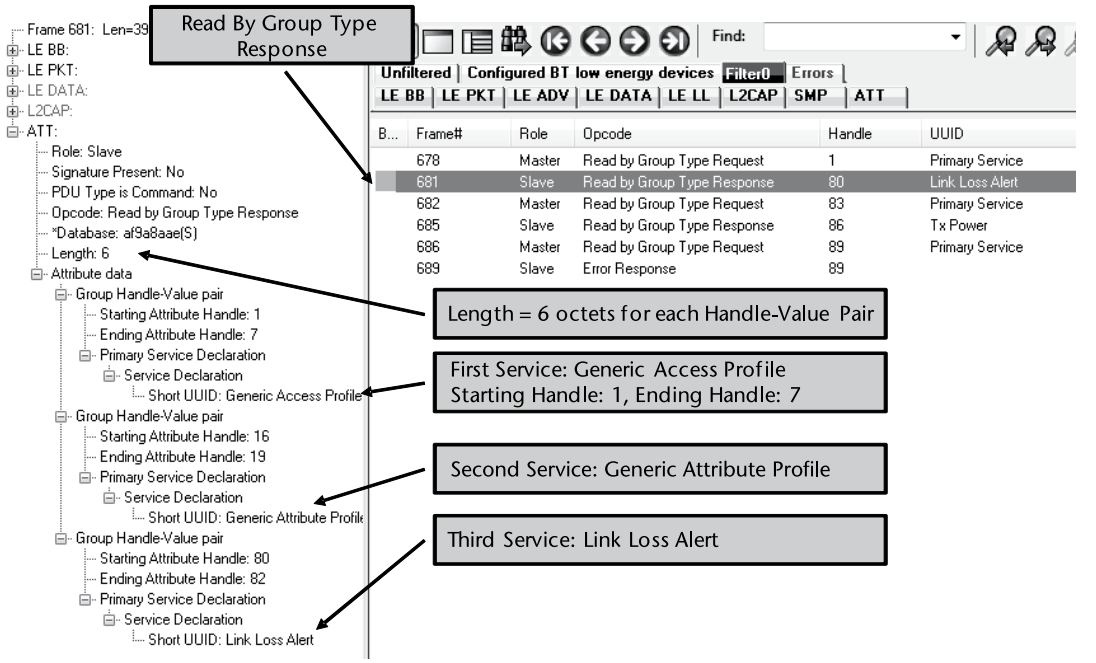

Figure 12.21 First read by group type Response providing the primary services.

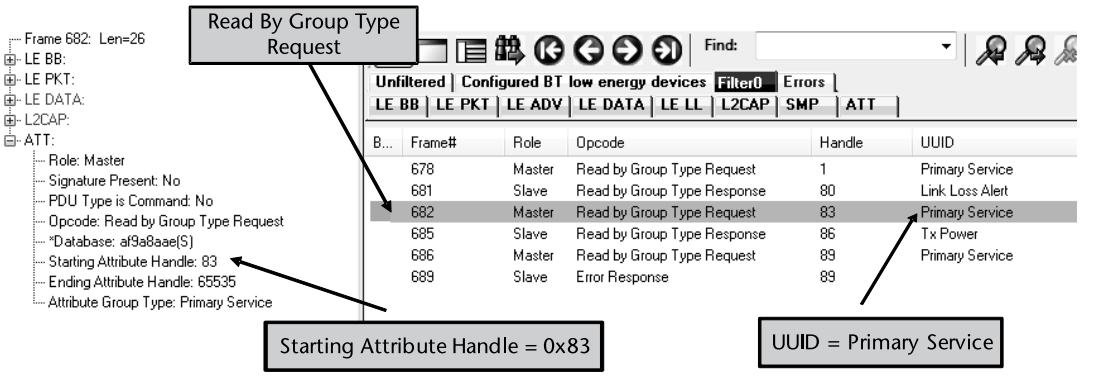

Figure 12.22 Second read by group type Request.

This time the client receives the last attribute handle as 0x88. So it sends a third Read By Attribute Type Request, this time with the Starting Handle 0x89 to retrieve the next set of primary services.

This time the server does not have any more Primary services to report between the specified Attribute Handles. So it responds with an Error Response. This is shown in Figure 12.24.

This indicates to the client that it has received all the attributes that it had requested.

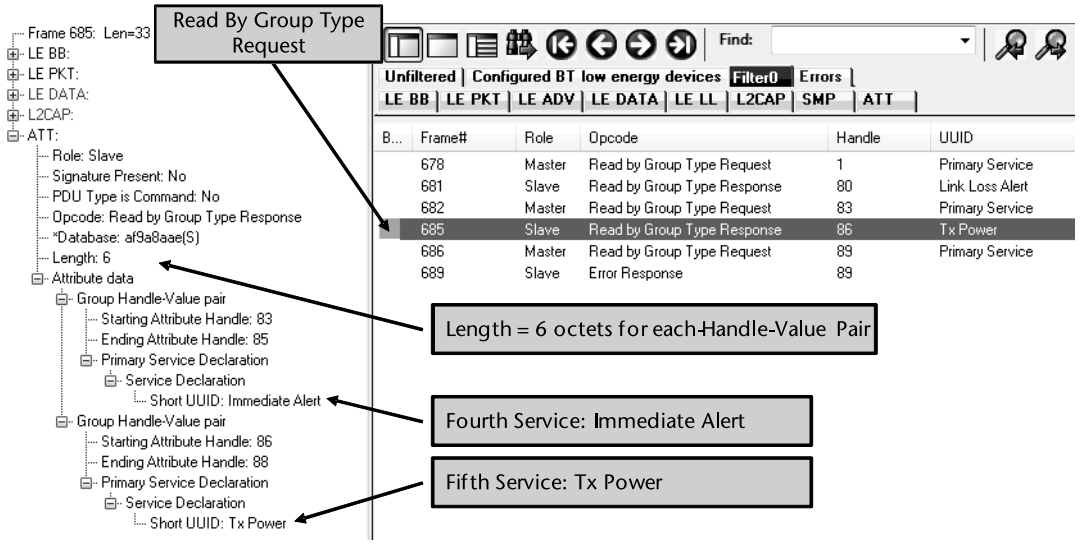

Figure 12.23 Second read by attribute type Response.

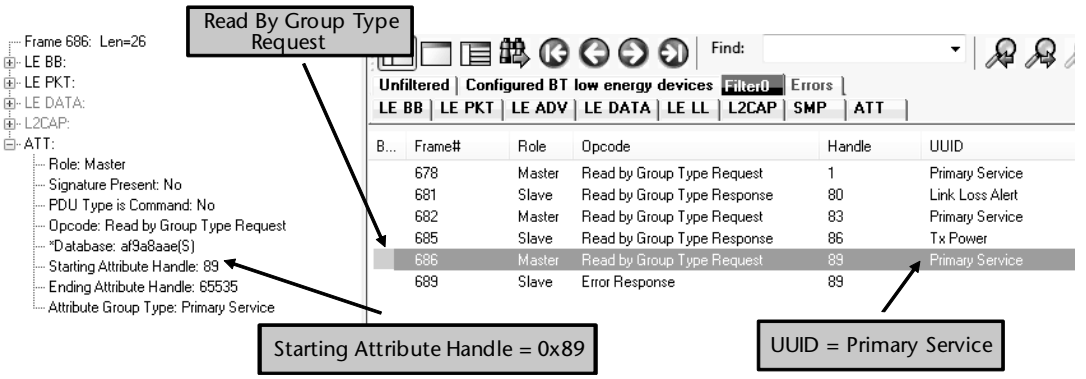

Figure 12.24 Third read by attribute Type Request.

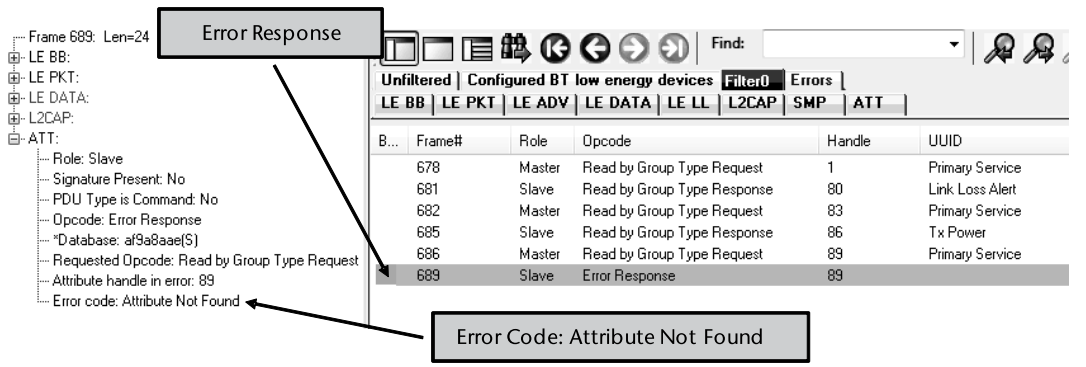

Figure 12.25 Error Response from the server indicating no further attributes.

12.6 Summary

The Attribute Protocol provides a very simple mechanism to exchange information between devices in the form of attributes. An attribute could be any data that a device decides to share with other devices. Besides the data, the attribute also contains additional information about the data like type of data, access permissions, etc.

ATT follows a client-server model. To keep the protocol simple, only sequential transactions are supported. This means that the next transaction is not initiated until a previous transaction is completed. In general the amount of data transferred in LE is quite small, typically in the range of 10–20 octets so that it can be fit in a single LE ACL data packet. Still ATT provides features to read and write longer amount of data by breaking that into smaller prepare transactions followed by an execute transaction.

The services of ATT are used by the Generic Attribute Profile which defines a hierarchy of services and characteristics using these attributes. This will be explained in detail in the next chapter.

Bibliography

Bluetooth Core Specification 4.0 http://www.bluetooth.org.

Assigned Numbers Specification: https://www.bluetooth.org/Technical/AssignedNumbers/home.htm.

CHAPTER 13

Generic Attribute Profile (GATT)

13.1 Introduction

In the previous chapter, the ATT protocol described the structure of an attribute and the mechanisms of accessing those attributes. The Generic Attribute Profile (GATT) defines a framework of services and characteristics using these attributes as building blocks. It also defines how a device will discover, read, write, notify, and indicate the characteristics. GATT also defines the mechanisms for configuring the broadcast of attributes. The position of GATT in LE protocol stack is shown in Figure 13.1.

The ATT protocol defines a flat set of attributes and mechanisms to access those attributes. ATT protocol does not define any hierarchy of the attributes. GATT allows the server to define a hierarchy so that the attributes are grouped into primary and secondary services and these services can include characteristics. Each of these characteristics has its own set of permissions which can be defined by GATT or a higher layer entity.

The GATT and ATT can be used on both BR/EDR and LE transport. It is mandatory to implement GATT and ATT for LE since these provide the basic capability to discover services of a remote device (similar to SDP in the case of BR/EDR). It is optional to implement GATT and ATT on the BR/EDR transport.

The GATT profile defines the client and server roles similar to the SDP profile. A device may act as a server, a client, or both. In general the LE-only devices (like sensors) are quite simple and only implement the server roles with services and characteristics needed to fulfill the application requirements.

The GATT profile is defined in such a manner that most of the complexity is encapsulated within this profile. The profiles that are on top of GATT are very simple.

The symbols << >> are used to indicate a Bluetooth SIG defined UUID. For example <<Characteristic>> is used to indicate a Bluetooth SIG defined UUID for Characteristic. It is a 128-bit unique value. In general, a shorter 16-bit equivalent of this value is used.

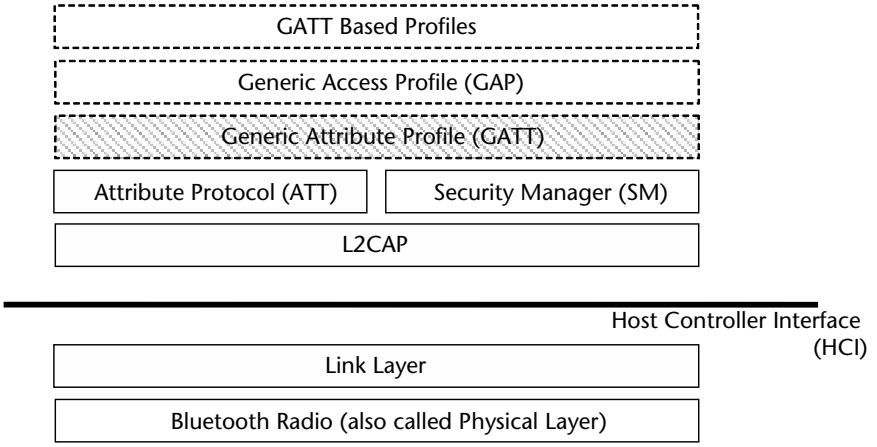

Figure 13.1 GATT in LE protocol stack.

13.1.1 Profile Dependencies

Similar to BR/EDR profiles, the dependencies of GATT based profiles are shown in Figure 13.2. A device can support one or more profiles at the same time.

Generic Access Profile (GAP) and Generic Attribute Profile (GATT) are mandatory for all devices that support LE. Devices may implement more profiles depending on the requirements of the application. The dependencies amongst profiles are depicted in Figure 13.2. One profile is dependent on another profile if it uses parts of that profile. A dependent profile is shown in an inner box and the outer box indicates profiles on which it is directly or indirectly dependent. GAP and GATT are shown in the outermost box since all other profiles are dependent on it.

GATT based profile architecture is much simpler than the architecture of BR/EDR profiles. It has only two layers. The outermost layer is GAP and GATT. All other profiles are located inside it at the same level. There is no further layering between the profiles (this is another step towards simplicity in LE).

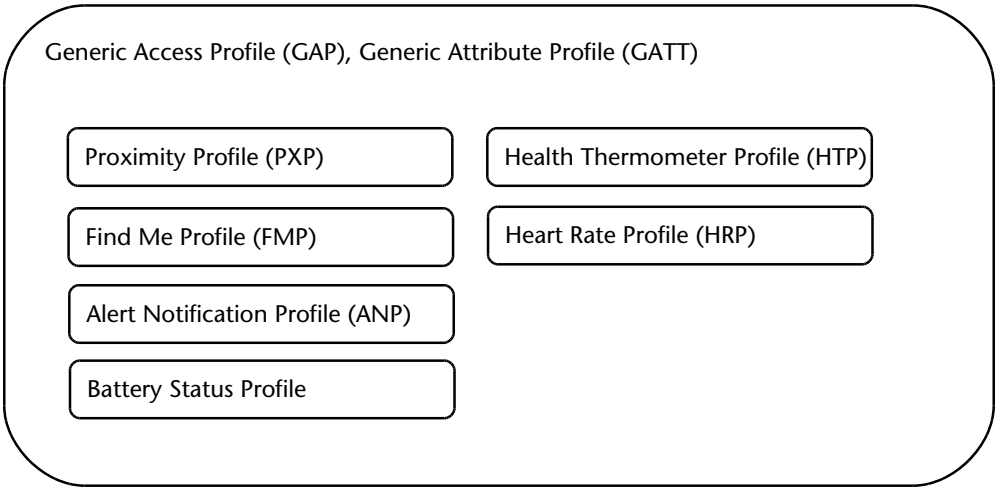

Figure 13.2 GATT-based profile dependencies.

13.1.2 GATT-Based Profile Architecture

The Bluetooth Core 4.0 specification introduced a very simple GATT based profile architecture. It is mandatory for LE devices and optional for BR/EDR devices.

In this architecture, the bulk of the profile complexity is embedded in GATT. The profiles on top are very simple and only needed to implement the functionality particular to devices of that particular category. For example, the health thermometer profile only has to implement the functionality specific to thermometer devices, like characteristics for providing the temperature. The remaining functionality like discovering the services, reading or writing attributes, or setting an indication are provided by GATT.

Another simplification introduced by LE is that profiles are defined in such a manner that the number of mandatory characteristics a device needs to support is kept to a minimum with the remaining characteristics defined as optional. So, depending on the complexity that the device can handle, it can either implement only minimum mandatory characteristics or many more optional characteristics.

For example, the device information service defines several characteristics including:

1. Manufacturer Name String.
2. Model Number String.
3. Serial Number String.
4. Hardware Revision String.
5. Firmware Revision String.
6. Software Revision String.

It is mandatory to implement any one of the characteristics and the remaining characteristics may be optional.

GATT defines the concept of a service and a characteristic. A profile can provide one or more services. Each service comprises one or more characteristics. This is shown in Figure 13.3.

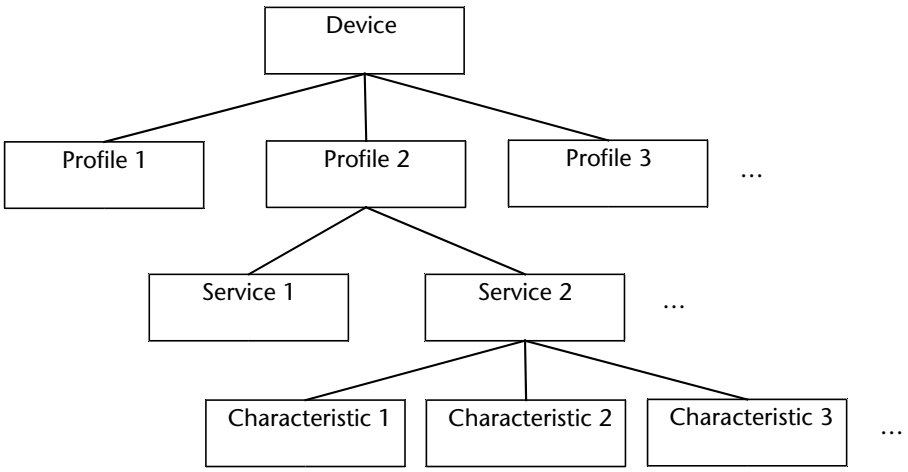

Figure 13.3 Relationship between profiles, services, and characteristcs.

Grouping of attributes—An Analogy.

As an analogy, let's take the example of an organization which contains, let's say 500 people. It would be very difficult to manage the organization if all the 500 people are organized in a flat structure. So they would be grouped into various departments. There could be a marketing department, an admin department, an engineering department, and so on. The engineering department may further be organized into subdepartments like software engineering, hardware engineering, architecture, development, and testing. Organizing the people into various departments and subdepartments makes the functioning of the organization smoother, better organized, and effective.

ATT defines a flat structure of attributes and relevant operations for those attributes. This could be considered to be the 500 people in the organization.

GATT organizes those into profiles, services, and characteristics. A profile could be considered similar to a department like HR or admin. Each department could be functioning independently. Similarly profiles are independent of each other. Each profile can provide one or more services. This could be considered to be services provided by each department. HR could provide services of payroll, training, and so on.

Finally, each service could either contain subservices or contain one or more characteristics. In our analogy, subservices could be the subdepartments like software engineering, or hardware engineering. The characteristics would be the people who are providing these services of payroll, training, engineering etc.

In summary, GATT groups similar attributes into structures which are easy to manage instead of one large collection of attributes. So it groups all attributes related to temperature together in one service, time together in another service, and heart rate in a third service. A profile may contain one or more such services.

The GATT-based profile architecture is illustrated in Figure 13.4 using four profiles as examples.

The Proximity Profile (PXP) has two roles—Monitor role and Reporter role. In the reporter role, it provides three services:

1. Link Loss Service—Mandatory.
2. Immediate Alert Service—Optional.
3. Tx Power Service—Optional.

The Link Loss Service contains the Alert Level Characteristic. The Tx Power Service contains the Tx Power Level Characteristic.

The Find Me Profile (FMP) has two roles—Locator and Target. In the locator role, it does not provide any services. In the target role, it provides only one service:

1. Immediate Alert Service—Mandatory.

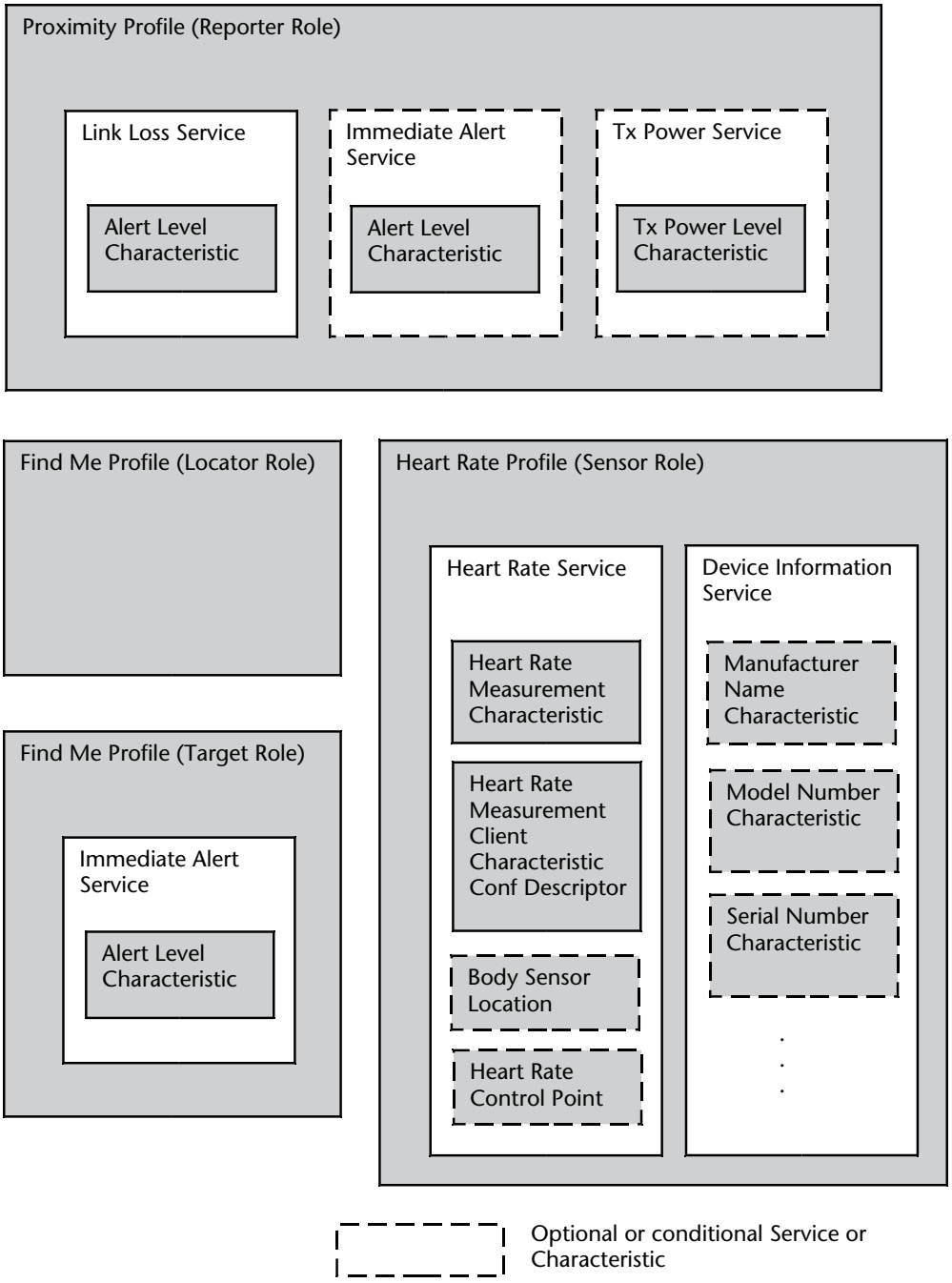

Figure 13.4 Example of profiles, services, and characteristics.

While the Immediate Alert Service is optional in PXP, it is mandatory in FMP. If a device supports both profiles, then this service can be shared with both of the profiles.

The Heart Rate Monitor profile (HRP) has two roles—Collector and Sensor. In the sensor role, it provides two services:

1. Heart Rate Service—Mandatory.
2. Device Information Service—Mandatory.

The Heart Rate Service (HRS) provides four characteristics:

1. Heart Rate Measurement— Mandatory.
2. Heart Rate Measurement Client Characteristic Configuration Descriptor—Mandatory.
3. Body Sensor Location—Conditional.
4. Heart Rate Control Point—Conditional.

13.1.2.1 Profile

A profile is composed of one or more services that are needed to fulfill a function. Some of the GATT-based profiles will be explained in detail in Chapter 15.

13.1.2.2 Service

A service can be considered to be a data structure used to describe a particular function or a feature. A service is a collection of characteristics. Besides characteristics, a service may reference other services using include definitions.

The specification defines the services separately from the profiles. So it is possible for two or more profiles to use some common services if those are needed to fulfill the particular use those profiles are supporting.

For example, the Device Information Service (DIS) shown in Figure 13.4 is a generic service that provides information about the device like the manufacturer name, serial number, model number, etc. This service can be included in the profiles which need to provide this information to the remote devices. As another example, the Immediate Alert Service (IAS) is included in both Find Me Profile and Proximity Profile. In the Find Me Profile, this service is mandatory while in the Proximity Profile this service is optional.

Every device must expose at least two services:

1. Generic Access Service.
2. Generic Attribute Service.

A service is defined by its service definition. The service definition may contain:

- References to other services using include definitions.
- Mandatory characteristics.
- Optional characteristics.

The service definition will be explained in detail later.

A service is defined in a specification which is different from the profile specifications. So there are service specifications for Immediate Alert Service, Device Information Service, Link Loss Service, etc.

There are two types of services:

1. Primary Service.
2. Secondary Service.

Primary Service
A primary service exposes the primary usable functionality of the device. The Primary services of a device are discovered using the Primary Service Discovery procedure that will be explained later in this chapter. For example, in the practical example shown in the previous chapter, the primary services discovered from the remote device included:

- Link Loss Alert.
- Immediate Alert Service.
- Tx Power Service.

Secondary Service
A secondary service is intended to be referenced from a primary service, another secondary service, or other higher layer specification. It provides auxiliary functionality for the device. It is only relevant in the context of the service or higher layer specification that references it.

Referenced Service
GATT provides a mechanism for a service to reference other services. When a service references another service, the entire referenced service becomes a part of that service. This includes any characteristics or included services as well. GATT permits the references to be nested without any restrictions on the depth of the references.

The service definition contains an include definition at the beginning of the service definition to include another service.

13.1.2.3 Characteristic

A characteristic is a value used in a service along with information about that value. A characteristic is defined by a characteristic definition. A characteristic definition contains a characteristic declaration, characteristic properties and a characteristic value. Besides this, it may optionally contain one or more characteristic descriptors that describe the value or permit configuring that value.

Some examples of characteristics are:

- Device Name Characteristic to provide the friendly name of the device to remote devices.
- Manufacturer Name, Model Number, Serial Number Characteristics to provide information about the device.
- Temperature Measurement Characteristic to provide a temperature measurement.
- Temperature Type Characteristic to describe the location on the human body where the temperature is measured.

Some examples of characteristic descriptors are:

- Characteristic Descriptor for the Temperature Measurement Characteristic to configure that characteristic to generate an indication when the temperature measurement is ready.
- Characteristic Descriptor for the Blood Pressure Measurement Characteristic to configure that characteristic to generate an indication whenever the blood pressure measurement is ready.

Figure 13.5 shows a simplified view of the Heart Rate Profile to illustrate the relationship between profile, service and characteristic.

13.2 Roles

Similar to ATT, GATT defines the following two roles:

1. *Client:* The client initiates transactions to the server and can receive responses from the server. This includes commands and requests sent to the server and responses, indications and notifications received from the server.
2. *Server:* The server receives the commands and requests from the client and sends responses, indications and notifications to the Client.

A device can act in both the roles at the same time. The various transactions between a client and the server are shown in Figure 13.6.

An example of a GATT client and server is shown in Figure 13.7. The mobile phone acts as a GATT client and the heart rate monitor acts as a GATT server. The mobile phone sends requests to the heart rate monitor to get information from it like the Model Number. The heart rate monitor sends the Model number in the response. Once the Heart Rate Monitor has collected a measurement, it may send a notification to the mobile phone of the Heart Rate Measurement Value.

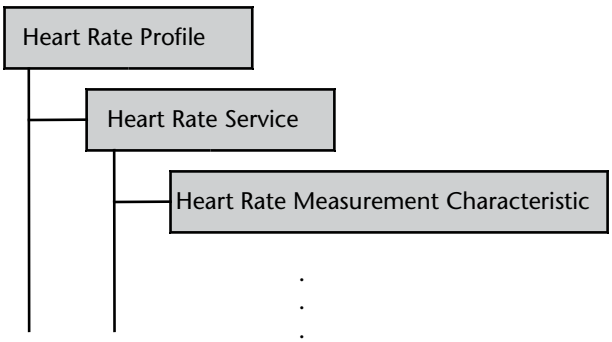

Figure 13.5 Profile, service, and characteristic for heart rate profile.

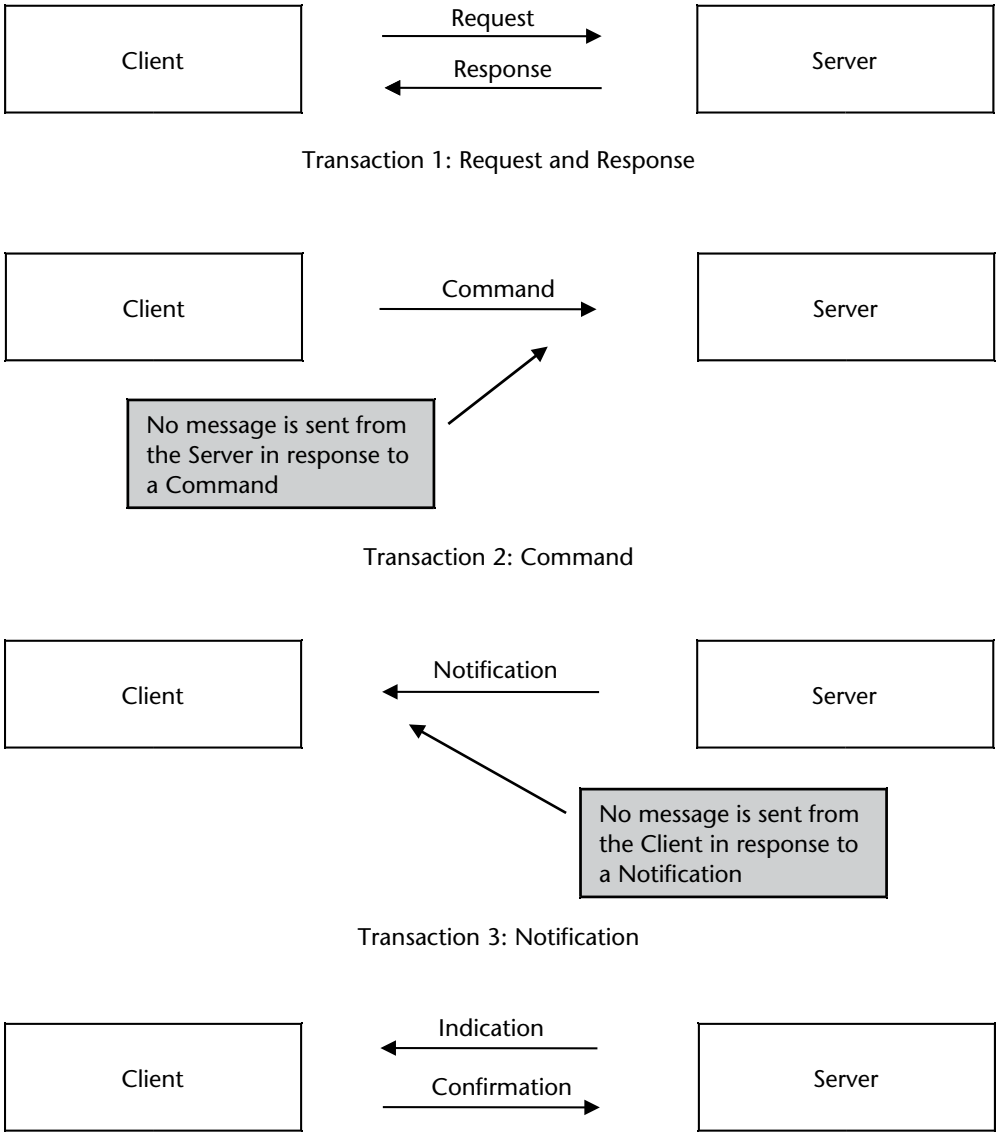

Figure 13.6 GATT Client and Server transactions.

13.3 Attributes

The structure of an attribute is defined by the ATT protocol as explained in Chapter 12. It is shown in Figure 13.8.

The Attribute Handle is used to uniquely refer to an attribute. It ranges from 0x0001 to 0xFFFF. The attributes are arranged in increasing order of attribute handles. It is not mandatory for the attribute handles to be contiguous. There may be gaps in the numbering of the attribute handles.

The Attribute Type describes the type of an attribute. It is in the form of a 16-bit UUID or 128-bit UUID.

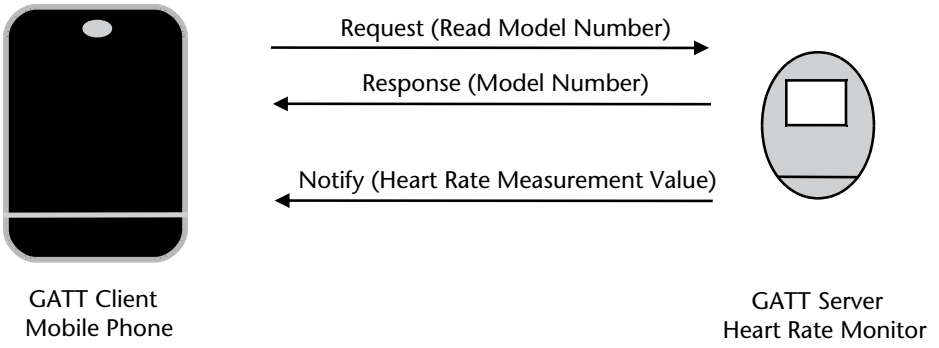

Figure 13.7 GATT Client and Server example.

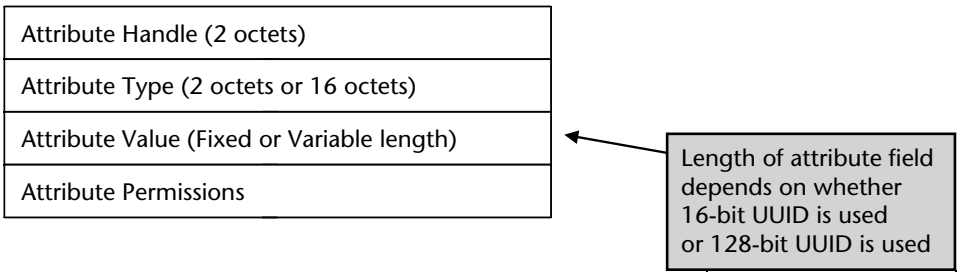

Figure 13.8 Attribute structure.

The Attribute Permissions are defined by either the GATT profile or a higher layer profile or application. This defines whether an attribute can be read or written or whether it requires authentication or authorization. As shown in Figure 13.10, the attributes containing list of services and characteristics supported by a device have the following permissions:

- Read-Only.
- No Authentication required.
- No Authorization required.

The permissions of the Characteristic Value Declaration or the Characteristic Descriptor Declaration are not set by the GATT profile. Rather these are set by the higher layer profile or the implementation. For example, the Health Thermometer Service defines the following permissions for the characteristics:

- Temperature Measurement—No Permissions required.
- Measurement Interval—No Permissions required for reading, Authentication required for writing.

This means that any device can read the measurement interval but the device needs to authenticate itself if it wants to write this characteristic.

13.3.1 Attribute Caching

In order to save power, GATT allows reducing the number of transactions between the client and server by supporting caching of attributes. The clients can discover the set of attributes on the server and store this information. The clients can use these values during the lifetime of a connection and also across connections. This means that once a client disconnects, it can again use the information it had discovered about the attributes when it connects again to the server. This saves both a significant number of packet exchanges and a lot of time that would otherwise have been spent in discovering the attributes again.

The devices in which the services can change over a period of time define a Service Changed Characteristic. This characteristic supports indication. If any of the services on the device changes (added, modified, or removed), the server sends an indication. Three scenarios are possible:

1. One or more clients are connected to the server: The indication is sent by the server to all the connected client(s).
2. One or more clients are bonded but not connected: The indication is sent by the server whenever the client(s) connect next time.
3. The clients are not bonded or connected: The attribute cache is considered to be valid only during the connection. So the clients should do an attribute discovery on connection if the server supports Service Changed Characteristic.

This is shown in Figure 13.9.

The Service Changed Indication contains the range of Attribute Handles that may have changed and for which the attribute cache is no longer valid. The Service Changed Characteristic is explained later in this chapter.

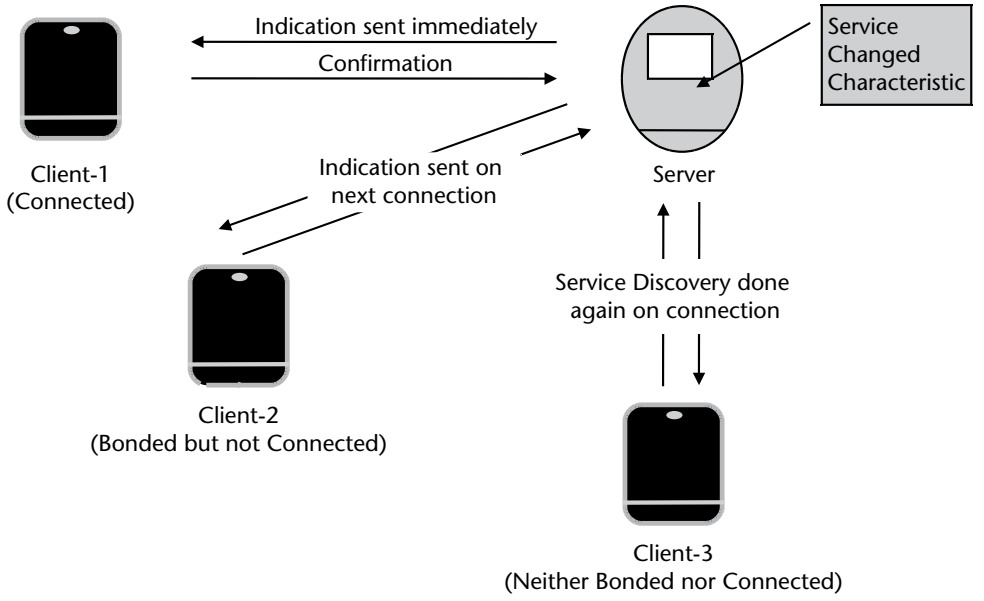

Figure 13.9 Attribute caching.

13.3.2 Attribute Grouping

ATT protocol provides support for attributes and procedures for accessing those services. GATT defines a structure using the attributes and groups the attributes into three types:

- <<Primary Service>>
- <<Secondary Service>>
- <<Characteristic>>

Depending on the implementation, a device may implement the groups that are required for that particular implementation. GATT provides procedures for accessing the services and characteristics.

13.3.3 Notification and Indication

Notifications and Indications are another power saving mechanism introduced by the LE specification. The power consumption of devices which use Notification and Indication procedures is generally much lower than the scenario where the same functionality is implemented using Read procedures.

As an analogy, consider the interrupt and polling mechanisms in the microprocessor world. If a peripheral like keyboard is connected to a microprocessor, the microprocessor can get the keystrokes from the keyboard using two mechanisms:

- Polling: The microprocessor reads the keyboard registers periodically to see if a key has been pressed.
- Interrupt: The keyboard sends an interrupt whenever the user presses a key.

If the microprocessor uses polling mechanism, it may need to read the keyboard registers as frequently as once in 10 milliseconds in order to ensure that keystrokes are not lost. If the user is away for several hours, this would mean that microprocessor will continue generating read transactions without fetching any meaningful data leading to unnecessary power consumption.

If the microprocessor uses the interrupt mechanism, it will be informed by the keyboard whenever a key is pressed and the microprocessor can do the appropriate processing only when the key is pressed. This would lead to significant power savings.

The Notification and Indication mechanisms are comparable to interrupts. Whenever the LE server has to inform the LE client(s) that it has some data to send, it can send a notification or indication and the client can do the appropriate processing only when the key is pressed.

Some examples are:

- The Health Thermometer Profile uses Indication for the Temperature Measurement characteristic. Whenever it has a temperature measurement to send to the clients, it sends an Indication.

- The Heart Rate Monitor Profile uses Notification for Heart Rate Measurement characteristic. Whenever it has a heart rate measurement to send to the clients, it sends a Notification.

The main difference between Notification and Indication is that with Notification there is no acknowledgement sent by the client to the server while in the case of Indication the client sends a Confirmation to the server so that the server is informed that the client has received the Indication.

13.4 Service Definition

As explained earlier, a service is defined by its service definition. A service definition contains three parts in the following order:

1. Service Declaration (Mandatory).
2. Include Definitions (Optional: Zero or More).
3. Characteristic Definitions (Optional: Zero or More).

The structure of the service definition is shown in Figure 13.10 and explained in the following sections.

13.4.1 Service Declaration

The definition of a service starts with a service declaration. The service declaration is the mandatory part of a service definition. This is an attribute with:

- Attribute Type = UUID of <<Primary Service>> or <<Secondary Service>>
- Attribute Value = UUID of service. This is called Service UUID.
- Attribute Permissions = Read Only, No Authentication, No Authorization.

13.4.2 Include Definition

An include definition is used to reference other services. A service definition may contain zero or more include definitions. Each include definition contains only one include declaration. The include declaration is an attribute with:

- Attribute Type = UUID of <<Include>>
- Attribute Value = Attribute handle of included service, end group handle, and service UUID.
- Attribute Permissions = Read Only, No Authentication, No Authorization.

13.4.3 Characteristic Definition

A characteristic definition is used to define a characteristic within a service. It contains a value as well as information about the value. A service definition may contain

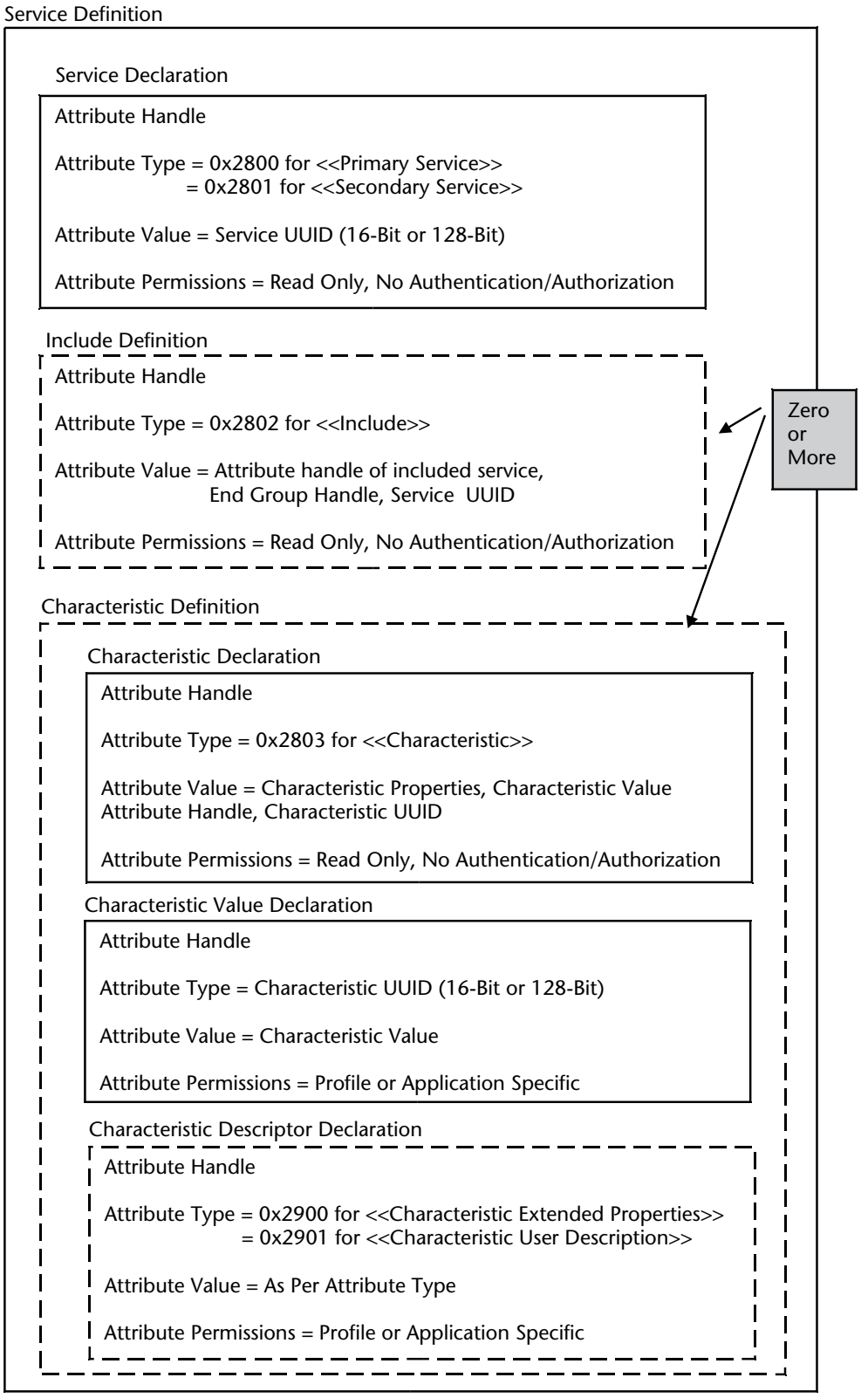

Figure 13.10 Service definition.

zero or more characteristic definition depending on the number of characteristics it supports. Each characteristic definition includes three parts in the following order:

1. Characteristic Declaration (Mandatory): Information about the characteristic.
2. Characteristic Value Declaration (Mandatory): Value of the characteristic.
3. Characteristic Descriptor Declaration (Optional: Zero or More): Additional information.

A service may contain both mandatory and optional characteristic definitions. The mandatory definitions are located before the optional ones.

13.4.3.1 Characteristic Declaration

A characteristic declaration is used to declare the information about a characteristic like the characteristic's UUID, properties, etc. It is mandatory to include it within a characteristic definition. It is an attribute with:

- Attribute Type = UUID for <<Characteristic>>
- Attribute Value = The Attribute Value containing:
  a. *Characteristic Properties:* This is a bit field that determines how the Characteristic Value can be used. For example, whether the value can be broadcast, read, written, notified or indicated. For example, a Read Only characteristic has this value set to 0x02 and a broadcast characteristic has this value set to 0x01.
  b. *Characteristic Value Attribute Handle:* This is the Attribute Handle of the Attribute that contains the Characteristic Value. Characteristic Value is explained in the next section.
  c. *Characteristic UUID:* This is a 16-bit or 128-bit to describe the type of Characteristic Value.
- Attribute Permissions = Read Only, No Authentication, No Authorization.

The significance of various bits in the Characteristic Properties bit field is shown in Table 13.1. For example if the value of properties is 0x08, then it would mean that the Characteristic Value can be read and written.

13.4.3.2 Characteristic Value Declaration

A Characteristic Value Declaration contains the value of the characteristic. It is mandatory to include it within a characteristic definition. This is an attribute with:

- Attribute Type = Same as the UUID provided in Characteristic Declaration.
- Attribute Value = Characteristic Value.
- Attribute Permissions = Higher layer profile or implementation specific.

It may be noted that for the previous attributes, the permissions were defined by GATT to be Read Only, No Authentication and No Authorization while for

Table 13.1 Bit-Fields for Characteristic Properties

Properties	*Value*	*Description*
Broadcast	0x01	Broadcast of Characteristic value permitted.
Read	0x02	Read permitted
Write Without Response	0x04	Write Without Response permitted
Write	0x08	Write permitted
Notify	0x10	Notifications permitted
Indicate	0x20	Indications permitted.
Authenticated Signed Writes	0x40	Signed write permitted
Extended Properties	0x80	Additional properties are defined in the Characteristics Extended Properties Descriptor.

the Characteristic Value the permissions are defined by the higher layer profile or implementation. This is because GATT defines that any client can read the list of services and characteristics supported by a device, but, whether a client can read or write the value of a characteristic is left to the discretion of the higher layer profile or implementation.

13.4.3.3 Characteristic Descriptor Declaration

The Characteristic Descriptors are used to provide additional information about the Characteristic Value. These are optional and a Characteristic Definition may contain more than one of these. A characteristic descriptor is an attribute with:

- Attribute Type = UUID for the additional information provided by this Characteristic Descriptor.
- Attribute Value = Value of the additional information provided by this Characteristic Descriptor.
- Attribute Permissions = Either defined by GATT or Higher layer profile or implementation specific.

For example a Characteristic Descriptor Declaration may be used to provide a user friendly string about the description of a Characteristic Value. If a Characteristic Value provides the temperature then this descriptor could provide a string saying “Room Temperature” or “Celcius”. These strings could be used for preparing user interfaces on the client side.

This descriptor is also used to specify if the value can be indicated, notified, or broadcast. For example, the Temperature Measurement Characteristic in the Health Thermometer Service has the properties of “Indicate”. This characteristic also has an associated Client Characteristic Configuration Descriptor which can be configured for indication. So once a temperature measurement is available and this descriptor is configured for indication, an indication is sent to the peer device.

The Characteristic Descriptor could also be used to specify the format of the Characteristic Value. For example, whether the value is 1-bit, 8-bit, 16-bit, signed or unsigned, a UTF-8 string or UTF-16 string, or whether it is a floating point value.

13.5 Configured Broadcast

A client can configure the server to broadcast certain characteristic values in advertising data. This is done by the client setting the broadcast configuration bit in the client characteristic configuration.

13.6 GATT Features

The GATT profile defines eleven features. Each feature is mapped into procedures and subprocedures which use the procedures defined by the ATT protocol.

A summary of the features and the corresponding procedures and sub-procedures is shown in Table 13.2. These are described in detail in the following sections.

13.6.1 Server Configuration

The server configuration feature is used to configure the parameters of the ATT protocol. It has only one subprocedure related to configuring the ATT_MTU size.

13.6.1.1 Exchange MTU

The exchange MTU subprocedure is initiated only once during a connection by the client to set the ATT_MTU to be used for the connection.

Table 13.2 GATT Features and Procedures

S. No	*Feature*	*Subprocedures*
1	Server Configuration	Exchange MTU
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 Characteristics of a Service
		Discover Characteristic by UUID
5	Characteristic Descriptor Discovery	Discover All Characteristic Descriptors
6	Characteristic Value Read	Read Characteristic Value
		Read Using Characteristic UUID
		Read Long Characteristic Values
		Read Multiple Characteristic Values
7	Characteristic Value Write	Write Without Response
		Signed Write Without Response
		Write Characteristic Value
		Write Long Characteristic Values
		Characteristic Values Reliable Writes
8	Characteristic Value Notification	Notifications
9	Characteristic Value Indication	Indications
10	Characteristic Descriptor Value Read	Read Characteristic Descriptors
		Read Long Characteristic Descriptors
11	Characteristic Descriptor Value Write	Write Characteristic Descriptors
		Write Long Characteristic Descriptors