Part G. Generic Attribute Profile (GATT)

This Part defines the Generic Attribute Profile that describes a service framework using the Attribute Protocol for discovering services, and for reading and writing characteristic values on a peer device.

1. Introduction

1.1. Scope

The Generic Attribute profile (GATT) defines a service framework using the Attribute Protocol. This framework defines procedures and formats of services and their characteristics. The procedures defined include discovering, reading, writing, notifying and indicating characteristics, as well as configuring the broadcast of characteristics.

1.2. Profile dependency

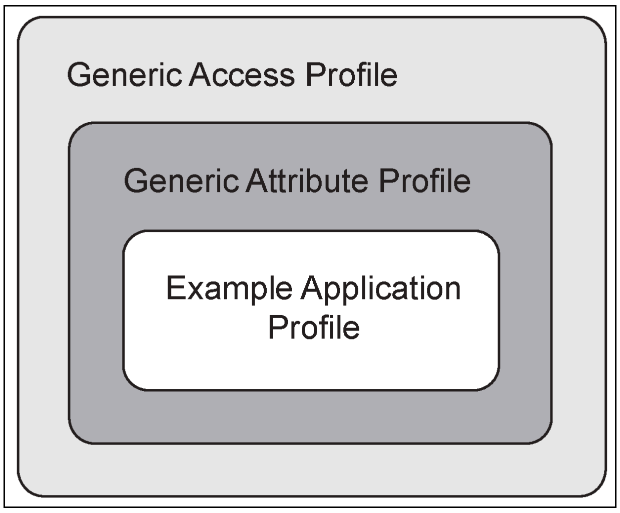


Figure 1.1: Profile dependencies

Figure 1.1[generic-attribute-profile--gatt-.html#UUID-8ab960a7-6d9e-ff83-780d-107f88f3a7e9_figure-idm46218671075200335988107992 64] depicts the structure and the dependencies of the profiles. A profile is dependent upon another profile if it re-uses parts of that profile by implicitly or explicitly referencing it.

1.3. Conformance

If conformance to this profile is claimed, all capabilities indicated as mandatory for this profile shall be supported in the specified manner (process-mandatory). This also applies for all optional and conditional capabilities for which support is indicated. All mandatory capabilities, and optional and conditional capabilities for which support is indicated, are subject to verification as part of the Bluetooth qualification program.

1.4. [This section is no longer used]

1.5. Conventions

In this Part the use of literal terms such as procedure, PDUs, opcodes, or function names appear in italics. Specific names of fields in structures, packets, etc. also appear in italics. The use of « » (e.g. «Primary Service») indicates a UUID. Attribute Protocol error codes (see [Vol 3] Part F, Table 3.4[attribute-protocol--att-.html#UUID-eefd3e8d-9b16-3af8-1fb4-fa90f52262e8_informaltable-idm13358908515990]) appear in italics followed by the numeric error code.

2. Profile overview

The GATT profile is designed to be used by an application or another profile, so that a client can communicate with a server. The server contains a number of attributes, and the GATT Profile defines how to use the Attribute Protocol to discover, read, write and obtain indications of these attributes, as well as configuring broadcast of attributes.

2.1. Protocol stack

Figure 2.1[generic-attribute-profile--gatt-.html#UUID-99035c00-d3cb-4f06-0aa3-7ac8924a5056_figure-idm4525247752496033598814852002] shows the peer protocols used by this profile.

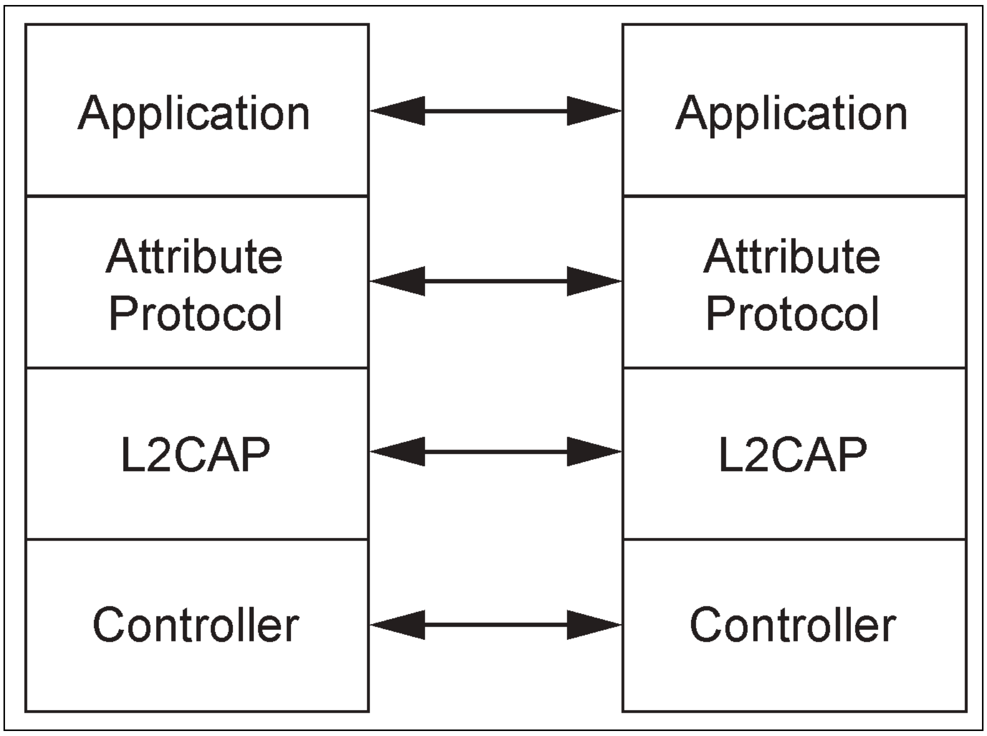


Figure 2.1: Protocol model

2.2. Configurations and roles

The following roles are defined for devices that implement this profile:

Client—This is the device that initiates commands and requests towards the server and can receive responses, indications and notifications sent by the server.

Server—This is the device that accepts incoming commands and requests from the client and sends responses, indications and notifications to a client.

Note: The roles are not fixed to the device. The roles are determined when a device initiates a defined procedure, and they are released when the procedure ends.

A device can act in both roles at the same time.

An example of configurations illustrating the roles for this profile is depicted in Figure 2.2[generic-attribute-profile--gatt-.html#UUID-94874938-df03-a9d4-b714-fc8b2b89af7b_figure-idm4525248300691233598821764718].

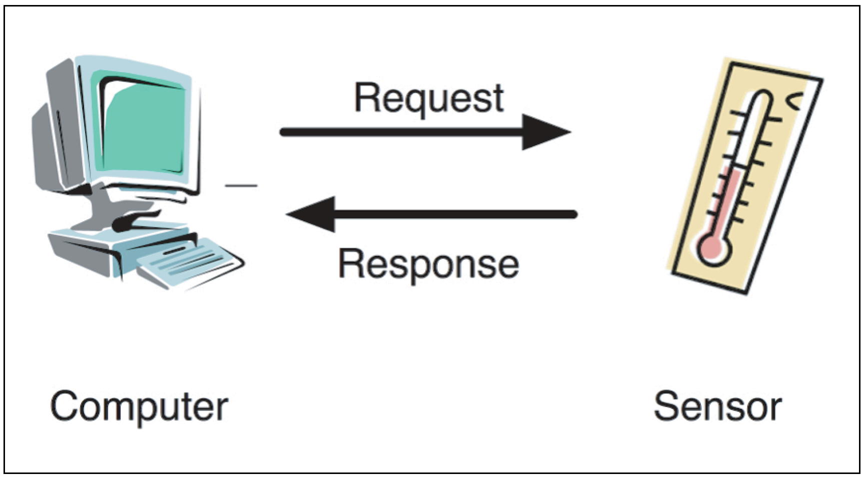


Figure 2.2: Examples of configuration

In Figure 2.2[generic-attribute-profile--gatt-.html#UUID-94874938-df03-a9d4-b714-fc8b2b89af7b_figure-idm4525248300691233598821764718], the computer is the temperature service client and the sensor is the temperature service server. The computer initiates procedures to configure the sensor or to read the sensor values. In this example the sensor provides information about the characteristics the sensor device exposes as part of the temperature service and may permit some characteristics to be written. Also, the sensor responds to read requests with the appropriate values.

2.3. User requirements and scenarios

The following scenarios are covered by this profile:

- Exchanging configuration
- Discovery of services and characteristics on a device
- Reading a characteristic value
- Writing a characteristic value
- Notification of a characteristic value
- Indication of a characteristic value

2.4. Profile fundamentals

This profile can be used over any physical link, using the Attribute Protocol L2CAP channel, known as the ATT Bearer. Here is a brief summary of lower layer requirements communication between the client and the server.

- An ATT bearer is established using “Channel Establishment” as defined in Section 6[generic-attribute-profile--gatt-.html#UUID-4d19101a-a74f-ce90-de18-86e5c263156f].
- The profile roles are not tied to the Controller roles (i.e. Central or Peripheral).
- On an LE Physical link, use of security features such as authorization, authentication and encryption are optional. On a BR/EDR physical link encryption is mandatory.
- Multi-octet fields within the GATT profile shall be sent least significant octet first (little-endian) with the exception of the Characteristic Value field. The Characteristic Value and any fields within it shall be little-endian unless otherwise defined in the specification which defines the characteristic.
- Multiple ATT bearers may be established between a client and a server. The server can determine if ATT bearers are from the same client, and vice versa, by using connection information such as the Bluetooth Device Address of the peer device.

2.5. Attribute Protocol

The GATT profile requires the implementation of the Attribute Protocol (See [Vol 3] Part F[attribute-protocol--att-.html]) and those Attribute Protocol PDUs required by Section 4.2[generic-attribute-profile--gatt-.html#UUID-d8c3186f-a603-00e3-8447-a3b9def32039] and Section 4.13[generic-attribute-profile--gatt-.html#UUID-b5a253e2-98a4-63df-fc1a-e4555ec83ddc].

2.5.1. Overview

The GATT Profile uses the Attribute Protocol to transport data in the form of commands, requests, responses, indications, notifications and confirmations between devices. This data is contained in Attribute Protocol PDUs as specified in Figure 2.3[generic-attribute-profile--gatt-.html#UUID-63607703-7747-a345-fb4e-4954eea4219b_figure-idm4508319785212833598825351662].

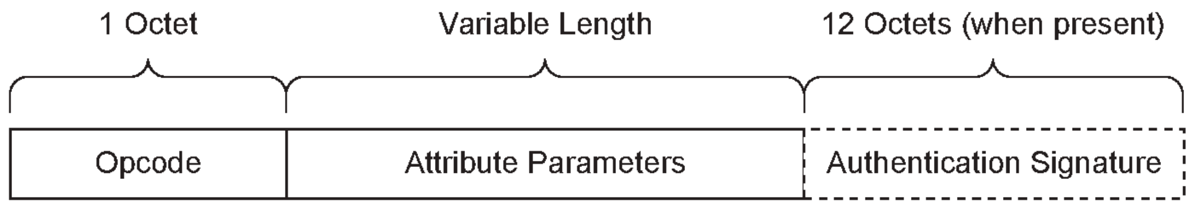


Figure 2.3: Attribute Protocol PDU

The *Opcode* contains the specific command, request, response, indication, notification or confirmation opcode and a flag for authentication. The *Attribute Parameters* contain data for the specific command or request or the data returned in a response, indication, or notification. The *Authentication Signature* is optional and is described in [Vol 3] Part H, Section 2.4.5[security-manager-specification.html#UUID-193f95ea-7252-1b51-853b-a1999393dddf].

Attribute Protocol commands and requests act on values stored in Attributes on the server device. An Attribute is composed of four parts: *Attribute Handle*, *Attribute Type*, *Attribute Value*, and *Attribute Permissions*. Figure 2.4[generic-attribute-profile--gatt-.html#UUID-63607703-7747-a345-fb4e-4954eea4219b_figure-idm4525248221760033598827807 24] shows a logical representation of an Attribute. The actual representation for a given implementation is specific to that implementation.

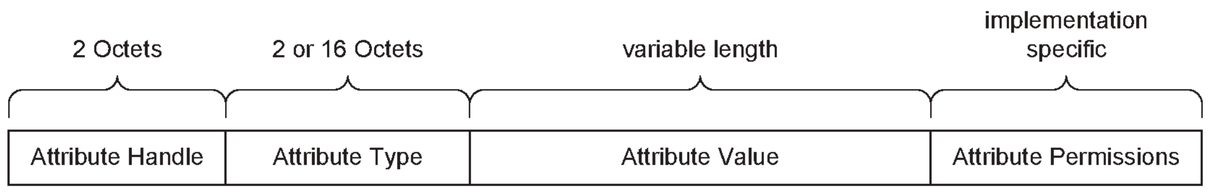


Figure 2.4: Logical attribute representation

The *Attribute Handle* is an index corresponding to a specific Attribute. The *Attribute Type* is a UUID that describes the *Attribute Value*. The *Attribute Value* is the data described by the *Attribute Type* and indexed by the *Attribute Handle*. The Attributes are ordered by increasing *Attribute Handle* values. *Attribute Handle* values may begin at any value between 0x0001 and 0xFFFF. Although the *Attribute Handle* values are in increasing order, following *Attribute Handle* values may differ by more than one. That is to say there may be gaps between successive *Attribute Handles*. When the specification requires two attribute handles to be adjacent or for one to immediately follow one the other, such gaps are still permitted and shall be ignored.

Attribute Permissions is part of the Attribute that cannot be read from or written to using the Attribute Protocol. It is used by the server to determine whether read or write access is permitted for a given attribute. *Attribute Permissions* are established by the GATT profile, a higher layer profile or are implementation specific if not specified.

2.5.2. Attribute caching

Attribute caching is an optimization that allows the client to discover the Attribute information such as *Attribute Handles* used by the server once and use the same Attribute information across reconnections without rediscovery. If the client does not cache the Attribute information, then it must rediscover the *Attribute* information at each reconnection. With caching, time is saved and a significant number of packets need not be exchanged between the client and server. The Attribute information that shall be cached by a client is the *Attribute Handles* of all server attributes and the GATT service characteristics values.

Attribute Handles used by the server should not change over time. This means that once an *Attribute Handle* is discovered by a client the *Attribute Handle* for that Attribute should not be changed.

Some circumstances may cause servers to change the *Attribute Handles* used for services, perhaps due to a factory reset or a firmware upgrade procedure being performed. The following is only required on the server if the services on the server can be added, modified or removed. If GATT based services on the server cannot be changed during the usable lifetime of the device, the *Service Changed* characteristic shall not exist on the server and the client does not need to ever perform service discovery after the initial service discovery for that server.

To support caching when a server supports changes in GATT based services, an indication is sent by the server to clients when a service is added, removed, or modified on the server. A client may also detect a service change by reading the *Database Hash* characteristic if that characteristic exists on the server. A GATT based service is considered modified if the binding of the *Attribute Handles* to the associated Attributes grouped within a service definition are changed. Any change to the GATT service definition characteristic values other than the *Service Changed* characteristic value and the *Client Supported Features* characteristic value themselves shall also be considered a modification.

For clients that have a trusted relationship (i.e. bond) with the server, the attribute cache is valid across connections. For clients with a trusted relationship and not in a connection when a service change occurs, the server shall send an indication when the client reconnects to the server (see Section 7.1[generic-attribute-profile--gatt-.html#UUID-6ee92321-3db4-dad2-554e-946a80ff7435]). For clients that do not have a trusted relationship with the server and that do not support reading the *Database Hash* characteristic, the attribute cache is valid only during the connection. Clients without a trusted relationship that do support reading the *Database Hash* characteristic may validate the attribute cache on connection setup. Clients without a trusted relationship shall receive an indication when the service change occurs only during the current connection.

Note: Clients without a trusted relationship that support caching must either perform service discovery or detect service changes by reading the *Database Hash* characteristic on each connection if the server supports the *Service Changed* characteristic.

The server shall send an ATT_HANDLE_VALUE_IND PDU containing the range of affected *Attribute Handles* that shall be considered invalid in the client's attribute cache. The start *Attribute Handle* shall be the start *Attribute Handle* of the service definition containing the change and the end *Attribute Handle* shall be the last *Attribute Handle* of the service definition containing the change. The value in the indication is composed of two 16-bit *Attribute Handles* concatenated to indicate the affected *Attribute Handle* range.

Note: A server may set the affected *Attribute Handle* range to 0x0001 to 0xFFFF to indicate to the client to rediscover the entire set of *Attribute Handles* on the server.

If the *Database Hash* characteristic exists on the server then, each time a service change occurs, the server shall update the *Database Hash* characteristic value with the new Database Hash (see Section 7.3[generic-attribute-profile--gatt-.html#UUID-cc9d0db2-8c48-0611-ff58-4b9323fc4d12]).

If the *Database Hash* characteristic value has changed since the last time it was read, the client shall consider its attribute cache invalid and shall not make use of the cached information until it has performed service discovery or obtained the changed database definitions using an out-of-band mechanism.

The client, upon receiving an ATT_HANDLE_VALUE_IND PDU containing the range of affected Attribute Handles, shall consider the attribute cache invalid over the affected Attribute Handle range. Any outstanding request transaction shall be considered invalid if the Attribute Handle is contained within the affected Attribute Handle range. The client must perform service discovery before the client uses any service that has an attribute within the affected Attribute Handle range. Alternatively, the client may read the *Database Hash* characteristic and obtain the changed database definitions using an out-of-band mechanism. If the client receives an ATT_HANDLE_VALUE_IND PDU during service discovery and the client has read the *Database Hash* characteristic prior to the service discovery, the client may read the *Database Hash* characteristic again to determine if the current service discovery can be continued or if a new service discovery is required.

Once the server has received the ATT_HANDLE_VALUE_CFM PDU, the server can consider the client to be aware of the updated Attribute Handles.

The client shall consider the affected *Attribute Handle* range to be invalid in its attribute cache and perform the discovery procedures to restore the attribute cache. The server shall store service changed information for all bonded devices.

2.5.2.1. Robust Caching

Robust Caching is a feature where the server sends an ATT_ERROR_RSP PDU to the client if the server does not consider the client to be aware of a service change.

If the *Database Hash* and *Service Changed* characteristics are both present on the server, then the server shall support the Robust Caching feature.

From the perspective of a server, each connected client is either "change-aware" or "change-unaware" regarding changes in the database definitions. After connecting to a server, the initial state of a client without a trusted relationship is change-aware. The initial state of a client with a trusted relationship is unchanged from the previous connection unless the database has been updated since the last connection, in which case the initial state is change-unaware.

Whenever the server updates the database definitions, all connected clients become change-unaware. A change-unaware connected client becomes change-aware when it reads the *Database Hash* characteristic and then the server receives another ATT request from the client. A change-unaware client using multiple ATT bearers shall wait until the server has responded to all pending requests before reading the *Database Hash* characteristic (see Figure 2.5[generic-attribute-profile--gatt-.html#UUID-787b0d4d-3112-9b07-6bef-89dda173a489_figure-idm4641016152856033598836874577]).

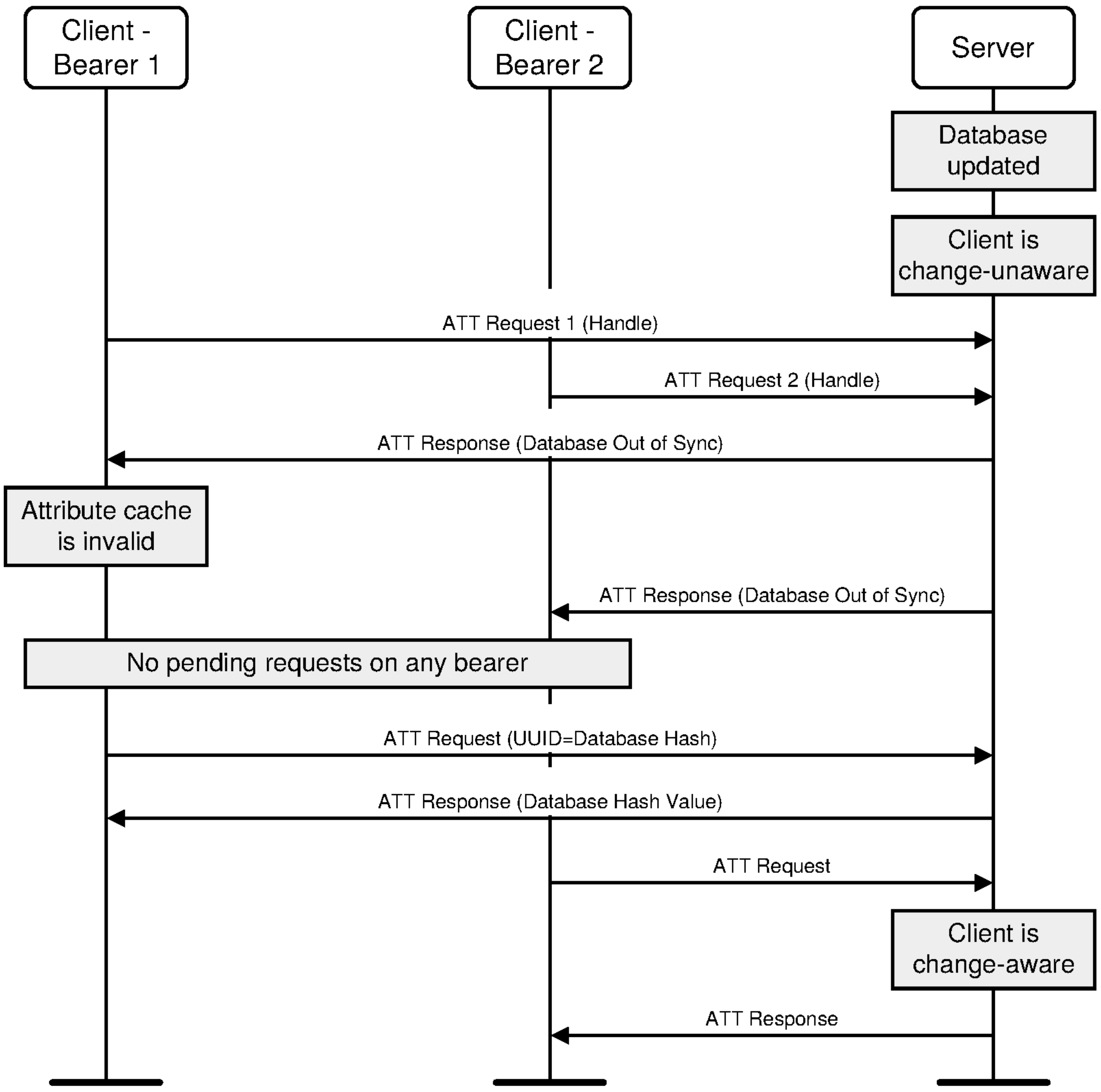

Client - Bearer 1
Client - Bearer 2
Server
Database updated
Client is change-unaware
ATT Request 1 (Handle)
ATT Request 2 (Handle)
ATT Response (Database Out of Sync)
Attribute cache is invalid
ATT Response (Database Out of Sync)
No pending requests on any bearer
ATT Request (UUID=Database Hash)
ATT Response (Database Hash Value)
ATT Request
Client is change-aware
ATT Response

Figure 2.5: Transition to change-aware state for a client using multiple ATT bearers

In addition, a change-unaware connected client using exactly one ATT bearer becomes change-aware when either of the following happen:

- The client receives and confirms a *Handle Value Indication* for the *Service Changed* characteristic (see Figure 2.6[generic-attribute-profile--gatt-.html#UUID-787b0d4d-3112-9b07-6bef-89dda173a489_figure-idm4621867096190433598838585219]).
- The server sends the client a response with the Error Code parameter set to *Database Out Of Sync* (0x12) and then the server receives another ATT request from the client (see Figure 2.7[generic-attribute-profile--gatt-.html#UUID-787b0d4d-3112-9b07-6bef-89dda173a489_figure-idm4641019516841633598840802112]).

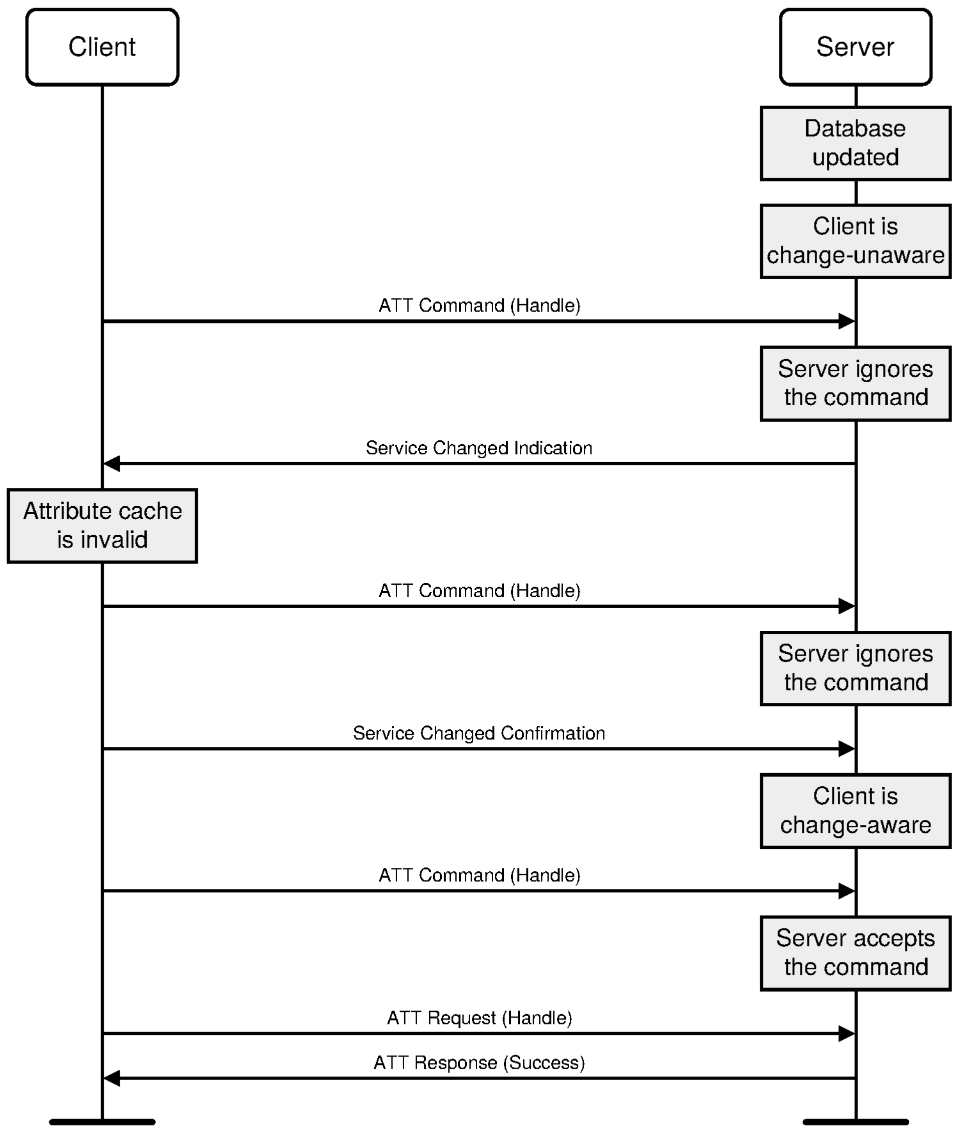


Figure 2.6: Transition to change-aware state for a client using exactly one ATT bearer, triggered by a Service Changed confirmation

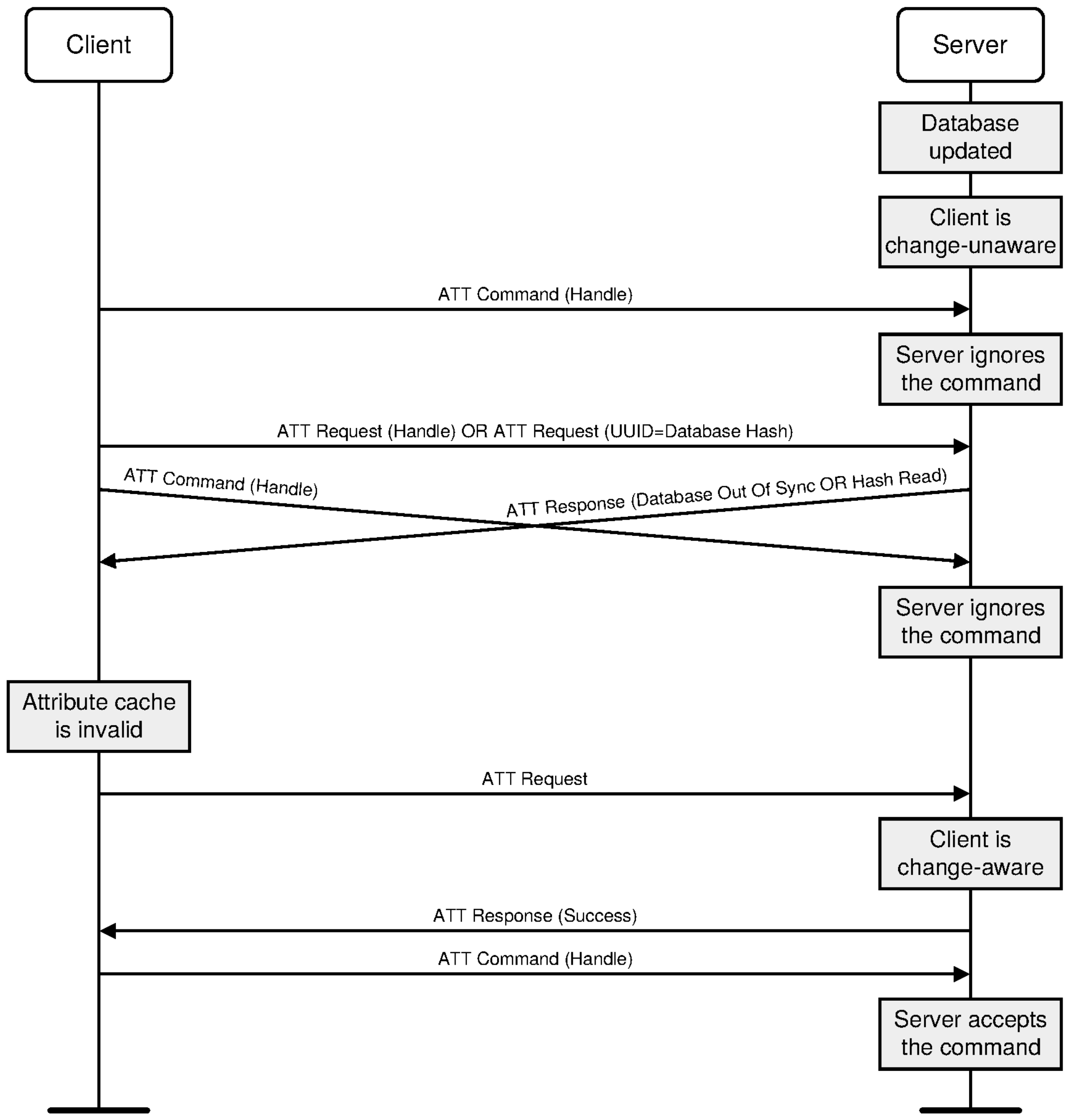

Client
Server
Database updated
Client is change-unaware
ATT Command (Handle)
Server ignores the command
ATT Request (Handle) OR ATT Request (UUID=Database Hash)
ATT Command (Handle)
ATT Response (Database Out Of Sync OR Hash Read)
Server ignores the command
Attribute cache is invalid
ATT Request
Client is change-aware
ATT Response (Success)
ATT Command (Handle)
Server accepts the command

Figure 2.7: Transition to change-aware state for a client using exactly one ATT bearer, triggered by the ATT response "Database Out Of Sync" or "Hash Read"

If a client that has indicated support for robust caching (by setting the *Robust Caching* bit in the *Client Supported Features* characteristic) is change-unaware then the server shall send an ATT_ERROR_RSP PDU with the Error Code parameter set to *Database Out Of Sync* (0x12) when either of the following happen:

- That client requests an operation at any *Attribute Handle* or list of *Attribute Handles* by sending an ATT request.
- That client sends an ATT_READ_BY_TYPE_REQ PDU with Attribute Type other than «Include» or «Characteristic» and an *Attribute Handle* range other than 0x0001 to 0xFFFF.

The ATT_ERROR_RSP PDU is sent only once per bearer after the client becomes change-unaware, unless the client disconnects or the database changes again before the client becomes change-aware in which case the ATT_ERROR_RSP PDU shall be sent again. If a change-unaware client sends an ATT command, the server shall ignore it. Except for a *Handle Value Indication* for the *Service Changed* characteristic, the server shall not send notifications and indications to such a client until it becomes change-aware. If a client is change-aware, then the server shall perform the operation normally.

Note: To reduce the probability of blocked notifications and indications, servers should send this indication as soon as possible after a service change.

If a client receives an ATT_ERROR_RSP PDU with the Error Code parameter set to *Database Out Of Sync* (0x12), it shall consider its attribute cache invalid and shall not make use of the cached information until it has performed service discovery or obtained the changed database definitions using an out-of-band mechanism.

2.5.3. Attribute grouping

The Generic Attribute Profile defines the grouping of attributes for three attribute types: «Primary Service», «Secondary Service» and «Characteristic». A group begins with a declaration, and ends as defined in Section 3.1[generic-attribute-profile--gatt-.html#UUID-2148a3e6-91f7-e758-750f-8b14377cab6e] for services and Section 3.3[generic-attribute-profile--gatt-.html#UUID-70d11f51-12cd-57a4-184a-fd8a4e0283f9] for characteristics. Not all of the grouping attributes can be used in the ATT_READ_BY_GROUP_TYPE_REQ PDU. The «Primary Service» and «Secondary Service» grouping types may be used in the ATT_READ_BY_GROUP_TYPE_REQ PDU. The «Characteristic» grouping type shall not be used in the ATT_READ_BY_GROUP_TYPE_REQ PDU.

2.5.4. UUIDs

All 16-bit UUIDs shall be contained in exactly 2 octets. All 128-bit UUIDs shall be contained in exactly 16 octets.

All 32-bit UUIDs shall be converted to 128-bit UUIDs when the UUID is contained in an ATT PDU. See [Vol 3] Part B, Section 2.5.1[service-discovery-protocol--sdp--specification.html#UUID-ef710684-4c7e-6793-4350-4a190ea9a7a4] for the method of conversion.

2.6. GATT Profile hierarchy

2.6.1. Overview

The GATT Profile specifies the structure in which profile data is exchanged. This structure defines basic elements such as services and characteristics, used in a profile. All of the elements are contained by Attributes. Attributes used in the Attribute Protocol are containers that carry this profile data.

The top level of the hierarchy is a profile. A profile is composed of one or more services necessary to fulfill a use case. A service is composed of characteristics or inclusions of other services. Each characteristic contains a value and may contain optional information about the value. The service and characteristic and the components of the characteristic (i.e. value and descriptors) contain the profile data and are all stored in Attributes on the server.

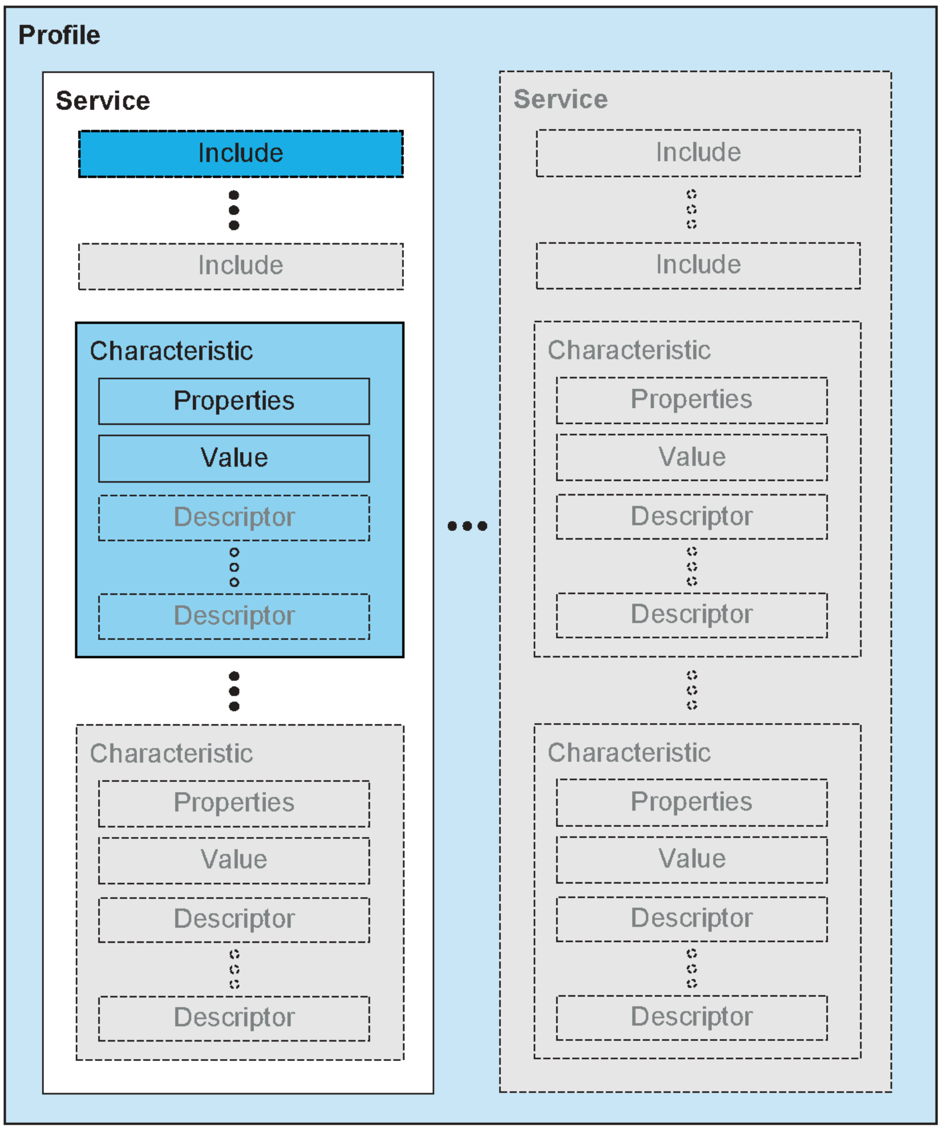


Figure 2.8: GATT Profile hierarchy

2.6.2. Service

A service is a collection of data and associated behaviors to accomplish a particular function or feature. In GATT, a service is defined by its service definition. A service definition may contain included services, mandatory characteristics, and optional characteristics.

To maintain backward compatibility with earlier clients, later versions of a service definition can only add new included services or optional characteristics. Later versions of a service definition are forbidden from changing behaviors from previous versions of the service definition.

There are two types of services: primary service and secondary service. A primary service is a service that exposes functionality of this device. A primary service can be included by another service. Primary services can be discovered using Primary Service Discovery procedures. A secondary service is a service that should only be included from a primary service or another secondary service or other higher layer specification. A secondary service is only relevant in the context of the entity that includes it.

The determination of whether a service is either a primary or secondary service can be mandated by a higher layer specification.

Note: There is no procedure for discovering secondary services.

Services may be used in one or more higher layer specifications to fulfill a particular use case.

The service definition is described in Section 3.1[generic-attribute-profile--gatt-.html#UUID-2148a3e6-91f7-e758-750f-8b14377cab6e].

2.6.3. Included services

An included service is a method to reference another service definition existing on the server into the service being defined. To include another service, an include definition is used at the beginning of the service definition. When a service definition uses an include definition to include a service, the entire included service definition becomes part of the new service definition. This includes all the included services and characteristics of the included service. The included service still exists as an independent service. A service that is included by another service shall not be changed by the act of inclusion or by the including service. There are no limits to the number of include definitions or the depth of nested includes in a service definition.

The include definition is described in Section 3.2[generic-attribute-profile--gatt-.html#UUID-a4a91ea7-a00f-77b6-9b37-fa0492451a21].

2.6.4. Characteristic

A characteristic is a value used in a service along with properties and configuration information about how the value is accessed and information about how the value is displayed or represented. In GATT, a characteristic is defined by its characteristic definition. A characteristic definition contains a characteristic declaration, characteristic properties, and a value and may contain descriptors that describe the value or permit configuration of the server with respect to the characteristic.

The characteristic definition is described in Section 3.3[generic-attribute-profile--gatt-.html#UUID-70d11f51-12cd-57a4-184a-fd8a4e0283f9].

2.7. Configured Broadcast

For LE physical links, Configured Broadcast is a method for a client to indicate to a server which *Characteristic Value* shall be broadcast in the advertising data when the server is executing the Broadcast mode procedure. For BR/EDR physical links, Configured Broadcast is not supported.

To configure a *Characteristic Value* to be broadcast by the server when in Broadcast mode, the client sets the broadcast configuration bit described in Section 3.3.3.4[generic-attribute-profile--gatt-.html#UUID-14717cf3-2f80-c811-a2d5-45b515fa5f1e]. The frequency of the broadcast is part of the service behavior definition. The data shall be broadcast as part of the Service Data Advertising Data type as defined in Section 1.11 of [3[generic-attribute-profile--gatt-.html#UUID-c32d82db-0074-99ba-c52a-307a41944215_bibliomixed-idm13391355412974]]. If multiple characteristics can simultaneously be enabled for broadcast, the service specification defines how the characteristics are to be formatted in the service data which follows the service UUID in the Service Data Advertising Data type payload.

3. Service interoperability requirements

3.1. Service definition

A service definition shall contain a service declaration and may contain include definitions and characteristic definitions. The service definition ends before the next service declaration or after the maximum *Attribute Handle* is reached. Service definitions appear on the server in an order based on *Attribute Handle*.

All include definitions and characteristic definitions contained within the service definition are considered to be part of the service. All include definitions shall immediately follow the service declaration and precede any characteristic definitions. A service definition may have zero or more include definitions. All characteristic definitions shall be immediately following the last include definition or, in the event of no include definitions, immediately following the service declaration. A service definition may have zero or more characteristic definitions. There is no upper limit for include or characteristic definitions.

A service declaration is an Attribute with the *Attribute Type* set to the UUID for «Primary Service» or «Secondary Service». The *Attribute Value* shall be the 16-bit Bluetooth UUID or 128-bit UUID for the service, known as the service UUID. A client shall support the use of both 16-bit and 128-bit UUIDs. A client may ignore any service definition with an unknown service UUID. An unknown service UUID is a UUID for an unsupported service. The *Attribute Permissions* shall be read-only and shall not require authentication or authorization.

When multiple services exist, services definitions with service declarations using 16-bit Bluetooth UUID should be grouped together (i.e. listed sequentially) and services definitions with service declarations using 128-bit UUID should be grouped together.

Attribute Handle	**Attribute Type**	**Attribute Value**	**Attribute Permission**
0xNNNN	0x2800 – UUID for «Primary Service» OR 0x2801 for «Secondary Service»	16-bit Bluetooth UUID or 128-bit UUID for Service	Read Only, No Authentication, No Authorization

Table 3.1: Service declaration

A device or higher level specification may have multiple service definitions and may have multiple service definitions with the same service UUID.

All Attributes on a server shall either contain a service declaration or exist within a service definition.

Service definitions contained in a server may appear in any order; a client shall not assume the order of service definitions on a server.

3.2. Include definition

An include definition shall contain only one include declaration.

The include declaration is an Attribute with the *Attribute Type* set to the UUID for «Include». The *Attribute Value* shall be set to the included service *Attribute Handle*, the End Group Handle, and the *service UUID*. The Service UUID shall only be present when the UUID is a 16-bit Bluetooth UUID.The *Attribute Permissions* shall be read only and not require authentication or authorization.

Attribute Handle	**Attribute Type**	**Attribute Value**			**Attribute Permission**
0xNNNN	0x2802 – UUID for «Include»	Included Service Attribute Handle	End Group Handle	Service UUID	Read Only, No Authentication, No Authorization

Table 3.2: Include declaration

A server shall not contain a service definition with an include definition to another service that includes the original service. This applies to each of the services the included definition references. This is referred to as a circular reference.

If the client detects a circular reference or detects nested include declarations to a greater level than it expects, it should terminate or stop using the ATT bearer.

3.3. Characteristic definition

A characteristic definition shall contain a characteristic declaration, a Characteristic Value declaration and may contain characteristic descriptor declarations. A characteristic definition ends at the start of the next characteristic declaration or service declaration or after the maximum *Attribute Handle*. Characteristic definitions appear on the server within a service definition in an order based on *Attribute Handle*.

Each declaration above is contained in a separate Attribute. The two required declarations are the characteristic declaration and the Characteristic Value declaration. The Characteristic Value declaration shall exist immediately following the characteristic declaration. Any optional characteristic descriptor declarations are placed after the Characteristic Value declaration. The order of the optional characteristic descriptor declarations is not significant.

A characteristic definition may be defined to concatenate several Characteristic Values into a single aggregated Characteristic Value. This may be used to optimize read and writes of multiple Characteristic Values through the reading and writing of a single aggregated Characteristic Value. This type of characteristic definition is the same as a normal characteristic definition. The characteristic declaration shall use a characteristic UUID that is unique to the aggregated characteristic definition. The aggregated characteristic definition may also contain a characteristic aggregate format descriptor that describes the display format of the aggregated Characteristic Value.

3.3.1. Characteristic declaration

A characteristic declaration is an Attribute with the Attribute Type set to the UUID for «Characteristic» and *Attribute Value* set to the Characteristic Properties, Characteristic Value *Attribute Handle* and Characteristic UUID. The Attribute Permissions shall be readable and not require authentication or authorization.

If the server changes any characteristic declaration *Attribute Value* while the server has a trusted relationship with any client, then it shall send each client a Service Changed Indication indicating a change in the service holding the Characteristic Declaration (see Section 7.1[generic-attribute-profile--gatt-.html#UUID-6ee92321-3db4-dad2-554e-946a80ff7435]).

Attribute Handle	**Attribute Types**	**Attribute Value**			**Attribute Permissions**
0xNNNN	0x2803–UUID for «Characteristic»	Characteristic Properties	Characteristic Value Attribute Handle	Characteristic UUID	Read Only, No Authentication, No Authorization

Table 3.3: Characteristic declaration

The *Attribute Value* of a characteristic declaration is read only.

Attribute Value	**Size**	**Description**
Characteristic Properties	1 octets	Bit field of characteristic properties
Characteristic Value Handle	2 octets	Handle of the Attribute containing the value of this characteristic
Characteristic UUID	2 or 16 octets	16-bit Bluetooth UUID or 128-bit UUID for Characteristic Value

Table 3.4: Attribute Value field in characteristic declaration

A service may have multiple characteristic definitions with the same Characteristic UUID.

Within a service definition, some characteristics may be mandatory and those characteristics shall be located after the include declarations and before any optional characteristics within the service definition. A client shall not assume any order of those characteristics that are mandatory or any order of those characteristics that are optional within a service definition. Whenever possible and within the requirements stated earlier, characteristics definitions with characteristic declarations using 16-bit Bluetooth UUIDs should be grouped together (i.e. listed sequentially) and characteristics definitions with characteristic declarations using 128-bit UUIDs should be grouped together.

3.3.1.1. Characteristic Properties

The Characteristic Properties bit field determines how the Characteristic Value can be used, or how the characteristic descriptors (see Section 3.3.3[generic-attribute-profile--gatt-.html#UUID-09487be3-178b-eeca-f49f-f783e8d462f6]) can be accessed. If the bits defined in Table 3.5[generic-attribute-profile--gatt-.html#UUID-80c13237-6499-b5da-e575-7147fd3c7413_table-idm13358911666278] are set, the action described is permitted. Multiple characteristic properties can be set.

These bits shall be set according to the procedures allowed for this characteristic, as defined by higher layer specifications, without regard to security requirements.

Properties	Value	Description
Broadcast	0x01	If set, permits broadcasts of the Characteristic Value using Server Characteristic Configuration Descriptor. If set, the Server Characteristic Configuration Descriptor shall exist.
Read	0x02	If set, permits reads of the Characteristic Value using procedures defined in Section 4.8[generic-attribute-profile--gatt-.html#UUID-66cf484d-50cd-199f-dd0d-d0d02ad02ce0]
Write Without Response	0x04	If set, permit writes of the Characteristic Value without response using procedures defined in Section 4.9.1[generic-attribute-profile--gatt-.html#UUID-9f1c2e38-8fbe-f60c-d885-076707c88a43].
Write	0x08	If set, permits writes of the Characteristic Value with response using procedures defined in Section 4.9.3[generic-attribute-profile--gatt-.html#UUID-ba4b856a-6994-01e4-97f6-357f9be40990] or Section 4.9.4[generic-attribute-profile--gatt-.html#UUID-6ab738ad-6d26-1da1-7417-e83da50e90c5].
Notify	0x10	If set, permits notifications of a Characteristic Value without acknowledgment using the procedure defined in Section 4.10[generic-attribute-profile--gatt-.html#UUID-74673cc2-e704-a2fa-14bb-d175c27193ab]. If set, the Client Characteristic Configuration Descriptor shall exist.
Indicate	0x20	If set, permits indications of a Characteristic Value with acknowledgment using the procedure defined in Section 4.11[generic-attribute-profile--gatt-.html#UUID-615ed0ff-f42b-827d-6427-6474fc21737c]. If set, the Client Characteristic Configuration Descriptor shall exist.
Authenticated Signed Writes	0x40	If set, permits signed writes to the Characteristic Value using the procedure defined in Section 4.9.2[generic-attribute-profile--gatt-.html#UUID-516bd731-2079-87f9-8002-c3ca92fbed4b].
Extended Properties	0x80	If set, additional characteristic properties are defined in the Characteristic Extended Properties Descriptor defined in Section 3.3.3.1[generic-attribute-profile--gatt-.html#UUID-9154505c-6e2d-a35a-3f30-1da0383a2425]. If set, the Characteristic Extended Properties Descriptor shall exist.

Table 3.5: Characteristic Properties bit field

3.3.1.2. Characteristic Value Attribute Handle

The Characteristic Value *Attribute Handle* field is the *Attribute Handle* of the Attribute that contains the *Characteristic Value*.

3.3.1.3. Characteristic UUID

The *Characteristic UUID* field is a 16-bit Bluetooth UUID or 128-bit UUID that describes the type of *Characteristic Value*. A client shall support the use of both 16-bit and 128-bit *Characteristic UUIDs*. A client may ignore any characteristic definition with an unknown *Characteristic UUID*. An unknown characteristic UUID is a UUID for an unsupported characteristic.

3.3.2. Characteristic Value declaration

The *Characteristic Value* declaration contains the value of the characteristic. It is the first Attribute after the characteristic declaration. All characteristic definitions shall have a *Characteristic Value* declaration.

A Characteristic Value declaration is an Attribute with the Attribute Type set to the 16-bit Bluetooth or 128-bit UUID for the Characteristic Value used in the characteristic declaration. The *Attribute Value* is set to the *Characteristic Value*. The *Attribute Permissions* are specified by the service or may be implementation specific if not specified otherwise.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0xUUUU – 16-bit Bluetooth UUID or 128-bit UUID for Characteristic UUID	Characteristic Value	Higher layer profile or implementation specific

Table 3.6: Characteristic Value declaration

3.3.3. Characteristic descriptor declarations

Characteristic descriptors are used to contain related information about the *Characteristic Value*. The GATT profile defines a standard set of characteristic descriptors that can be used by higher layer profiles. Higher layer profiles may define additional characteristic descriptors that are profile specific. Each characteristic descriptor is identified by the characteristic descriptor UUID. A client shall support the use of both 16-bit and 128-bit characteristic descriptor UUIDs. A client may ignore any characteristic descriptor declaration with an unknown characteristic descriptor UUID. An unknown characteristic descriptor UUID is a UUID for an unsupported characteristic descriptor.

Characteristic descriptors if present within a characteristic definition shall follow the *Characteristic Value* declaration. The characteristic descriptor declaration may appear in any order within the characteristic definition. The client shall not assume the order in which a characteristic descriptor declaration appears in a characteristic definition following the *Characteristic Value* declaration.

Characteristic descriptor declaration permissions are defined by a higher layer profile or are implementation specific. A client shall not assume all characteristic descriptor declarations are readable.

3.3.3.1. Characteristic Extended Properties

The *Characteristic Extended Properties* declaration is a descriptor that defines additional *Characteristic Properties*. If the *Extended Properties* bit of the *Characteristic Properties* is set then this characteristic descriptor shall exist. The characteristic descriptor may occur in any position within the characteristic definition after the Characteristic Value. Only one *Characteristic Extended Properties* declaration shall exist in a characteristic definition.

The characteristic descriptor is contained in an Attribute and the *Attribute Type* shall be set to the UUID for «Characteristic Extended Properties» and the *Attribute Value* shall be two octets in length and shall contain the *Characteristic Extended Properties Bit Field*. The *Attribute Permissions* shall be readable without authentication and authorization being required.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2900 – UUID for «Characteristic Extended Properties»	Characteristic Extended Properties Bit Field	Read Only, No Authentication, No Authorization

Table 3.7: Characteristic Extended Properties declaration

The *Characteristic Extended Properties* bit field describes additional properties on how the Characteristic Value can be used, or how the characteristic descriptors (see Section 3.3.3.3[generic-attribute-profile--gatt-.html#UUID-58fcda17-4f4b-3f53-3ca8-077bbfc77c5d]) can be accessed. If the bits defined in Table 3.8[generic-attribute-profile--gatt-.html#UUID-9154505c-6e2d-a35a-3f30-1da0383a2425_table-idm13358912021912] are set, the action described is permitted. Multiple additional properties can be set.

Bit Number	Property	Description
0	Reliable Write	If set, permits reliable writes of the Characteristic Value using the procedure defined in Section 4.9.5[generic-attribute-profile--gatt-.html#UUID-7fc3dafa-7199-3f9c-a137-1575597b2a8c]
1	Writable Auxiliaries	If set, permits writes to the characteristic descriptor defined in Section 3.3.3.2[generic-attribute-profile--gatt-.html#UUID-4677edc4-2ca2-41b8-8033-9695180060a3]
All other bits		Reserved for future use

Table 3.8: Characteristic Extended Properties bit field

3.3.3.2. Characteristic User Description

The *Characteristic User Description* declaration is an optional characteristic descriptor that defines a UTF-8 string of variable size that is a user textual description of the *Characteristic Value*. If the *Writable Auxiliaries* bit of the *Characteristic Extended Properties* is set then this characteristic descriptor can be written. The characteristic descriptor may occur in any position within the characteristic definition after the *Characteristic Value*. Only one *Characteristic User Description* declaration shall exist in a characteristic definition.

The characteristic descriptor is contained in an Attribute and the *Attribute Type* shall be set to the UUID for «Characteristic User Description» and the *Attribute Value* shall be set to the characteristic user description UTF-8 string. The *Attribute Permissions* are specified by the profile or may be implementation specific if not specified otherwise.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2901 – UUID for «Characteristic User Description»	Characteristic User Description UTF-8 String	Higher layer profile or implementation specific

Table 3.9: Characteristic User Description declaration

3.3.3.3. Client Characteristic Configuration

The *Client Characteristic Configuration* declaration is an optional characteristic descriptor that defines how the characteristic may be configured by a specific client. The Client Characteristic Configuration descriptor value shall be persistent across connections for bonded devices. The Client Characteristic Configuration descriptor value shall be set to the default value at each connection with non-bonded devices.The characteristic descriptor value is a bit field. When a bit is set, that action shall be enabled, otherwise it will not be used. The *Client Characteristic Configuration* descriptor may occur in any position within the characteristic definition after the Characteristic Value. Only one *Client Characteristic Configuration* declaration shall exist in a characteristic definition.

A client may write this configuration descriptor to control the configuration of this characteristic on the server for the client. Each client has its own instantiation of the *Client Characteristic Configuration*. Reads of the *Client Characteristic Configuration* only shows the configuration for that client and writes only affect the configuration of that client. Authentication and authorization may be required by the server to write the configuration descriptor. The *Client Characteristic Configuration* declaration shall be readable and writable.

The characteristic descriptor is contained in an Attribute. The *Attribute Type* shall be set to the UUID for «Client Characteristic Configuration». The *Attribute Value* shall be two octets in length and shall be set to the characteristic descriptor value. The *Attribute Permissions* are specified by the profile or may be implementation specific if not specified otherwise.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2902 – UUID for «Client Characteristic Configuration»	Characteristic Configuration Bits	Readable with no authentication or authorization. Writable with authentication and authorization defined by a higher layer specification or is implementation specific.

Table 3.10: Client Characteristic Configuration declaration

The following Client Characteristic Configuration bits are defined:

Bit Number	Configuration	Description
0	Notification	The Characteristic Value shall be notified.This value can only be set if the characteristic's properties have the notify bit set.
1	Indication	The Characteristic Value shall be indicated. This value can only be set if the characteristic's properties have the indicate bit set.
All other bits		Reserved for future use.

Table 3.11: Client Characteristic Configuration bit field definition

The default value for the *Client Characteristic Configuration* descriptor value shall be 0x0000.

Between a client and a server there shall be a single Client Characteristic Configuration Descriptor irrespective of the number of ATT bearers between them.

3.3.3.4. Server Characteristic Configuration

The *Server Characteristic Configuration* declaration is an optional characteristic descriptor that defines how the characteristic may be configured for the server. The characteristic descriptor value is a bit field. When a bit is set, that action shall be enabled, otherwise it will not be used. The *Server Characteristic Configuration* descriptor may occur in any position within the characteristic definition after the *Characteristic Value*. Only one *Server Characteristic Configuration* declaration shall exist in a characteristic definition. The *Server Characteristic Configuration* declaration shall be readable and writable.

A client may write this configuration descriptor to control the configuration of this characteristic on the server for all clients. There is a single instantiation of the *Server Characteristic Configuration* for all clients. Reads of the *Server Characteristic Configuration* shows the configuration all clients and writes affect the configuration for all clients. Authentication and authorization may be required by the server to write the configuration descriptor.

The characteristic descriptor is contained in an Attribute. The *Attribute Type* shall be set to the UUID for «Server Characteristic Configuration». The *Attribute Value* shall be two octets in length and shall be set to the characteristic descriptor value. The *Attribute Permissions* are specified by the profile or may be implementation specific if not specified otherwise.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2903 – UUID for «Server Characteristic Configuration»	Characteristic Configuration Bits	Readable with no authentication or authorization. Writable with authentication and authorization defined by a higher layer specification or is implementation specific.

Table 3.12: Server Characteristic Configuration declaration

The following *Server Characteristic Configuration* bits are defined:

Bit Number	Configuration	Description
0	Broadcast	The Characteristic Value shall be broadcast when the server is in the broadcast procedure if advertising data resources are available. This value can only be set if the characteristic's properties have the broadcast bit set.
All other bits		Reserved for future use.

Table 3.13: Server Characteristic Configuration bit field definition

3.3.3.5. Characteristic Presentation Format

The *Characteristic Presentation Format* declaration is an optional characteristic descriptor that defines the format of the *Characteristic Value*. The characteristic descriptor may occur in any position within the characteristic definition after the *Characteristic Value*. If more than one *Characteristic Presentation Format* declaration exists in a characteristic definition, then a *Characteristic Aggregate Format* declaration shall exist as part of the characteristic definition.

The characteristic presentation format value is composed of five parts: format, exponent, unit, name space, and description.

The characteristic descriptor is contained in an Attribute. The *Attribute Type* shall be set to the UUID for «Characteristic Presentation Format». The *Attribute Value* shall be set to the characteristic descriptor value. The *Attribute Permissions* shall be read only and not require authentication or authorization.

Attribute Handle	Attribute Type	Attribute Value					Attribute Permissions
0xNNNN	0x2904 – UUID for «Characteristic Presentation Format»	Format	Exponent	Unit	Name Space	Description	Read only No Authentication, No Authorization

Table 3.14: Characteristic Presentation Format declaration

The definition of the Characteristic Presentation Format descriptor Attribute Value field is the following.

Field Name	Value Size	Description
Format	1 octet	Format of the value of this characteristic as defined in [1[generic-attribute-profile--gatt-.html#UUID-c32d82db-0074-99ba-c52a-307a41944215_bibliomixed-idm13391355391050]].
Exponent	1 octet	Exponent field to determine how the value of this characteristic is further formatted.
Unit	2 octets	The unit of this characteristic as defined in [1[generic-attribute-profile--gatt-.html#UUID-c32d82db-0074-99ba-c52a-307a41944215_bibliomixed-idm13391355391050]]
Name Space	1 octet	The name space of the description as defined in [1[generic-attribute-profile--gatt-.html#UUID-c32d82db-0074-99ba-c52a-307a41944215_bibliomixed-idm13391355391050]]
Description	2 octets	The description of this characteristic as defined in a higher layer profile.

Table 3.15: Characteristic Presentation Format value definition

3.3.3.5.1. Bit ordering

The bit ordering used for the Characteristic Presentation Format descriptor shall be little-endian.

3.3.3.5.2. Format

The format field determines how a single value contained in the *Characteristic Value* is formatted. The values of this field are defined in Assigned Numbers [1[generic-attribute-profile--gatt-.html#UUID-c32d82db-0074-99ba-c52a-307a41944215_bibliomixed-idm13391355391050]].

3.3.3.5.3. Exponent

The exponent field is used with integer data types to determine how the value is further formatted. The exponent field is only used on integer format types. The exponent field has type sint8.

actual value = *Characteristic Value* * 10^{Exponent}

As can be seen in the above equation, the actual value is a combination of the Characteristic Value and the value 10 to the power Exponent. This is sometimes known as a fixed point number.

For example, if the Exponent is 2 and the *Characteristic Value* is 23, the actual value would be 2300.

For example, if the Exponent is -3 and the *Characteristic Value* is 3892, the actual value would be 3.892.

3.3.3.5.4. Unit

The Unit is a UUID as defined in Assigned Numbers[https://www.bluetooth.com/specifications/assigned-numbers] [1[generic-attribute-profile--gatt-.html#UUID-c32d82db-0074-99ba-c52a-307a41944215_bibliomixed-idm13391355391050]].

3.3.3.5.5. Name Space

The Name Space field is used to identify the organization, as defined in Assigned Numbers[https://www.bluetooth.com/specifications/assigned-numbers] [1[generic-attribute-profile--gatt-.html#UUID-c32d82db-0074-99ba-c52a-307a41944215_bibliomixed-idm13391355391050]], that is responsible for defining the enumerations for the description field.

3.3.3.5.6. Description

The Description is an enumerated value as defined in Assigned Numbers[https://www.bluetooth.com/specifications/assigned-numbers] [1[generic-attribute-profile--gatt-.html#UUID-c32d82db-0074-99ba-c52a-307a41944215_bibliomixed-idm13391355391050]] from the organization identified by the Name Space field.

3.3.3.6. Characteristic Aggregate Format

The *Characteristic Aggregate Format* declaration is an optional characteristic descriptor that defines the format of an aggregated *Characteristic Value*.

The characteristic descriptor may occur in any position within the characteristic definition after the *Characteristic Value*. Only one *Characteristic Aggregate Format* declaration shall exist in a characteristic definition.

The Characteristic Aggregate Format value is composed of a list of Attribute Handles of *Characteristic Presentation Format* declarations, where each *Attribute Handle* points to a *Characteristic Presentation Format* declaration.

The *Attribute Permissions* shall be read only and not require authentication or authorization.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2905 – UUID for «Characteristic Aggregate Format»	List of *Attribute Handles* for the Characteristic Presentation Format Declarations	Read only No authentication No authorization

Table 3.16: Characteristic Aggregate Format declaration

The *List of Attribute Handles* is the concatenation of multiple 16-bit *Attribute Handle* values into a single *Attribute Value*. The list shall contain at least two *Attribute Handle for Characteristic Presentation Format declarations*. The Characteristic Value shall be decomposed by each of the *Characteristic Presentation Format* declarations pointed to by the *Attribute Handles*. The order of the *Attribute Handles* in the list is significant.

If more than one *Characteristic Presentation Format* declarations exist in a characteristic definition, there shall also be one Characteristic Aggregate Format declaration. The *Characteristic Aggregate Format* declaration shall include each *Characteristic Presentation Format* declaration in the characteristic definition in the list of *Attribute Handles*. Characteristic Presentation Format declarations from other characteristic definitions may also be used.

A *Characteristic Aggregate Format* declaration may exist without a *Characteristic Presentation Format* declaration existing in the characteristic definition. The *Characteristic Aggregate Format* declaration may use *Characteristic Presentation Format* declarations from other characteristic definitions.

3.4. Summary of GATT Profile attribute types

Table 3.17[generic-attribute-profile--gatt-.html#UUID-1006e684-4a22-9c69-642a-876c98793a2f_table-idm13358912381556] summarizes the attribute types defined by the GATT Profile.

Attribute Type	UUID	Description
«Primary Service»	0x2800	Primary Service Declaration
«Secondary Service»	0x2801	Secondary Service Declaration
«Include»	0x2802	Include Declaration
«Characteristic»	0x2803	Characteristic Declaration
«Characteristic Extended Properties»	0x2900	Characteristic Extended Properties
«Characteristic User Description»	0x2901	Characteristic User Description Descriptor
«Client Characteristic Configuration»	0x2902	Client Characteristic Configuration Descriptor
«Server Characteristic Configuration»	0x2903	Server Characteristic Configuration Descriptor
«Characteristic Presentation Format»	0x2904	Characteristic Presentation Format Descriptor
«Characteristic Aggregate Format»	0x2905	Characteristic Aggregate Format Descriptor

Table 3.17: Summary of GATT Profile attribute types

4. GATT feature requirements

4.1. Overview

There are 11 features defined in the GATT Profile:

1. Server Configuration
2. Primary Service Discovery
3. Relationship Discovery
4. Characteristic Discovery
5. Characteristic Descriptor Discovery
6. Reading a Characteristic Value
7. Writing a Characteristic Value
8. Notification of a Characteristic Value
9. Indication of a Characteristic Value
10. Reading a Characteristic Descriptor
11. Writing a Characteristic Descriptor

Each of the features is mapped to procedures and sub-procedures. These procedures and sub-procedures describe how the Attribute Protocol is used to accomplish the corresponding feature.

4.2. Feature support and procedure mapping

Table 4.1[generic-attribute-profile--gatt-.html#UUID-d8c3186f-a603-00e3-8447-a3b9def32039_table-idm13358912569806] maps each feature to the procedures used for that feature, and indicates whether the procedure is optional or mandatory for that feature. The procedures are described in the referenced section.

If an ATT PDU is supported on any ATT bearer, then it shall be supported on all supported ATT bearers with the following exceptions:

- The Exchange MTU sub-procedure shall only be supported on the LE Fixed Channel Unenhanced ATT bearer.
- The Signed Write Without Response sub-procedure shall only be supported on the LE Fixed Channel Unenhanced ATT bearer.

Feature	Sub-Procedure	Ref.	Support in Client	Support in Server
Server Configuration	Exchange MTU	4.3.1[generic-attribute-profile--gatt-.html#UUID-68c91ae1-e9e9-f48a-a0ba-d712edf16172]	O	O
Primary Service Discovery	Discover All Primary Services	4.4.1[generic-attribute-profile--gatt-.html#UUID-8db5587d-f49d-6c82-73f3-23246f2dc4f2]	O	M
	Discover Primary Services By Service UUID	4.4.2[generic-attribute-profile--gatt-.html#UUID-ab366af5-af65-6f9c-d957-0b8e7c905fa8]	O	M
Relationship Discovery	Find Included Services	4.5.1[generic-attribute-profile--gatt-.html#UUID-257d4215-8c7b-6163-ceec-b86a74e62328]	O	M
Characteristic Discovery	Discover All Characteristic of a Service	4.6.1[generic-attribute-profile--gatt-.html#UUID-acab67bd-4175-9a3e-c37e-8ae415495349]	O	M
	Discover Characteristic by UUID	4.6.2[generic-attribute-profile--gatt-.html#UUID-b676406c-3e78-696a-29e8-e0f025f34426]	O	M

Feature	Sub-Procedure	Ref.	Support in Client	Support in Server
Characteristic Descriptor Discovery	Discover All Characteristic Descriptors	4.7.1[generic-attribute-profile--gatt-.html#UUID-ca708893-b226-d29d-7308-bece78a8fe9c]	O	M
Characteristic Value Read	Read Characteristic Value	4.8.1[generic-attribute-profile--gatt-.html#UUID-90b3e212-b224-6a3f-8f2e-e6fb63641e31]	O	M
	Read Using Characteristic UUID	4.8.2[generic-attribute-profile--gatt-.html#UUID-3569a57b-2934-e1bd-9527-3d591375eb8d]	O	M
	Read Long Characteristic Values	4.8.3[generic-attribute-profile--gatt-.html#UUID-0a5cb6bc-58f4-5a47-2c66-889a1bc466d1]	O	C.4
	Read Multiple Characteristic Values	4.8.4[generic-attribute-profile--gatt-.html#UUID-2bf8cd06-0fa3-0d4b-1989-c622b8af05c2]	O	O
	Read Multiple Variable Length Characteristic Values	4.8.5[generic-attribute-profile--gatt-.html#UUID-7189d557-5507-079f-b195-16a0fef3013e]	O	C.4
Characteristic Value Write	Write Without Response	4.9.1[generic-attribute-profile--gatt-.html#UUID-9f1c2e38-8fbe-f60c-d885-076707c88a43]	O	C.1
	Signed Write Without Response	4.9.2[generic-attribute-profile--gatt-.html#UUID-516bd731-2079-87f9-8002-c3ca92fbed4b]	O	O
	Write Characteristic Value	4.9.3[generic-attribute-profile--gatt-.html#UUID-ba4b856a-6994-01e4-97f6-357f9be40990]	O	C.2
	Write Long Characteristic Values	4.9.4[generic-attribute-profile--gatt-.html#UUID-6ab738ad-6d26-1da1-7417-e83da50e90c5]	O	C.4
	Characteristic Value Reliable Writes	4.9.5[generic-attribute-profile--gatt-.html#UUID-7fc3dafa-7199-3f9c-a137-1575597b2a8c]	O	O
Characteristic Value Notifications	Single Notifications	4.10.1[generic-attribute-profile--gatt-.html#UUID-52ccee7c-a90c-b3d8-7173-5a34a371218d]	C.4	C.4
	Multiple Variable Length Notifications	4.10.2[generic-attribute-profile--gatt-.html#UUID-ef60fab0-0495-034a-7f7d-574b6721bb69]	C.4	C.4
Characteristic Value Indication	Indications	4.11.1[generic-attribute-profile--gatt-.html#UUID-a5e37122-a510-899c-9c4d-a54fe324a7d8]	M	C.3

Feature	Sub-Procedure	Ref.	Support in Client	Support in Server
Characteristic Descriptor Value Read	Read Characteristic Descriptors	4.12.1[generic-attribute-profile--gatt-.html#UUID-5a1a9293-614a-f4ed-7771-fd8b4143d076]	O	C.4
	Read Long Characteristic Descriptors	4.12.2[generic-attribute-profile--gatt-.html#UUID-84794b9f-7220-7665-0f14-b7a462da94b7]	O	C.4
Characteristic Descriptor Value Write	Write Characteristic Descriptors	4.12.3[generic-attribute-profile--gatt-.html#UUID-0b0e2ea7-b310-8201-4ff4-cc43787afe88]	O	C.4
	Write Long Characteristic Descriptors	4.12.4[generic-attribute-profile--gatt-.html#UUID-3e9d6b9c-cd01-2b51-a3d3-47741f30f086]	O	O
C.1: Write Without Response is mandatory if Signed Write Without Response or Enhanced ATT Bearers are supported otherwise optional C.2: Write Characteristic Value is mandatory if Write Long Characteristic Values or Enhanced ATT Bearers are supported otherwise optional C.3: If *Service Changed Characteristic* is present, this feature is mandatory, otherwise optional. C.4: If Enhanced ATT Bearers are supported then this feature is mandatory, otherwise optional.				

Table 4.1: GATT feature mapping to procedures

4.3. Server configuration

This procedure is used by the client to configure the Attribute Protocol. This procedure has only one sub-procedure used to set the MTU sizes.

4.3.1. Exchange MTU

This sub-procedure is used by the client to set the ATT_MTU to the maximum possible value that can be supported by both devices when the client supports a value greater than the default ATT_MTU for the Attribute Protocol. This sub-procedure shall only be initiated once during a connection.

This sub-procedure shall not be used on a BR/EDR physical link since the MTU size is negotiated using L2CAP channel configuration procedures.

The ATT_EXCHANGE_MTU_REQ PDU is used by this sub-procedure. The Client Rx MTU parameter shall be set to the maximum MTU that this client can receive.

Two possible responses can be sent from the server for the ATT_EXCHANGE_MTU_REQ PDU: ATT_EXCHANGE_MTU_-RSP and ATT_ERROR_RSP PDUs.

An ATT_ERROR_RSP PDU is returned if an error occurred on the server.

The server shall respond to this message with an ATT_EXCHANGE_MTU_RSP PDU with the Server Rx MTU parameter set to the maximum MTU that this server can receive.

If the ATT_ERROR_RSP PDU is sent by the server with the Error Code parameter set to *Request Not Supported* (0x06), the *Attribute Opcode* is not supported and the default MTU shall be used.

Once the messages have been exchanged, the ATT_MTU shall be set to the minimum of the Client Rx MTU and Server Rx MTU values.

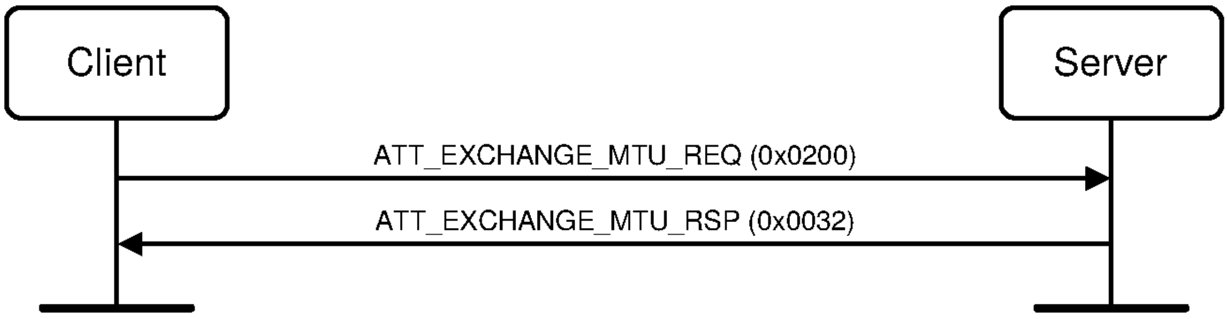


Figure 4.1: Exchange MTU

For example, in Figure 4.1[generic-attribute-profile--gatt-.html#UUID-68c91ae1-e9e9-f48a-a0ba-d712edf16172_figure-idm4508319755326433598988156958], based on the exchanged ATT_MTU values, the ATT_MTU would be 0x0032.

4.4. Primary Service Discovery

This procedure is used by a client to discover primary services on a server. Once the primary services are discovered, additional information about the primary services can be accessed using other procedures, including characteristic discovery and relationship discovery to find other related primary and secondary services.

There are two sub-procedures that can be used for primary service discovery: Discover All Primary Services and Discover Primary Services by Service UUID.

4.4.1. Discover All Primary Services

This sub-procedure is used by a client to discover all the primary services on a server.

The ATT_READ_BY_GROUP_TYPE_REQ PDU shall be used with the Attribute Type parameter set to the UUID for «Primary Service». The *Starting Handle* shall be set to 0x0001 and the *Ending Handle* shall be set to 0xFFFF.

Two possible responses can be sent from the server for the ATT_READ_BY_GROUP_TYPE_REQ PDU: ATT_READ_BY_-GROUP_TYPE_RSP and ATT_ERROR_RSP PDUs.

An ATT_ERROR_RSP PDU is returned if an error occurred on the server.

The ATT_READ_BY_GROUP_TYPE_RSP PDU returns a list of *Attribute Handle*, *End Group Handle*, and *Attribute Value* tuples corresponding to the services supported by the server. Each *Attribute Value* contained in the response is the Service UUID of a service supported by the server. The *Attribute Handle* is the handle for the service declaration. The *End Group Handle* is the handle of the last attribute within the service definition. The *End Group Handle* of the last service in a device can be 0xFFFF. The ATT_READ_BY_GROUP_TYPE_REQ PDU shall be issued again with the *Starting Handle* set to one greater than the last *End Group Handle* in the ATT_READ_BY_GROUP_TYPE_RSP PDU.

This sub-procedure is complete when the ATT_ERROR_RSP PDU is received and the Error Code parameter is set to *Attribute Not Found* (0x0A) or when the *End Group Handle* in the *Read by Type Group Response* is 0xFFFF.

The sub-procedure may end early if a desired primary service is found prior to discovering all the primary services on the server.

The service declaration described in Section 3.1[generic-attribute-profile--gatt-.html#UUID-2148a3e6-91f7-e758-750f-8b14377cab6e] specifies that the service declaration is readable and requires no authentication or authorization, therefore insufficient authentication or read not permitted errors shall not occur.

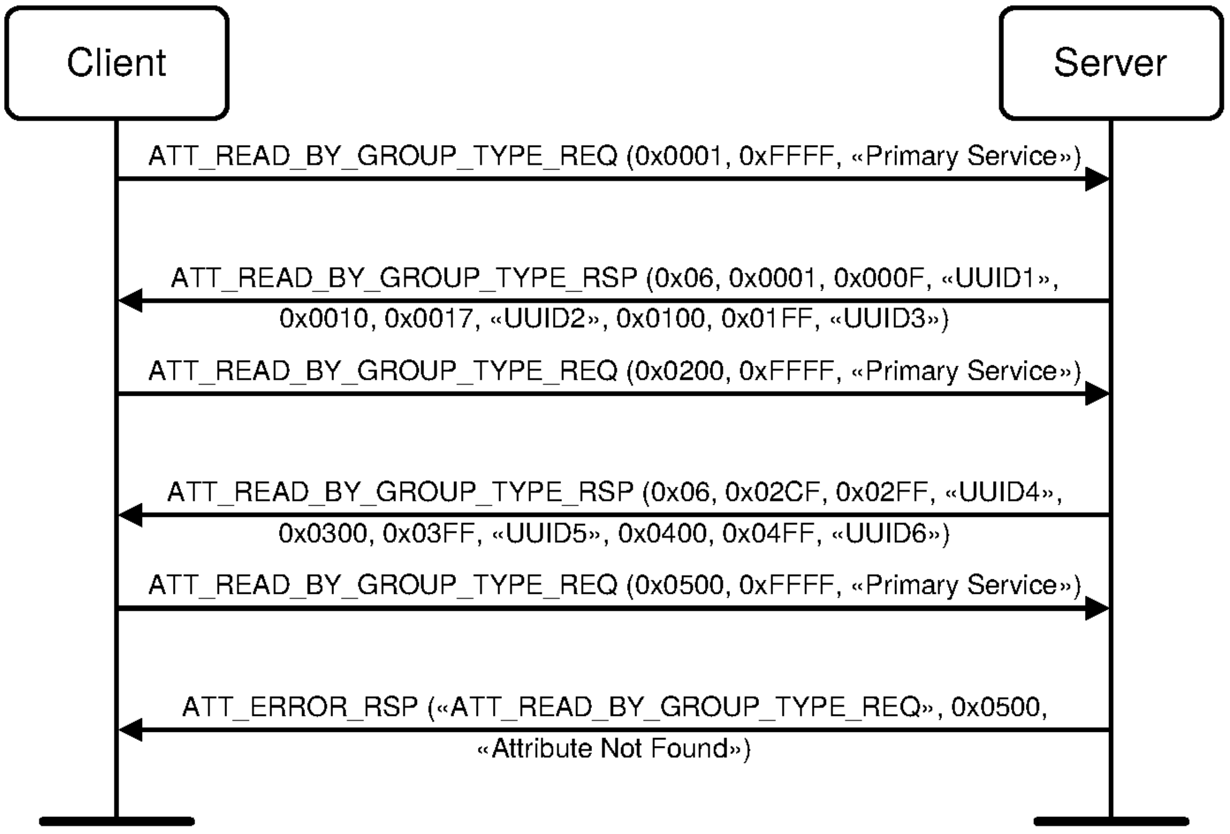

Client
Server
ATT_READ_BY_GROUP_TYPE_REQ (0x0001, 0xFFFF, «Primary Service»)
ATT_READ_BY_GROUP_TYPE_RSP (0x06, 0x0001, 0x000F, «UUID1», 0x0010, 0x0017, «UUID2», 0x0100, 0x01FF, «UUID3»)
ATT_READ_BY_GROUP_TYPE_REQ (0x0200, 0xFFFF, «Primary Service»)
ATT_READ_BY_GROUP_TYPE_RSP (0x06, 0x02CF, 0x02FF, «UUID4», 0x0300, 0x03FF, «UUID5», 0x0400, 0x04FF, «UUID6»)
ATT_READ_BY_GROUP_TYPE_REQ (0x0500, 0xFFFF, «Primary Service»)
ATT_ERROR_RSP («ATT_READ_BY_GROUP_TYPE_REQ», 0x0500, «Attribute Not Found»)

Figure 4.2: Discover All Primary Services example

4.4.2. Discover Primary Service by Service UUID

This sub-procedure is used by a client to discover a specific primary service on a server when only the Service UUID is known. The specific primary service may exist multiple times on a server. The primary service being discovered is identified by the service UUID.

The ATT_FIND_BY_TYPE_VALUE_REQ PDU shall be used with the Attribute Type parameter set to the UUID for «Primary Service» and the *Attribute Value* set to the 16-bit Bluetooth UUID or 128-bit UUID for the specific primary service. The *Starting Handle* shall be set to 0x0001 and the *Ending Handle* shall be set to 0xFFFF.

Two possible responses can be sent from the server for the ATT_FIND_BY_TYPE_VALUE_REQ PDU: ATT_FIND_BY_TYPE_VALUE_RSP and ATT_ERROR_RSP PDUs.

An ATT_ERROR_RSP PDU is returned if an error occurred on the server.

The ATT_FIND_BY_TYPE_VALUE_RSP PDU returns a list of *Attribute Handle* ranges. The *Attribute Handle* range is the starting handle and the ending handle of the service definition. The *End Group Handle* of the last service in a device can be 0xFFFF. If the *Attribute Handle* range for the Service UUID being searched is returned and the End Found Handle is not 0xFFFF, the ATT_FIND_BY_TYPE_VALUE_REQ PDU may be issued again with the *Starting Handle* set to one greater than the last *Attribute Handle* range in the ATT_FIND_BY_TYPE_VALUE_RSP PDU.

This sub-procedure is complete when the ATT_ERROR_RSP PDU is received and the Error Code parameter is set to *Attribute Not Found* (0x0A) or when the *End Group Handle* in the ATT_FIND_BY_TYPE_VALUE_RSP PDU is 0xFFFF.

The sub-procedure may end early if a desired primary service is found prior to discovering all the primary services of the specified service UUID supported on the server.

The service declaration described in Section 3.1[generic-attribute-profile--gatt-.html#UUID-2148a3e6-91f7-e758-750f-8b14377cab6e] specifies that the service declaration is readable and requires no authentication or authorization, therefore insufficient authentication or read not permitted errors shall not occur.

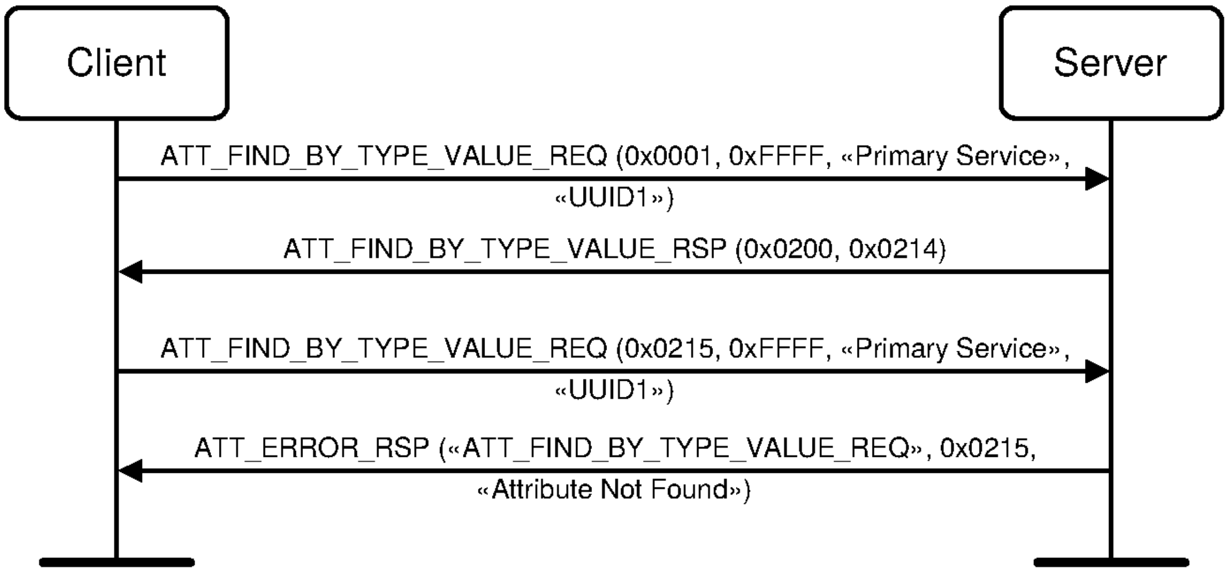

Client
Server
ATT_FIND_BY_TYPE_VALUE_REQ (0x0001, 0xFFFF, «Primary Service», «UUID1»)
ATT_FIND_BY_TYPE_VALUE_RSP (0x0200, 0x0214)
ATT_FIND_BY_TYPE_VALUE_REQ (0x0215, 0xFFFF, «Primary Service», «UUID1»)
ATT_ERROR_RSP («ATT_FIND_BY_TYPE_VALUE_REQ», 0x0215, «Attribute Not Found»)

Figure 4.3: Discover Primary Service by Service UUID example

4.5. Relationship Discovery

This procedure is used by a client to discover service relationships to other services.

There is one sub-procedure that can be used for relationship discovery: Find Included Services.

4.5.1. Find Included Services

This sub-procedure is used by a client to find include service declarations within a service definition on a server. The service specified is identified by the service handle range.

The ATT_READ_BY_TYPE_REQ PDU shall be used with the *Attribute Type* parameter set to the UUID for «Include» The *Starting Handle* shall be set to the starting handle of the specified service and the *Ending Handle* shall be set to the ending handle of the specified service. The sub-procedure may end early if a desired included service is found prior to discovering all the included services of the specified service supported on the server.

Two possible responses can be sent from the server for the ATT_READ_BY_TYPE_REQ PDU: ATT_READ_BY_TYPE_RSP and ATT_ERROR_RSP PDUs.

An ATT_ERROR_RSP PDU is returned if an error occurred on the server.

The ATT_READ_BY_TYPE_RSP PDU returns a set of *Attribute Handle* and *Attribute Value* pairs corresponding to the included services in the service definition. Each *Attribute Value* contained in the response is composed of the *Attribute Handle* of the included service declaration and the *End Group Handle*. If the service UUID is a 16-bit Bluetooth UUID it is also returned in the response. The ATT_READ_BY_TYPE_REQ PDU shall be issued again with the *Starting Handle* set to one greater than the last *Attribute Handle* in the ATT_READ_BY_TYPE_RSP PDU.

The sub-procedure is complete when either the ATT_ERROR_RSP PDU is received with the Error Code parameter set to *Attribute Not Found* (0x0A) or the ATT_READ_BY_TYPE_RSP PDU has an *Attribute Handle* of the included service declaration that is equal to the *Ending Handle* of the request.

To get the included service UUID when the included service uses a 128-bit UUID, the ATT_READ_REQ PDU is used. The *Attribute Handle* for the ATT_READ_REQ PDU is the *Attribute Handle* of the included service.

The include declaration described in Section 3.2[generic-attribute-profile--gatt-.html#UUID-a4a91ea7-a00f-77b6-9b37-fa0492451a21] specifies that the include declaration is readable and requires no authentication or authorization, therefore insufficient authentication or read not permitted errors shall not occur.

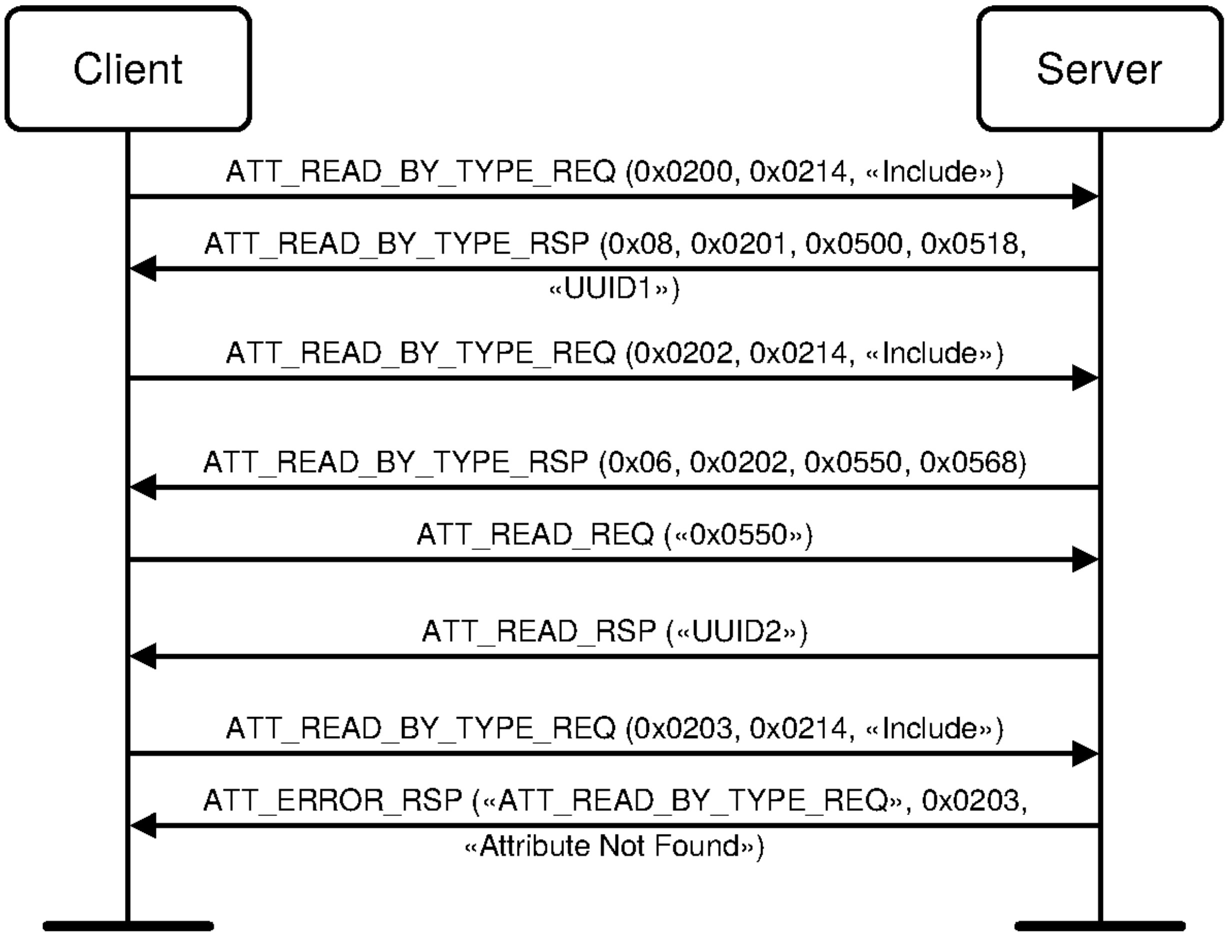

Client
Server
ATT_READ_BY_TYPE_REQ (0x0200, 0x0214, «Include»)
ATT_READ_BY_TYPE_RSP (0x08, 0x0201, 0x0500, 0x0518, «UUID1»)
ATT_READ_BY_TYPE_REQ (0x0202, 0x0214, «Include»)
ATT_READ_BY_TYPE_RSP (0x06, 0x0202, 0x0550, 0x0568)
ATT_READ_REQ («0x0550»)
ATT_READ_RSP («UUID2»)
ATT_READ_BY_TYPE_REQ (0x0203, 0x0214, «Include»)
ATT_ERROR_RSP («ATT_READ_BY_TYPE_REQ», 0x0203, «Attribute Not Found»)

Figure 4.4: Find Included Services example

4.6. Characteristic discovery

This procedure is used by a client to discover service characteristics on a server. Once the characteristics are discovered additional information about the characteristics can be discovered or accessed using other procedures.

There are two sub-procedures that can be used for characteristic discovery: Discover All Characteristics of a Service and Discover Characteristics by UUID.

4.6.1. Discover All Characteristics of a Service

This sub-procedure is used by a client to find all the characteristic declarations within a service definition on a server when only the service handle range is known. The service specified is identified by the service handle range.

The ATT_READ_BY_TYPE_REQ PDU shall be used with the *Attribute Type* parameter set to the UUID for «Characteristic» The *Starting Handle* shall be set to starting handle of the specified service and the *Ending Handle* shall be set to the ending handle of the specified service.

Two possible responses can be sent from the server for the ATT_READ_BY_TYPE_REQ PDU: ATT_READ_BY_TYPE_RSP and ATT_ERROR_RSP PDUs.

An ATT_ERROR_RSP PDU is returned if an error occurred on the server.

The ATT_READ_BY_TYPE_RSP PDU returns a list of *Attribute Handle* and *Attribute Value* pairs corresponding to the characteristics in the service definition. The *Attribute Handle* is the handle for the characteristic declaration. The *Attribute Value* is the Characteristic Properties, Characteristic Value Handle and Characteristic UUID. The ATT_READ_BY_TYPE_REQ PDU shall be issued again with the *Starting Handle* set to one greater than the last *Attribute Handle* in the ATT_READ_BY_TYPE_RSP PDU.

The sub-procedure is complete when the ATT_ERROR_RSP PDU is received and the Error Code parameter is set to *Attribute Not Found* (0x0A) or the ATT_READ_BY_TYPE_RSP PDU has an *Attribute Handle* that is equal to the *Ending Handle* of the request.

The sub-procedure may end early if a desired characteristic is found prior to discovering all the characteristics of the specified service supported on the server.

Note: The characteristic declaration described in Section 3.3[generic-attribute-profile--gatt-.html#UUID-70d11f51-12cd-57a4-184a-fd8a4e0283f9] specifies that the characteristic declaration is readable and requires no authentication or authorization, therefore insufficient authentication or read not permitted errors should not occur.

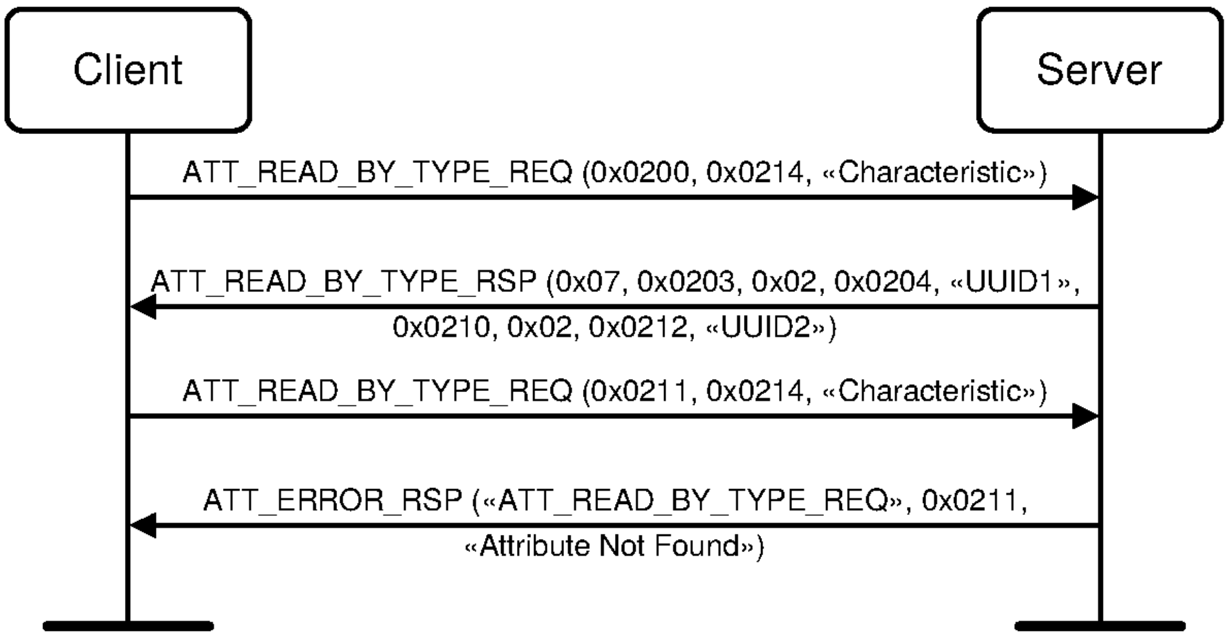

Client
Server
ATT_READ_BY_TYPE_REQ (0x0200, 0x0214, «Characteristic»)
ATT_READ_BY_TYPE_RSP (0x07, 0x0203, 0x02, 0x0204, «UUID1», 0x0210, 0x02, 0x0212, «UUID2»)
ATT_READ_BY_TYPE_REQ (0x0211, 0x0214, «Characteristic»)
ATT_ERROR_RSP («ATT_READ_BY_TYPE_REQ», 0x0211, «Attribute Not Found»)

Figure 4.5: Discover All Characteristics of a Service example

Note: In this example «UUID1» and «UUID2» are 16 bits (2 octets).
If they were 128 bits (16 octets) then the ATT_READ_BY_TYPE_RSP PDU data would instead be:
0x15, 0x0203, 0x02, 0x0204, «UUID1», 0x0210, 0x02, 0x0212, «UUID2»

4.6.2. Discover Characteristics by UUID

This sub-procedure is used by a client to discover service characteristics on a server when only the service handle ranges are known and the characteristic UUID is known. The specific service may exist multiple times on a server. The characteristic being discovered is identified by the characteristic UUID.

The ATT_READ_BY_TYPE_REQ PDU is used to perform the beginning of the sub-procedure. The *Attribute Type* is set to the UUID for «Characteristic» and the *Starting Handle* and *Ending Handle* parameters shall be set to the service handle range.

Two possible responses can be sent from the server for the ATT_READ_BY_TYPE_REQ PDU: ATT_READ_BY_TYPE_RSP and ATT_ERROR_RSP PDUs.

An ATT_ERROR_RSP PDU is returned if an error occurred on the server.

The ATT_READ_BY_TYPE_RSP PDU returns a list of *Attribute Handle* and *Attribute Value* pairs corresponding to the characteristics contained in the handle range provided. Each *Attribute Value* in the list is the *Attribute Value* for the characteristic declaration. The *Attribute Value* contains the characteristic properties, Characteristic *Value Handle* and characteristic UUID. The *Attribute Value* for each *Attribute Handle* and *Attribute Value* pairs are checked for a matching characteristic UUID. Once found, the sub-procedure continues until the end of the service handle range is exhausted. The ATT_READ_BY_TYPE_REQ PDU is issued again with the *Starting Handle* set to one greater than the last *Attribute Handle* in the ATT_READ_BY_TYPE_RSP PDU.

If the ATT_ERROR_RSP PDU is sent by the server with the Error Code parameter set to *Attribute Not Found* (0x0A), the characteristic does not exist on the server within the handle range provided.

The sub-procedure may end early if a desired characteristic is found prior to discovering all the characteristics for the specified service supported on the server.

The characteristic declaration described in Section 3.3[generic-attribute-profile--gatt-.html#UUID-70d11f51-12cd-57a4-184a-fd8a4e0283f9] specifies that the characteristic declaration is readable and requires no authentication or authorization, therefore insufficient authentication or read not permitted errors shall not occur.

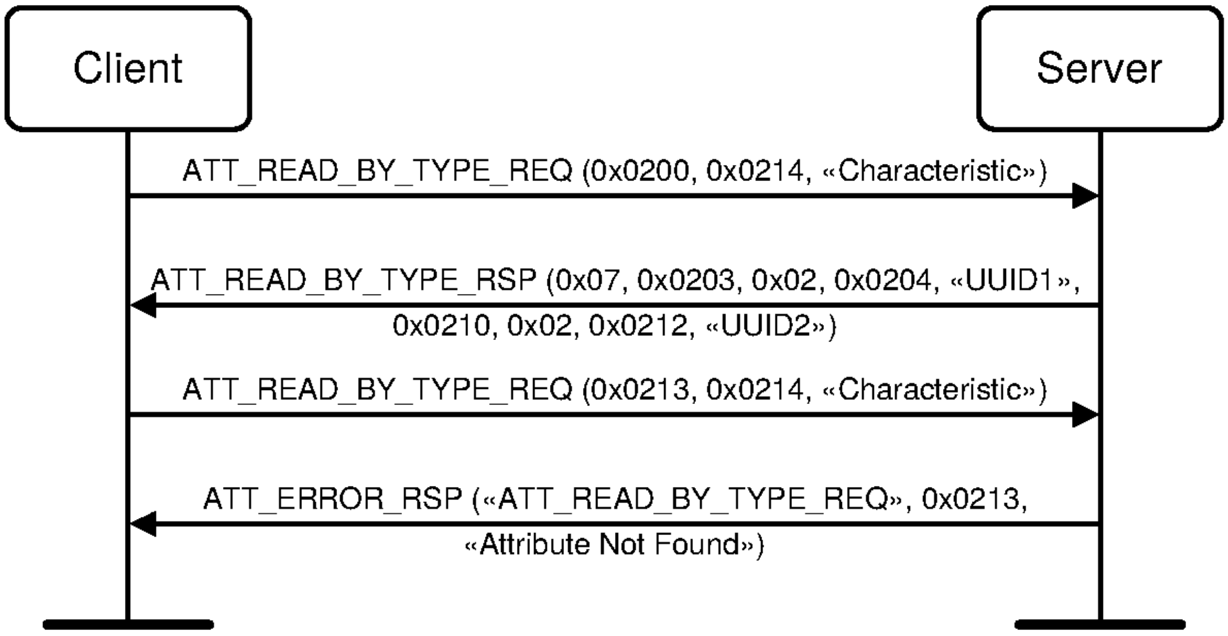

Client
Server
ATT_READ_BY_TYPE_REQ (0x0200, 0x0214, «Characteristic»)
ATT_READ_BY_TYPE_RSP (0x07, 0x0203, 0x02, 0x0204, «UUID1», 0x0210, 0x02, 0x0212, «UUID2»)
ATT_READ_BY_TYPE_REQ (0x0213, 0x0214, «Characteristic»)
ATT_ERROR_RSP («ATT_READ_BY_TYPE_REQ», 0x0213, «Attribute Not Found»)

Figure 4.6: Discover Characteristics by UUID example

4.7. Characteristic Descriptor Discovery

This procedure is used by a client to discover characteristic descriptors of a characteristic. Once the characteristic descriptors are discovered additional information about the characteristic descriptors can be accessed using other procedures.

There is one sub-procedure that can be used for characteristic descriptor discovery: Discover All Characteristic Descriptors.

4.7.1. Discover All Characteristic Descriptors

This sub-procedure is used by a client to find all the characteristic descriptor’s Attribute Handles and Attribute Types within a characteristic definition when only the characteristic handle range is known. The characteristic specified is identified by the characteristic handle range.

The ATT_FIND_INFORMATION_REQ PDU shall be used with the Starting Handle set to the handle of the specified characteristic value + 1 and the *Ending Handle* set to the ending handle of the specified characteristic.

Two possible responses can be sent from the server for the ATT_FIND_INFORMATION_REQ PDU: ATT_FIND_INFORMATION_RSP and ATT_ERROR_RSP PDUs.

An ATT_ERROR_RSP PDU is returned if an error occurred on the server.

The ATT_FIND_INFORMATION_RSP PDU returns a list of *Attribute Handle* and *Attribute Type* pairs corresponding to the characteristic descriptors in the characteristic definition. The *Attribute Handle* is the handle for the characteristic descriptor declaration. The *Attribute Type* is the characteristic descriptor UUID. The ATT_FIND_INFORMATION_REQ PDU shall be issued again with the *Starting Handle* set to one greater than the last *Attribute Handle* in the ATT_FIND_INFORMATION_RSP PDU.

The sub-procedure is complete when the ATT_ERROR_RSP PDU is received and the Error Code parameter is set to *Attribute Not Found* (0x0A) or the ATT_FIND_INFORMATION_RSP PDU has an *Attribute Handle* that is equal to the *Ending Handle* of the request.

The sub-procedure may end early if a desired Characteristic Descriptor is found prior to discovering all the characteristic descriptors of the specified characteristic.

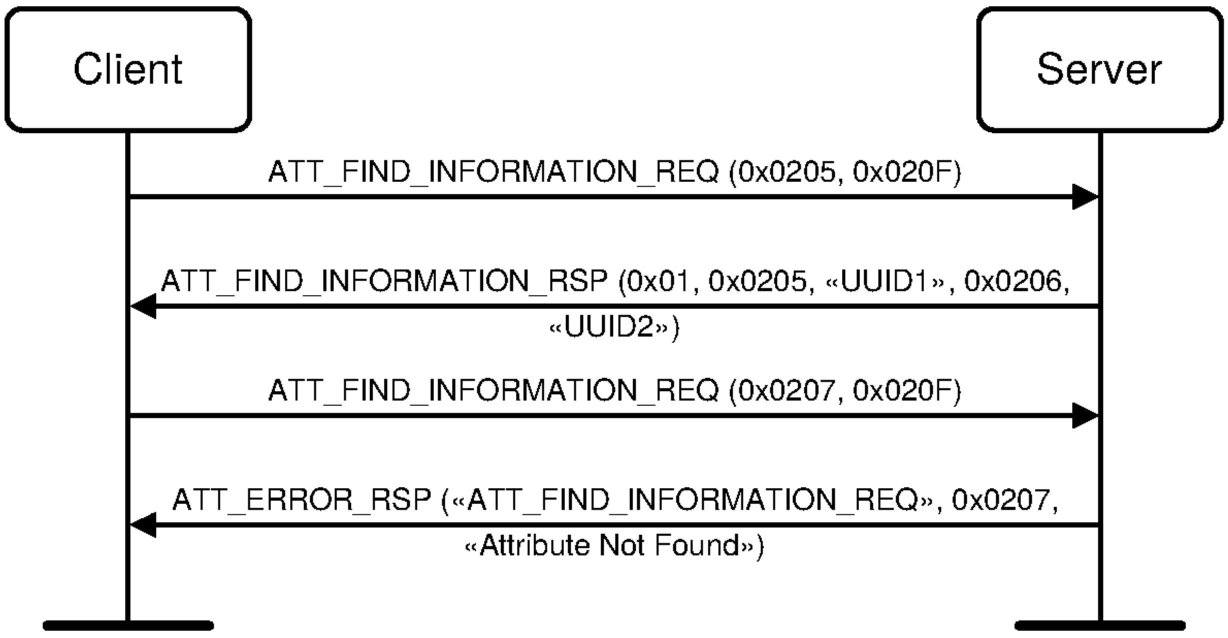

Client
Server
ATT_FIND_INFORMATION_REQ (0x0205, 0x020F)
ATT_FIND_INFORMATION_RSP (0x01, 0x0205, «UUID1», 0x0206, «UUID2»)
ATT_FIND_INFORMATION_REQ (0x0207, 0x020F)
ATT_ERROR_RSP («ATT_FIND_INFORMATION_REQ», 0x0207, «Attribute Not Found»)

Figure 4.7: Discover All Characteristic Descriptors example

4.8. Characteristic Value Read

This procedure is used to read a *Characteristic Value* from a server. There are four sub-procedures that can be used to read a *Characteristic Value*: Read Characteristic Value, Read Using Characteristic UUID, Read Long Characteristic Values, and Read Multiple Characteristic Values.

4.8.1. Read Characteristic Value

This sub-procedure is used to read a *Characteristic Value* from a server when the client knows the *Characteristic Value Handle*. The ATT_READ_REQ PDU is used with the *Attribute Handle* parameter set to the *Characteristic Value Handle*. The ATT_READ_RSP PDU returns the *Characteristic Value* in the *Attribute Value* parameter.

The ATT_READ_RSP PDU only contains the complete *Characteristic Value* if that is less than or equal to (ATT_MTU – 1) octets in length. If the *Characteristic Value* is greater than (ATT_MTU – 1) octets in length, the ATT_READ_RSP PDU only contains the first portion of the *Characteristic Value* and the *Read Long Characteristic Value* procedure may be used if the rest is required.

An ATT_ERROR_RSP PDU shall be sent by the server in response to the ATT_READ_REQ PDU if insufficient authentication, insufficient authorization, a too-short encryption key size is used by the client, or if a read operation is not permitted on the *Characteristic Value*. The *Error Code* parameter is set as specified in the Attribute Protocol.

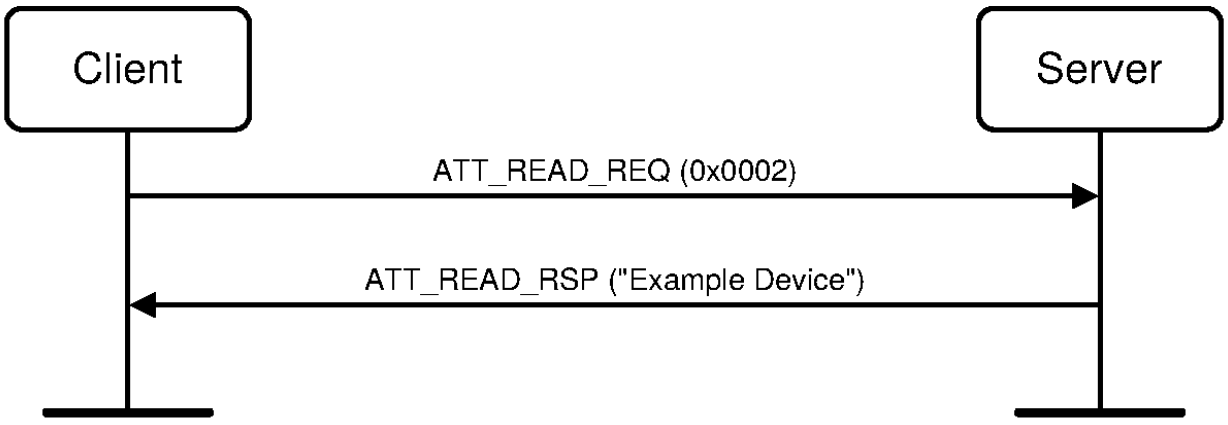


Figure 4.8: Read Characteristic Value example

4.8.2. Read Using Characteristic UUID

This sub-procedure is used to read a *Characteristic Value* from a server when the client only knows the characteristic UUID and does not know the handle of the characteristic.

The ATT_READ_BY_TYPE_REQ PDU is used to perform the sub-procedure. The Attribute Type is set to the known characteristic UUID and the Starting Handle and Ending Handle parameters shall be set to the range over which this read is to be performed. This is typically the handle range for the service in which the characteristic belongs.

Two possible responses can be sent from the server for the ATT_READ_BY_TYPE_REQ PDU: ATT_READ_BY_TYPE_RSP and ATT_ERROR_RSP PDUs.

An ATT_ERROR_RSP PDU is returned if an error occurred on the server.

The ATT_READ_BY_TYPE_RSP PDU returns a list of *Attribute Handle* and *Attribute Value* pairs corresponding to the first characteristics contained in the handle range that will fit into the ATT_READ_BY_TYPE_RSP PDU. This procedure does not return the complete list of all characteristics with the given characteristic UUID within the range of values. If such an operation is required, then the *Discover All Characteristics by UUID* sub procedure shall be used.

If the ATT_ERROR_RSP PDU is sent by the server with the Error Code parameter set to *Attribute Not Found* (0x0A), the characteristic does not exist on the server within the handle range provided.

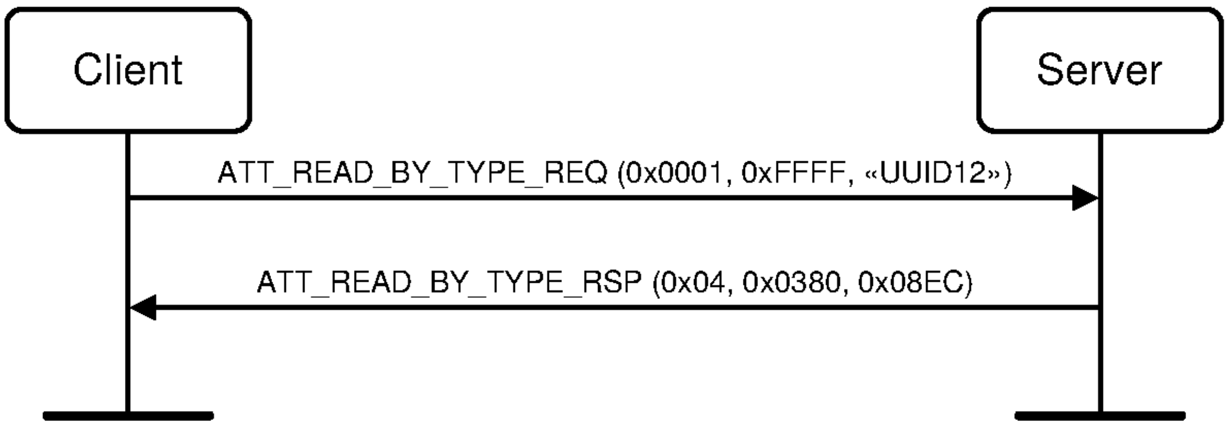

Client
Server
ATT_READ_BY_TYPE_REQ (0x0001, 0xFFFF, «UUID12»)
ATT_READ_BY_TYPE_RSP (0x04, 0x0380, 0x08EC)

Figure 4.9: Read Using Characteristic UUID example

4.8.3. Read Long Characteristic Values

This sub-procedure is used to read a *Characteristic Value* from a server when the client knows the *Characteristic Value Handle* and the length of the *Characteristic Value* is longer than can be sent in a single ATT_READ_RSP PDU.

The ATT_READ_REQ and ATT_READ_BLOB_REQ PDUs are used to perform this sub-procedure. The *Attribute Handle* shall be set to the *Characteristic Value Handle* of the *Characteristic Value* to be read. To read the complete *Characteristic Value* an ATT_READ_REQ PDU should be used for the first part of the value and ATT_READ_BLOB_REQ PDUs shall used for the rest. The Value Offset parameter of each ATT_READ_BLOB_REQ PDU shall be set to the offset of the next octet within the *Characteristic Value* that has yet to be read. The ATT_READ_BLOB_REQ PDU is repeated until the ATT_READ_BLOB_RSP PDU's *Part Attribute Value* parameter is shorter than (ATT_MTU – 1).

For each ATT_READ_BLOB_REQ PDU a ATT_READ_BLOB_RSP PDU is received with a portion of the Characteristic Value contained in the *Part Attribute Value* parameter.

An ATT_ERROR_RSP PDU shall be sent by the server in response to the ATT_READ_BLOB_REQ PDU if insufficient authentication, insufficient authorization, a too-short encryption key size is used by the client, or if a read operation is not permitted on the *Characteristic Value*. The Error Code parameter is set as specified in the Attribute Protocol. If the *Characteristic Value* has a fixed length that is not longer than (ATT_MTU – 1), then the server may respond to the first ATT_READ_BLOB_REQ_PDU with an ATT_ERROR_RSP PDU with the Error Code parameter set to *Attribute Not Long* (0x0B).

Note: The ATT_READ_BLOB_REQ PDU with zero offset may be used to read the first part of the value of the Attribute.

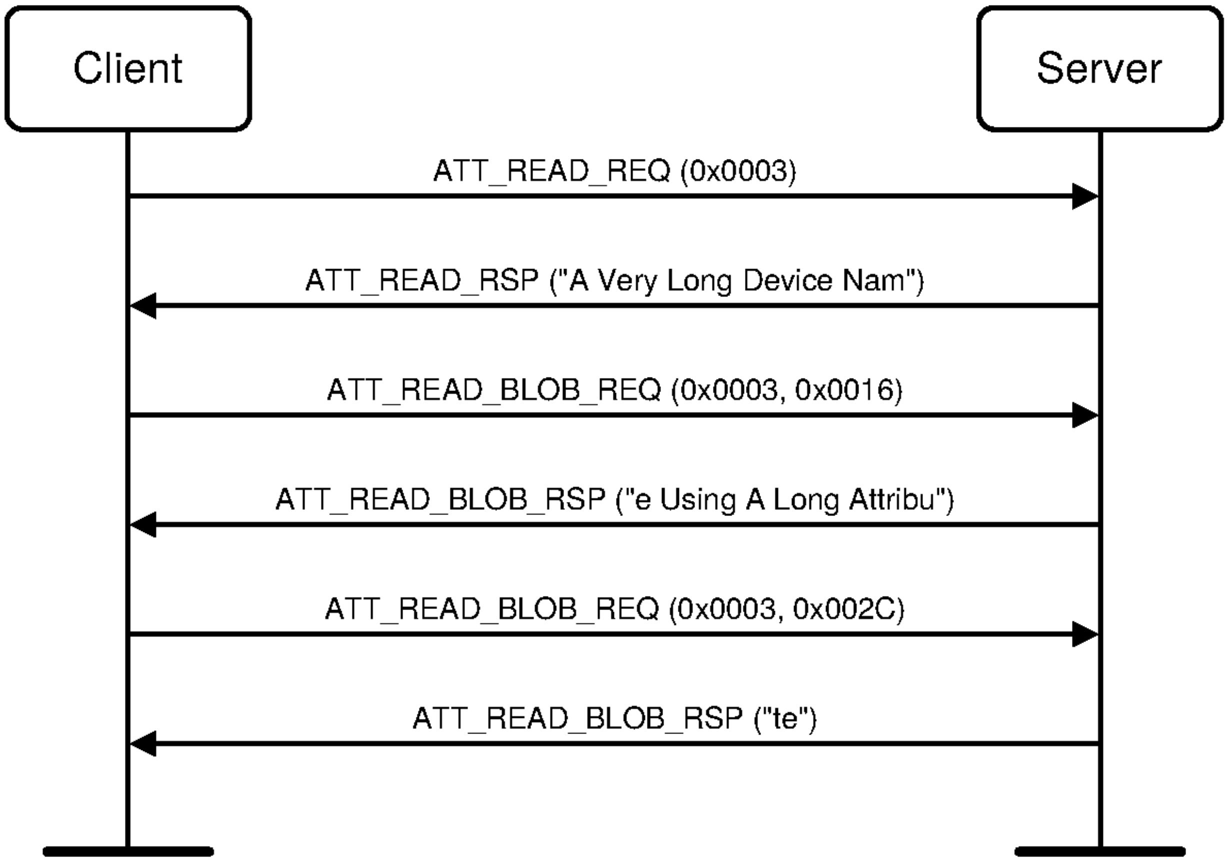

Client
Server
ATT_READ_REQ (0x0003)
ATT_READ_RSP ("A Very Long Device Nam")
ATT_READ_BLOB_REQ (0x0003, 0x0016)
ATT_READ_BLOB_RSP ("e Using A Long Attribu")
ATT_READ_BLOB_REQ (0x0003, 0x002C)
ATT_READ_BLOB_RSP ("te")

Figure 4.10: Read Long Characteristic Values example

4.8.4. Read Multiple Characteristic Values

This sub-procedure is used to read multiple *Characteristic Values* from a server when the client knows the *Characteristic Value Handles*. The ATT_READ_MULTIPLE_REQ PDU is used with the *Set Of Handles* parameter set to the *Characteristic Value Handles*. The ATT_READ_MULTIPLE_RSP PDU returns the *Characteristic Values* in the *Set Of Values* parameter.

The ATT_READ_MULTIPLE_RSP PDU only contains a set of *Characteristic Values* that is less than or equal to (ATT_MTU – 1) octets in length. If the *Set Of Values* is greater than (ATT_MTU – 1) octets in length, only the first (ATT_MTU – 1) octets are included in the response.

Note: A client should not request multiple *Characteristic Values* when the response's *Set Of Values* parameter is equal to (ATT_MTU – 1) octets in length since it is not possible to determine if the last *Characteristic Value* was read or additional *Characteristic Values* exist but were truncated.

An ATT_ERROR_RSP PDU shall be sent by the server in response to the ATT_READ_MULTIPLE_RSP PDU if insufficient authentication, insufficient authorization, a too-short encryption key size, or insufficient encryption is used by the client, or if a read operation is not permitted on any of the *Characteristic Values*. The *Error Code* parameter is set as specified in the Attribute Protocol.

Refer to the Attribute Protocol specification for the format of the *Set Of Handles* and *Set Of Values* parameter.

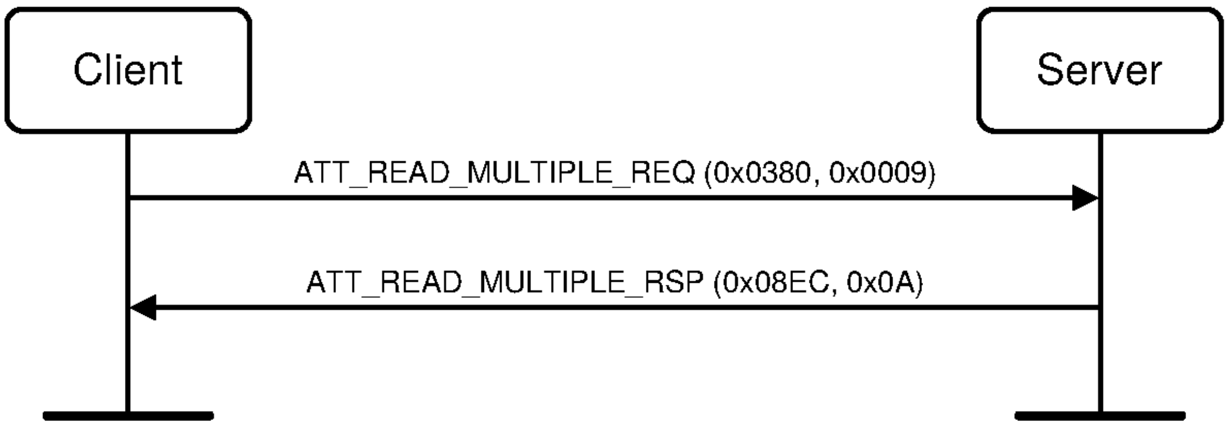

Client
Server
ATT_READ_MULTIPLE_REQ (0x0380, 0x0009)
ATT_READ_MULTIPLE_RSP (0x08EC, 0x0A)

Figure 4.11: Read Multiple Characteristic Values example

4.8.5. Read Multiple Variable Length Characteristic Values

This sub-procedure is used to read multiple variable length Characteristic Values from a server when the client knows the Characteristic Value Handles. The Attribute Protocol ATT_READ_MULTIPLE_VARIABLE_REQ PDU is used with the Set Of Handles parameter set to the Characteristic Value Handles. The ATT_READ_MULTIPLE_VARIABLE_RSP PDU returns the Characteristic Values in the Length Value Tuple List parameter.

The ATT_READ_MULTIPLE_VARIABLE_RSP PDU can only contain a Length Value Tuple List that is less than or equal to (ATT_MTU – 1) octets in length. If the Length Value Tuple List is greater than (ATT_MTU – 1) octets in length, only the first (ATT_MTU – 1) octets are included in the response.

An ATT_ERROR_RSP PDU shall be sent by the server in response to the ATT_READ_MULTIPLE_VARIABLE_REQ PDU if insufficient authentication, insufficient authorization, a too-short encryption key size, or insufficient encryption is used by the client, or if a read operation is not permitted on any of the Characteristic Values. The Error Code parameter is set as specified in the Attribute Protocol.

Refer to the Attribute Protocol specification for the format of the Set Of Handles and Length Value Tuple List parameter.

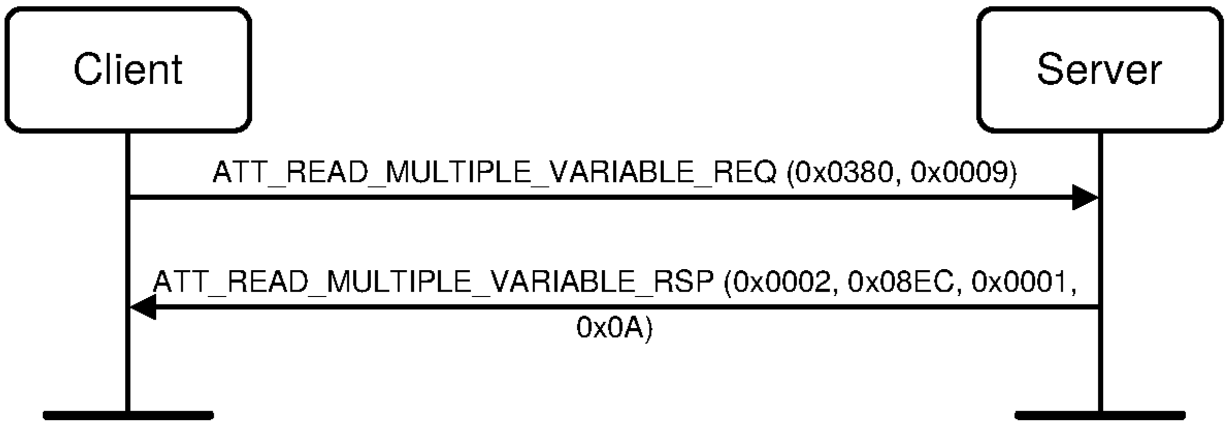

Client
Server
ATT_READ_MULTIPLE_VARIABLE_REQ (0x0380, 0x0009)
ATT_READ_MULTIPLE_VARIABLE_RSP (0x0002, 0x08EC, 0x0001, 0x0A)

Figure 4.12: Read Multiple Variable Length Characteristic Values example

4.9. Characteristic Value Write

This procedure is used to write a *Characteristic Value* to a server.

There are five sub-procedures that can be used to write a *Characteristic Value*: Write Without Response, Signed Write Without Response, Write Characteristic Value, Write Long Characteristic Values and Reliable Writes.

4.9.1. Write Without Response

This sub-procedure is used to write a *Characteristic Value* to a server when the client knows the *Characteristic Value Handle* and the client does not need an acknowledgment that the write was successfully performed. This sub-procedure only writes the first (ATT_MTU – 3) octets of a *Characteristic Value*. This sub-procedure cannot be used to write a long characteristic; instead the *Write Long Characteristic Values* sub-procedure should be used.

The ATT_WRITE_CMD PDU is used for this sub-procedure. The *Attribute Handle* parameter shall be set to the *Characteristic Value Handle*. The *Attribute Value* parameter shall be set to the new *Characteristic Value*.

If the *Characteristic Value* write request is the wrong size, or has an invalid value as defined by the profile, then the write shall not succeed and no error shall be generated by the server.

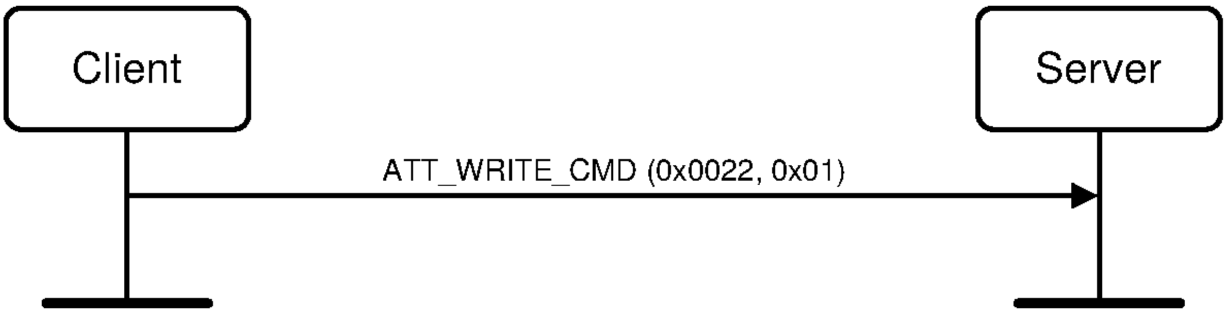


Figure 4.13: Write Without Response example

4.9.2. Signed Write Without Response

This sub-procedure is used to write a *Characteristic Value* to a server when the client knows the *Characteristic Value Handle* and the ATT bearer is not encrypted. This sub-procedure shall only be used if the *Characteristic Properties* authenticated bit is enabled and the client and server device share a bond as defined in [Vol 3] Part C, Generic Access Profile[generic-access-profile.html].

This sub-procedure only writes the first (ATT_MTU – 15) octets of an *Attribute Value.* This sub-procedure cannot be used to write a long Attribute.

The ATT_SIGNED_WRITE_CMD PDU is used for this sub-procedure. The *Attribute Handle* parameter shall be set to the *Characteristic Value Handle*. The *Attribute Value* parameter shall be set to the new *Characteristic Value* authenticated by signing the value, as defined in the Security Manager [Vol 3] Part H, Section 2.4.5[security-manager-specification.html#UUID-193f95ea-7252-1b51-853b-a1999393dddf].

If the authenticated *Characteristic Value* that is written is the wrong size, has an invalid value as defined by the profile, or the signed value does not authenticate the client, then the write shall not succeed and no error shall be generated by the server.

If a connection is already encrypted with LE security mode 1, level 2 or level 3 as defined in [Vol 3] Part C, Section 10.2[generic-access-profile.html#UUID-a8b8b051-dfe4-037b-ec0b-3efdfd6d9d3c] then, a Write Without Response as defined in Section 4.9.1[generic-attribute-profile--gatt-.html#UUID-9f1c2e38-8fbe-f60c-d885-076707c88a43] shall be used instead of a Signed Write Without Response.

On BR/EDR, the ATT bearer is always encrypted, due to the use of Security Mode 4, therefore this sub-procedure shall not be used.

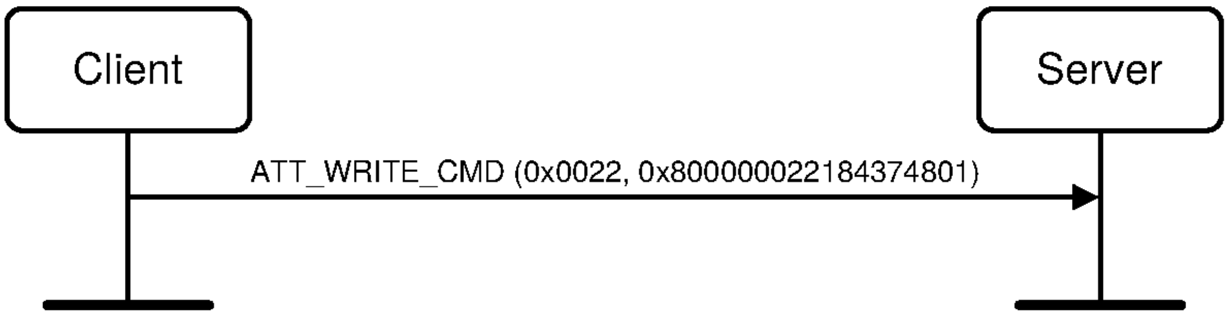


Figure 4.14: Signed Write Without Response example

4.9.3. Write Characteristic Value

This sub-procedure is used to write a *Characteristic Value* to a server when the client knows the *Characteristic Value Handle*. This sub-procedure only writes the first (ATT_MTU – 3) octets of a *Characteristic Value*. This sub-procedure cannot be used to write a long Attribute; instead the *Write Long Characteristic Values* sub-procedure should be used.

The ATT_WRITE_REQ PDU is used to for this sub-procedure. The *Attribute Handle* parameter shall be set to the *Characteristic Value Handle*. The *Attribute Value* parameter shall be set to the new characteristic.

An ATT_WRITE_RSP PDU shall be sent by the server if the write of the *Characteristic Value* succeeded.

An ATT_ERROR_RSP PDU shall be sent by the server in response to the ATT_WRITE_REQ PDU if insufficient authentication, insufficient authorization, a too-short encryption key size is used by the client, or if a write operation is not permitted on the *Characteristic Value*. The Error Code parameter is set as specified in the Attribute Protocol. If the *Characteristic Value* that is written is the wrong size, or has an invalid value as defined by the profile, then the value shall not be written and an ATT_ERROR_RSP PDU shall be sent with the Error Code parameter set to *Application Error* (0x80 – 0x9F) by the server.

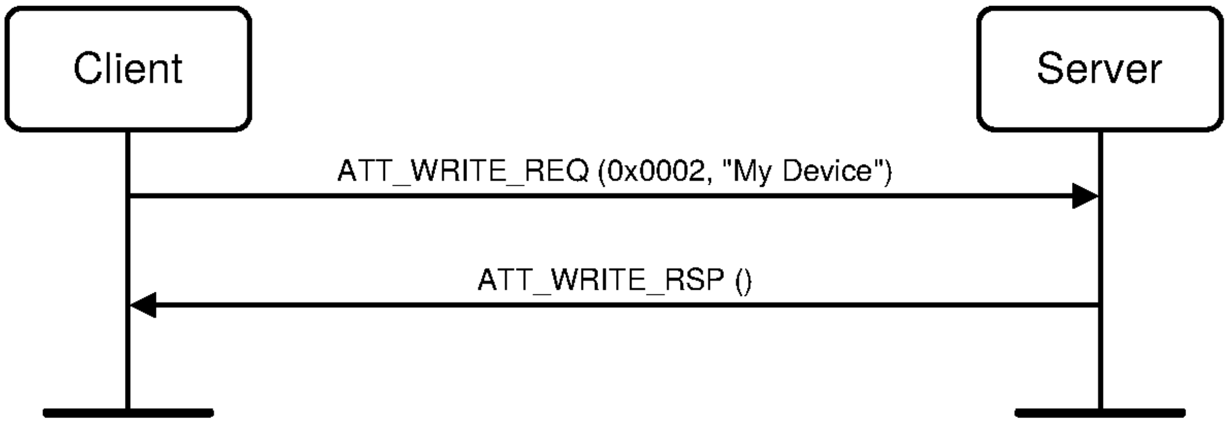


Figure 4.15: Write Characteristic Value example

4.9.4. Write Long Characteristic Values

This sub-procedure is used to write a *Characteristic Value* to a server when the client knows the *Characteristic Value Handle* but the length of the *Characteristic Value* is longer than can be sent in a single ATT_WRITE_REQ PDU.

The ATT_PREPARE_WRITE_REQ and ATT_EXECUTE_WRITE_REQ PDUs are used to perform this sub-procedure. The *Attribute Handle* parameter shall be set to the *Characteristic Value Handle* of the *Characteristic Value* to be written. The *Part Attribute Value* parameter shall be set to the part of the *Attribute Value* that is being written. The *Value Offset* parameter shall be the offset within the *Characteristic Value* to be written. To write the complete *Characteristic Value* the offset should be set to 0x0000 for the first ATT_PREPARE_WRITE_REQ PDU. The offset for subsequent ATT_PREPARE_WRITE_REQ PDUs is the next octet that has yet to be written. The ATT_PREPARE_WRITE_REQ PDU is repeated until the complete *Characteristic Value* has been transferred, after which an ATT_EXECUTE_WRITE_REQ PDU is used to write the complete value.

Note: The values in the ATT_PREPARE_WRITE_RSP PDU do not need to be verified in this sub-procedure.

An ATT_ERROR_RSP PDU shall be sent by the server in response to the ATT_PREPARE_WRITE_REQ PDU if insufficient authentication, insufficient authorization, a too-short encryption key size is used by the client, or if a write operation is not permitted on the *Characteristic Value*. The Error Code parameter is set as specified in the Attribute Protocol. If the *Attribute Value* that is written is the wrong size, or has an invalid value as defined by the profile, then the write shall not succeed and an ATT_ERROR_RSP PDU shall be sent with the Error Code parameter set to *Application Error* (0x80 – 0x9F) by the server.

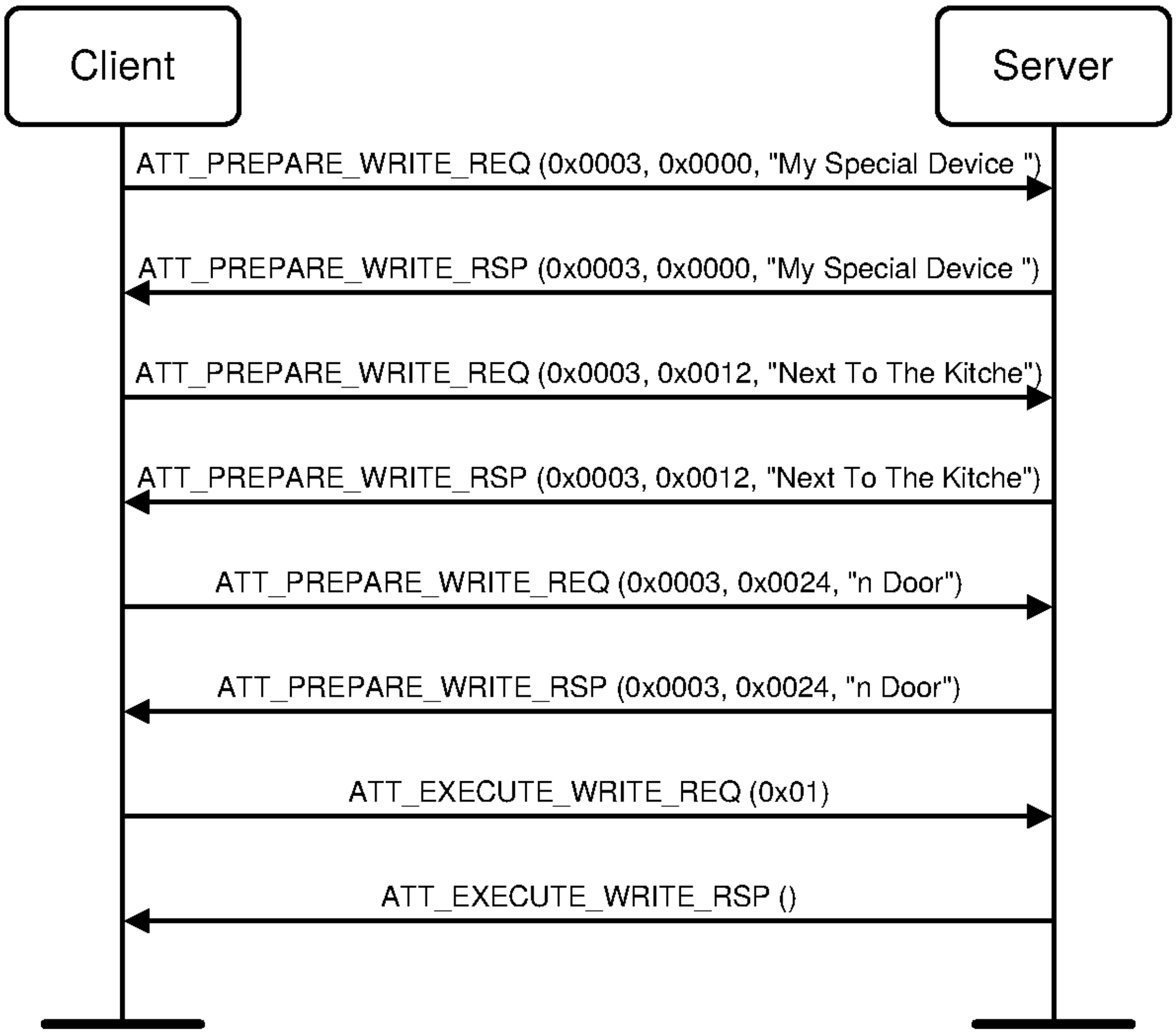

Client
Server
ATT_PREPARE_WRITE_REQ (0x0003, 0x0000, "My Special Device ")
ATT_PREPARE_WRITE_RSP (0x0003, 0x0000, "My Special Device ")
ATT_PREPARE_WRITE_REQ (0x0003, 0x0012, "Next To The Kitche")
ATT_PREPARE_WRITE_RSP (0x0003, 0x0012, "Next To The Kitche")
ATT_PREPARE_WRITE_REQ (0x0003, 0x0024, "n Door")
ATT_PREPARE_WRITE_RSP (0x0003, 0x0024, "n Door")
ATT_EXECUTE_WRITE_REQ (0x01)
ATT_EXECUTE_WRITE_RSP ()

Figure 4.16: Write Long Characteristic Values example

4.9.5. Reliable Writes

This sub-procedure is used to write a *Characteristic Value* to a server when the client knows the *Characteristic Value Handle*, and assurance is required that the correct *Characteristic Value* is going to be written by transferring the *Characteristic Value* to be written in both directions before the write is performed. A higher-layer protocol can also use this sub-procedure to write multiple values in order in a single operation.

The sub-procedure has two phases; the first phase prepares the *Characteristic Values* to be written. To do this, the client transfers the *Characteristic Values* to the server. The server checks the validity of the *Characteristic Values*. The client also checks each *Characteristic Value* to verify it was correctly received by the server using the server responses. Once this is complete, the second phase performs the execution of all of the prepared Characteristic Value writes on the server from this client.

In the first phase, the ATT_PREPARE_WRITE_REQ PDU is used. The *Attribute Handle* shall be set to the *Characteristic Value Handle* that is to be prepared to write. The *Value Offset* and *Part Attribute Value* parameter shall be set to the new *Characteristic Value*.

Two possible responses can be sent from the server for the ATT_PREPARE_WRITE_REQ PDU: ATT_PREPARE_WRITE_-RSP and ATT_ERROR_RSP PDUs.

If the number of prepared write requests exceeds the number of prepared writes supported, then an ATT_ERROR_RSP PDU with the Error Code parameter set to *Prepare Queue Full* (0x09) shall be sent by the server.

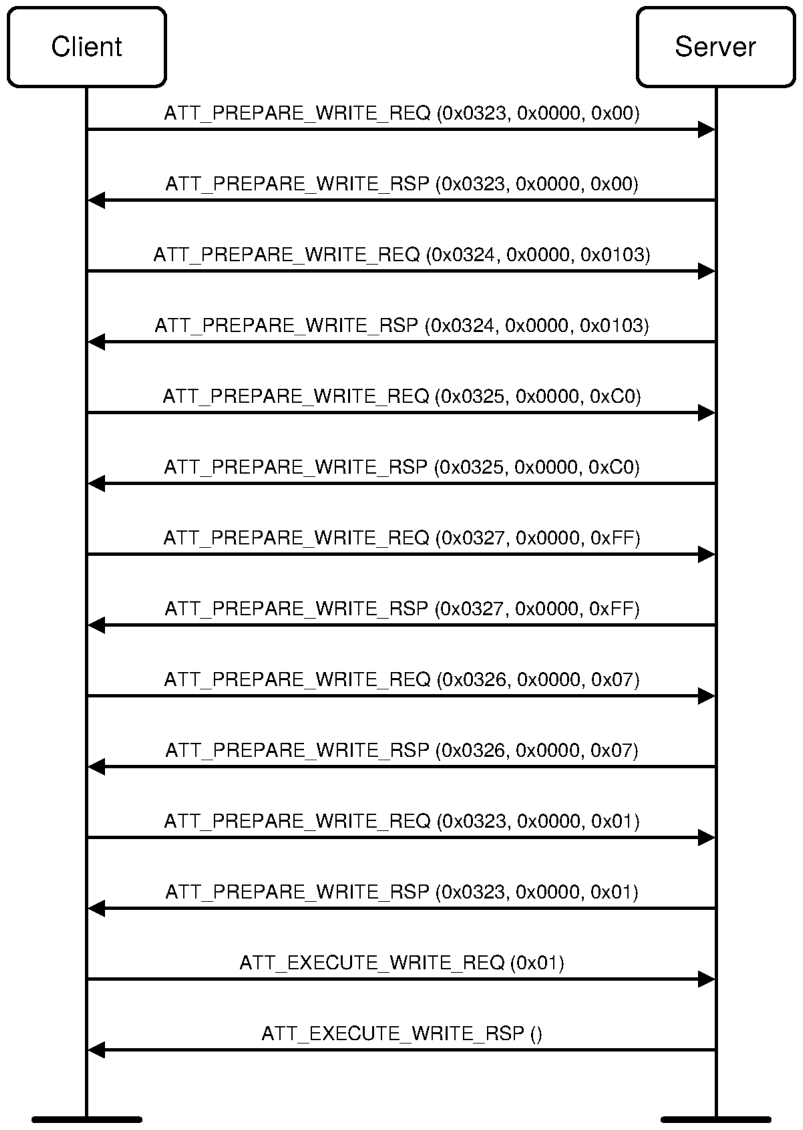
An ATT_ERROR_RSP PDU shall be sent by the server in response to the ATT_PREPARE_WRITE_REQ PDU if insufficient authentication, insufficient authorization, a too-short encryption key size is used by the client, or if a write operation is not permitted on the *Characteristic Value*. The *Error Code* parameter is set as specified in the Attribute Protocol.

If a *Characteristic Value* is prepared two or more times during this sub-procedure, then all prepared values are written to the same *Characteristic Value* in the order that they were prepared.

If an ATT_PREPARE_WRITE_RSP PDU is returned, then the *Value Offset* and *Part Attribute Value* parameter in the response shall be checked with the *Value Offset* and *Part Attribute Value* parameter that was sent in the ATT_PREPARE_WRITE_REQ PDU; if they are different, then the value has been corrupted during transmission, and the sub-procedure shall be aborted by sending an ATT_EXECUTE_WRITE_REQ PDU with the Flags parameter set to 0x00 to cancel all prepared writes. The complete sub-procedure may be restarted.

Multiple ATT_PREPARE_WRITE_REQ PDUs can be sent by a client, each of which will be queued by the server.

In the second phase, the ATT_EXECUTE_WRITE_REQ PDU is used. The Attribute Flags parameter shall be set to 0x01 to immediately write all pending prepared values in the order that they were prepared. The server shall write the prepared writes once it receives this request and shall only send the ATT_EXECUTE_WRITE_RSP PDU once all the prepared values have been successfully written. If the *Characteristic Value* that is written is the wrong size, or has an invalid value as defined by the profile, then the write shall not succeed, and an ATT_ERROR_RSP PDU with the Error Code parameter set to *Application Error* (0x80 – 0x9F) shall be sent by the server. The state of the Characteristic Values that were prepared is undefined.



Client
Server
ATT_PREPARE_WRITE_REQ (0x0323, 0x0000, 0x00)
ATT_PREPARE_WRITE_RSP (0x0323, 0x0000, 0x00)
ATT_PREPARE_WRITE_REQ (0x0324, 0x0000, 0x0103)
ATT_PREPARE_WRITE_RSP (0x0324, 0x0000, 0x0103)
ATT_PREPARE_WRITE_REQ (0x0325, 0x0000, 0xC0)
ATT_PREPARE_WRITE_RSP (0x0325, 0x0000, 0xC0)
ATT_PREPARE_WRITE_REQ (0x0327, 0x0000, 0xFF)
ATT_PREPARE_WRITE_RSP (0x0327, 0x0000, 0xFF)
ATT_PREPARE_WRITE_REQ (0x0326, 0x0000, 0x07)
ATT_PREPARE_WRITE_RSP (0x0326, 0x0000, 0x07)
ATT_PREPARE_WRITE_REQ (0x0323, 0x0000, 0x01)
ATT_PREPARE_WRITE_RSP (0x0323, 0x0000, 0x01)
ATT_EXECUTE_WRITE_REQ (0x01)
ATT_EXECUTE_WRITE_RSP ()

Figure 4.17: Reliable Writes example

4.10. Characteristic Value Notification

This procedure is used to notify a client of the value of a *Characteristic Value* from a server without expecting any Attribute Protocol layer acknowledgment that the notification was successfully received. There are two sub-procedures that can be used to notify a value: Single Notifications and Multiple Variable Length Notifications. Notifications can be configured using the Client Characteristic Configuration descriptor (See Section 3.3.3.3[generic-attribute-profile--gatt-.html#UUID-58fcda17-4f4b-3f53-3ca8-077bbfc77c5d]).

A profile defines when to use Notifications.

4.10.1. Single Notifications

This sub-procedure is used when a server is configured to notify a *Characteristic Value* to a client.

The ATT_HANDLE_VALUE_NTF PDU is used to perform this sub-procedure. The *Attribute Handle* parameter shall be set to the *Characteristic Value Handle* being notified, and the *Attribute Value* parameter shall be set to the *Characteristic Value*.

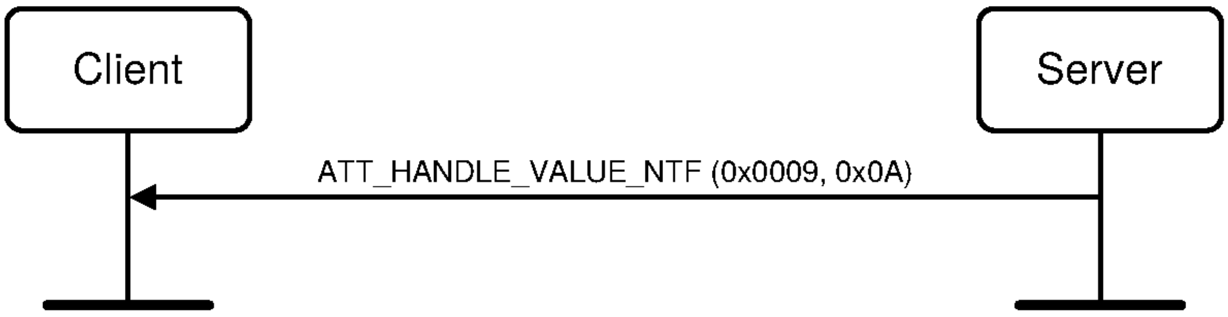


Figure 4.18: Notifications example

4.10.2. Multiple Variable Length Notifications

This sub-procedure is used when a server is configured to notify multiple Characteristic Values to a client.

The Attribute Protocol ATT_MULTIPLE_HANDLE_VALUE_NTF PDU is used to perform this sub-procedure. The Handle Length Value Tuple List parameter shall include the set of Characteristic Value Handles and associated Attribute Values.

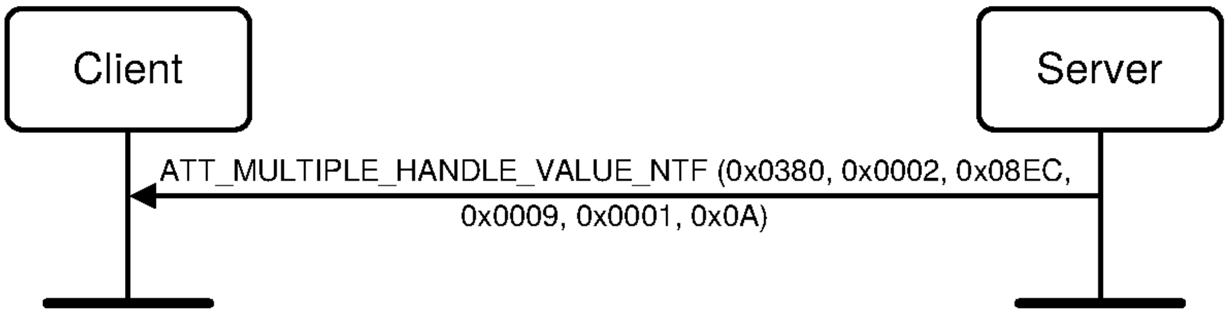

Client
Server
ATT_MULTIPLE_HANDLE_VALUE_NTF (0x0380, 0x0002, 0x08EC, 0x0009, 0x0001, 0x0A)

Figure 4.19: Multiple Variable Length Notifications example

4.11. Characteristic Value Indications

This procedure is used to indicate the *Characteristic Value* from a server to a client. There is one sub-procedure that can be used to indicate a value: Indications. Indications can be configured using the Client Characteristic Configuration descriptor (See Section 3.3.3.3[generic-attribute-profile--gatt-.html#UUID-58fcda17-4f4b-3f53-3ca8-077bbfc77c5d]).

A profile defines when to use Indications.

4.11.1. Indications

This sub-procedure is used when a server is configured to indicate a *Characteristic Value* to a client and expects an Attribute Protocol layer acknowledgment that the indication was successfully received.

The ATT_HANDLE_VALUE_IND PDU is used to perform this sub-procedure. The *Attribute Handle* parameter shall be set to the *Characteristic Value Handle* being indicated, and the *Attribute Value* parameter shall be set to the *Characteristic Value*. Once the ATT_HANDLE_VALUE_IND PDU is received by the client, the client shall respond with an ATT_HANDLE_VALUE_CFM PDU.

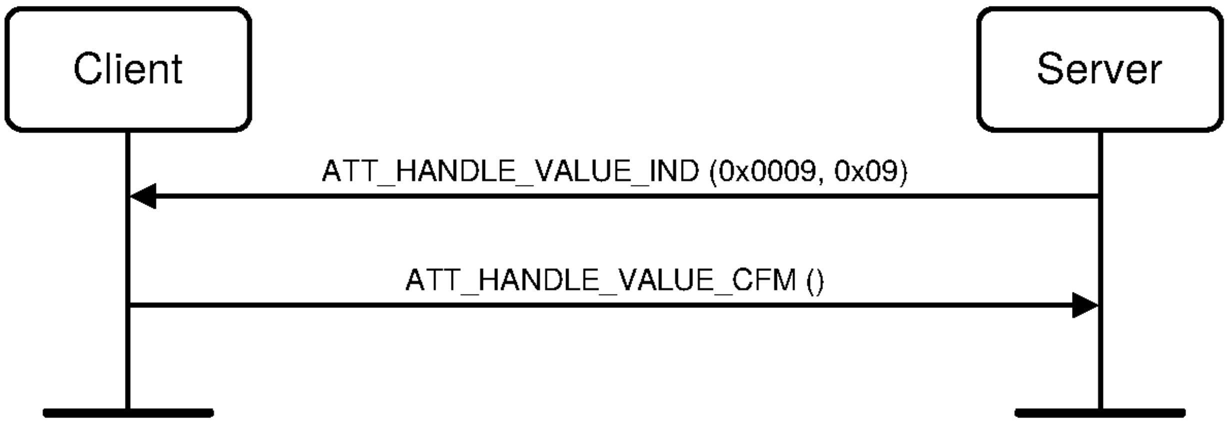


Figure 4.20: Indications example

4.12. Characteristic Descriptors

This procedure is used to read and write characteristic descriptors on a server. There are two sub-procedures that can be used to read and write characteristic descriptors: Read Characteristic Descriptors and Write Characteristic Descriptors.

4.12.1. Read Characteristic Descriptors

This sub-procedure is used to read a characteristic descriptor from a server when the client knows the characteristic descriptor declaration’s Attribute handle.

The ATT_READ_REQ PDU is used for this sub-procedure. The ATT_READ_REQ PDU is used with the *Attribute Handle* parameter set to the characteristic descriptor handle. The ATT_READ_RSP PDU returns the characteristic descriptor value in the *Attribute Value* parameter.

An ATT_ERROR_RSP PDU shall be sent by the server in response to the ATT_READ_REQ PDU if insufficient authentication, insufficient authorization, a too-short encryption key size is used by the client, or if a read operation is not permitted on the *Characteristic Value*. The *Error Code* parameter is set accordingly.

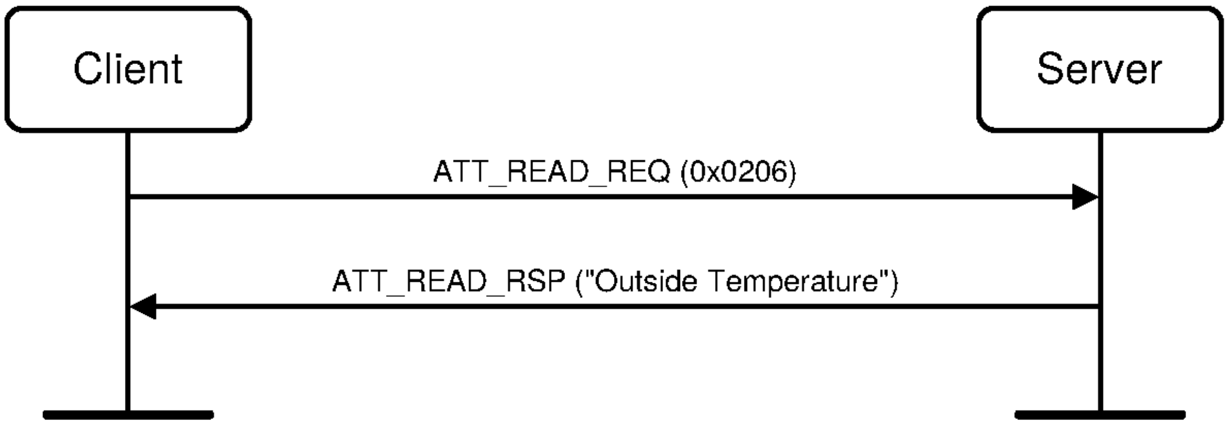


Figure 4.21: Read Characteristic Descriptors example

4.12.2. Read Long Characteristic Descriptors

This sub-procedure is used to read a characteristic descriptor from a server when the client knows the characteristic descriptor declaration’s Attribute handle and the length of the characteristic descriptor declaration is longer than can be sent in a single ATT_READ_RSP PDU.

The ATT_READ_BLOB_REQ PDU is used to perform this sub-procedure. The Attribute Handle parameter shall be set to the characteristic descriptor handle. The Value Offset parameter shall be the offset within the characteristic descriptor to be read. To read the complete characteristic descriptor the offset should be set to 0x00 for the first ATT_READ_BLOB_REQ PDU. The offset for subsequent ATT_READ_BLOB_REQ PDUs is the next octet that has yet to be read. The ATT_READ_BLOB_REQ PDU is repeated until the ATT_READ_BLOB_RSP PDU's Part Attribute Value parameter is zero or an ATT_ERROR_RSP PDU is sent by the server with the Error Code parameter set to *Invalid Offset* (0x07).

For each ATT_READ_BLOB_REQ PDU an ATT_READ_BLOB_RSP PDU is received with a portion of the characteristic descriptor value contained in the Part Attribute Value parameter.

An ATT_ERROR_RSP PDU shall be sent by the server in response to the ATT_READ_BLOB_REQ PDU if insufficient authentication, insufficient authorization, a too-short encryption key size is used by the client, or if a read operation is not permitted on the characteristic descriptor. The Error Code parameter is set accordingly.

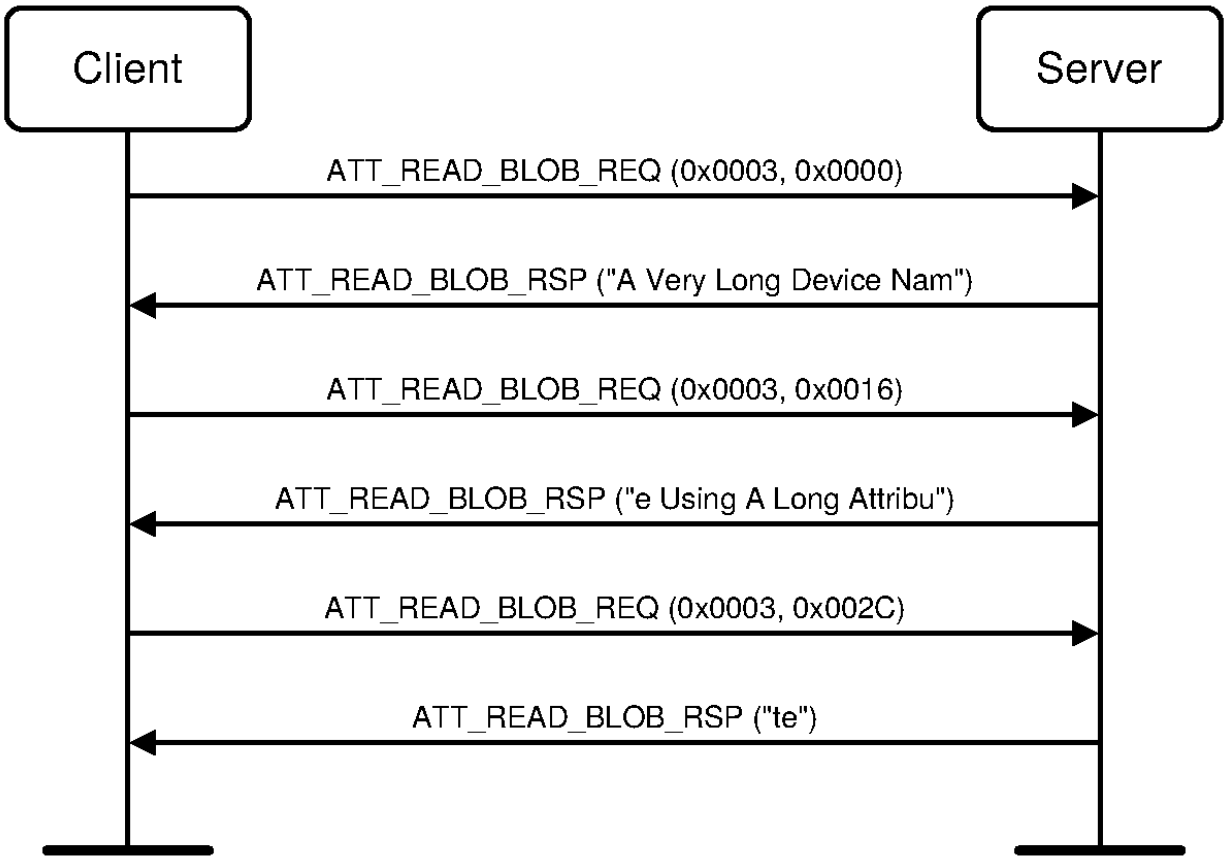

Client
Server
ATT_READ_BLOB_REQ (0x0003, 0x0000)
ATT_READ_BLOB_RSP ("A Very Long Device Nam")
ATT_READ_BLOB_REQ (0x0003, 0x0016)
ATT_READ_BLOB_RSP ("e Using A Long Attribu")
ATT_READ_BLOB_REQ (0x0003, 0x002C)
ATT_READ_BLOB_RSP ("te")

Figure 4.22: Read Long Characteristic Descriptors example

Note: The ATT_READ_BLOB_REQ PDU may be used to read the remainder of an characteristic descriptor value where the first part was read using a simple ATT_READ_REQ PDU.

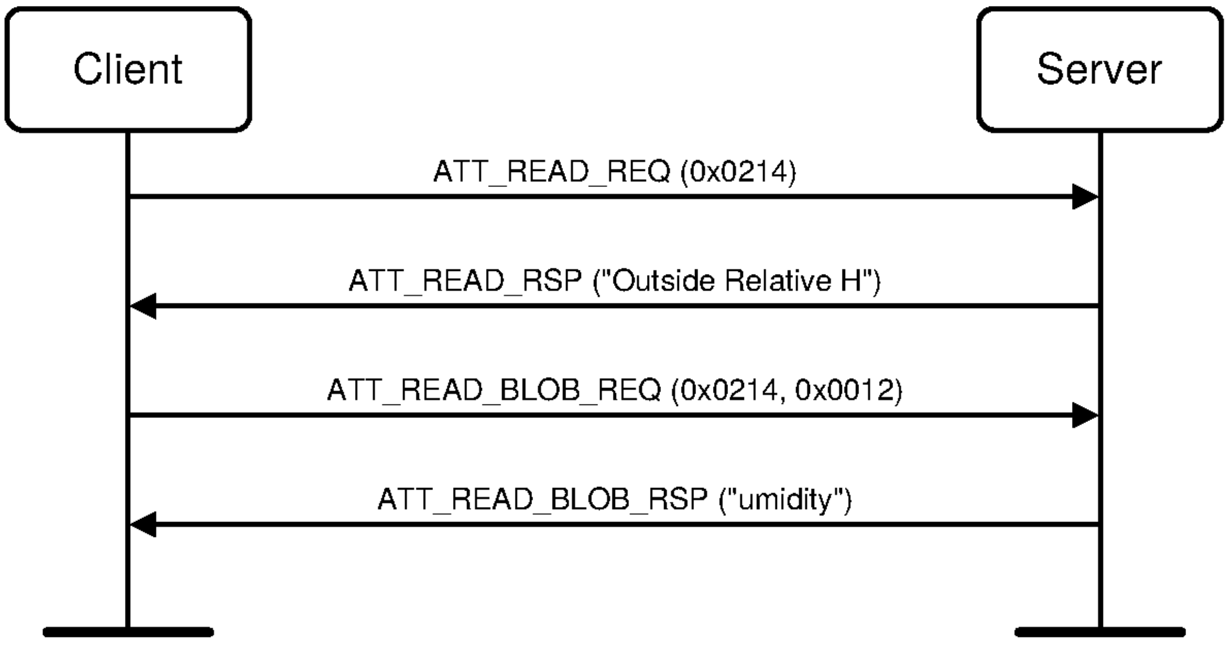

Client
Server
ATT_READ_REQ (0x0214)
ATT_READ_RSP ("Outside Relative H")
ATT_READ_BLOB_REQ (0x0214, 0x0012)
ATT_READ_BLOB_RSP ("umidity")

Figure 4.23: Read Long Characteristic Descriptors (following simple read) example

4.12.3. Write Characteristic Descriptors

This sub-procedure is used to write a characteristic descriptor value to a server when the client knows the characteristic descriptor handle.

The ATT_WRITE_REQ PDU is used for this sub-procedure. The *Attribute Handle* parameter shall be set to the characteristic descriptor handle. The *Attribute Value* parameter shall be set to the new characteristic descriptor value.

An ATT_WRITE_RSP PDU shall be sent by the server if the write of the characteristic descriptor value succeeded.

An ATT_ERROR_RSP PDU shall be sent by the server in response to the ATT_WRITE_REQ PDU if insufficient authentication, insufficient authorization, a too-short encryption key size is used by the client, or if a write operation is not permitted on the *Characteristic Value*. The Error Code parameter shall be set as specified in the Attribute Protocol. If the characteristic descriptor value that is written is the wrong size, or has an invalid value as defined by the profile, or the operation is not permitted at this time then the value shall not be written and an ATT_ERROR_RSP PDU shall be sent with the Error Code parameter set to *Application Error* (0x80 – 0x9F) by the server.

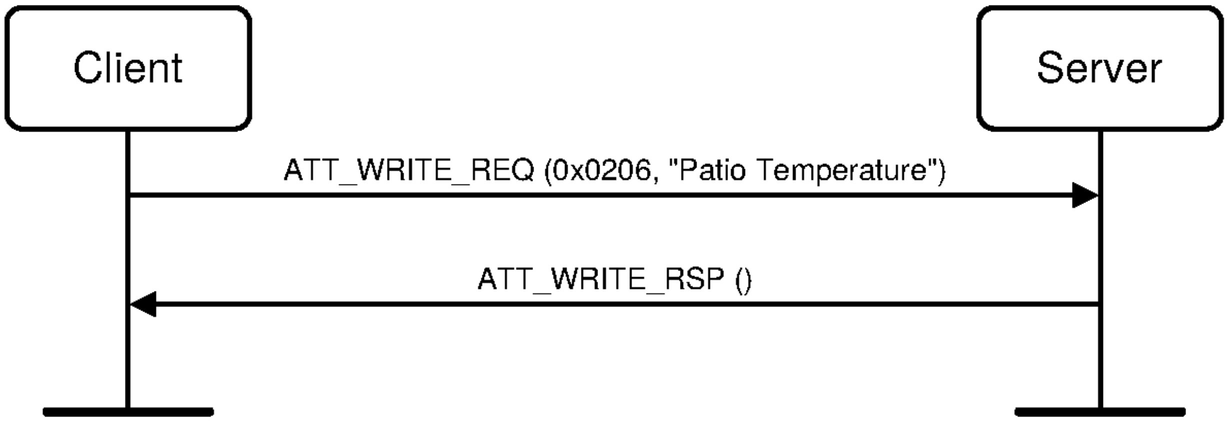


Figure 4.24: Write Characteristic Descriptors example

4.12.4. Write Long Characteristic Descriptors

This sub-procedure is used to write a characteristic descriptor value to a server when the client knows the characteristic descriptor handle but the length of the characteristic descriptor value is longer than can be sent in a single ATT_WRITE_REQ PDU.

The ATT_PREPARE_WRITE_REQ and ATT_EXECUTE_WRITE_REQ PDUs are used to perform this sub-procedure. The *Attribute Handle* parameter shall be set to the *Characteristic Descriptor Handle* of the *Characteristic Value* to be written. The *Part Attribute Value* parameter shall be set to the part of the *Attribute Value* that is being written. The *Value Offset* parameter shall be the offset within the *Characteristic Value* to be written. To write the complete *Characteristic Value* the offset should be set to 0x0000 for the first ATT_PREPARE_WRITE_REQ PDU. The offset for subsequent ATT_PREPARE_WRITE_REQ PDUs is the next octet that has yet to be written. The ATT_PREPARE_WRITE_REQ PDU is repeated until the complete *Characteristic Value* has been transferred, after which an ATT_EXECUTE_WRITE_REQ PDU is used to write the complete value.

Note: The values in the ATT_PREPARE_WRITE_RSP PDU do not need to be verified in this sub-procedure.

An ATT_ERROR_RSP PDU shall be sent by the server in response to the ATT_PREPARE_WRITE_REQ or ATT_EXECUTE_WRITE_REQ PDUs if insufficient authentication, insufficient authorization, a too-short encryption key size is used by the client, or if a write operation is not permitted on the *Characteristic Value*. The Error Code parameter is set as specified in the Attribute Protocol. If the *Attribute Value* that is written is the wrong size, or has an invalid value as defined by the profile, then the write shall not succeed and an ATT_ERROR_RSP PDU shall be sent with the Error Code parameter set to *Application Error* (0x80 – 0x9F) by the server.

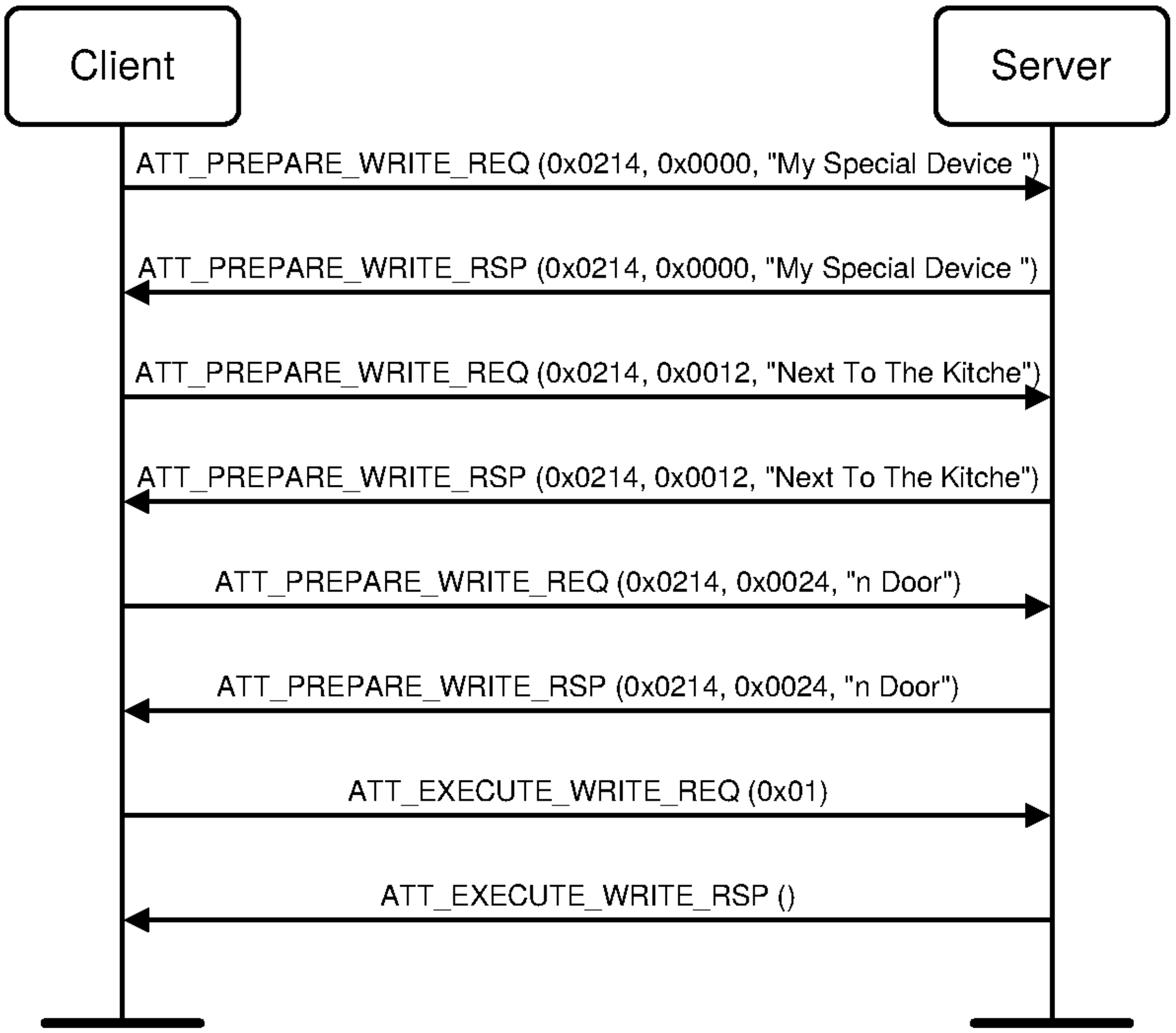

Client
Server
ATT_PREPARE_WRITE_REQ (0x0214, 0x0000, "My Special Device ")
ATT_PREPARE_WRITE_RSP (0x0214, 0x0000, "My Special Device ")
ATT_PREPARE_WRITE_REQ (0x0214, 0x0012, "Next To The Kitche")
ATT_PREPARE_WRITE_RSP (0x0214, 0x0012, "Next To The Kitche")
ATT_PREPARE_WRITE_REQ (0x0214, 0x0024, "n Door")
ATT_PREPARE_WRITE_RSP (0x0214, 0x0024, "n Door")
ATT_EXECUTE_WRITE_REQ (0x01)
ATT_EXECUTE_WRITE_RSP ()

Figure 4.25: Write Long Characteristic Descriptors example

4.13. GATT procedure mapping to ATT protocol opcodes

Table 4.2[generic-attribute-profile--gatt-.html#UUID-b5a253e2-98a4-63df-fc1a-e4555ec83ddc_table-idm13358912755578] describes the mapping of the ATT protocol opcodes to the GATT procedures and sub-procedures. Only those portions of the ATT protocol requests, responses, notifications or indications necessary to implement the mandatory or supported optional sub-procedures is required.

Feature	Sub-Procedure	ATT Protocol Opcodes
Server Configuration	Exchange MTU	ATT_EXCHANGE_MTU_REQ ATT_EXCHANGE_MTU_RSP ATT_ERROR_RSP PDU
Primary Service Discovery	Discover All Primary Services	ATT_READ_BY_GROUP_TYPE_REQ ATT_READ_BY_GROUP_TYPE_RSP ATT_ERROR_RSP PDU
	Discover Primary Services By Service UUID	ATT_FIND_BY_TYPE_VALUE_REQ ATT_FIND_BY_TYPE_VALUE_RSP ATT_ERROR_RSP PDU
Relationship Discovery	Find Included Services	ATT_READ_BY_TYPE_REQ ATT_READ_BY_TYPE_RSP ATT_ERROR_RSP PDU
Characteristic Discovery	Discover All Characteristic of a Service	ATT_READ_BY_TYPE_REQ ATT_READ_BY_TYPE_RSP ATT_ERROR_RSP PDU
	Discover Characteristic by UUID	ATT_READ_BY_TYPE_REQ ATT_READ_BY_TYPE_RSP ATT_ERROR_RSP PDU
Characteristic Descriptor Discovery	Discover All Characteristic Descriptors	ATT_FIND_INFORMATION_REQ ATT_FIND_INFORMATION_RSP ATT_ERROR_RSP PDU
Characteristic Value Read	Read Characteristic Value	ATT_READ_REQ ATT_READ_RSP ATT_ERROR_RSP PDU
	Read Using Characteristic UUID	ATT_READ_BY_TYPE_REQ ATT_READ_BY_TYPE_RSP ATT_ERROR_RSP PDU
	Read Long Characteristic Values	ATT_READ_BLOB_REQ ATT_READ_BLOB_RSP ATT_ERROR_RSP PDU
	Read Multiple Characteristic Values	ATT_READ_MULTIPLE_REQ ATT_READ_MULTIPLE_RSP ATT_ERROR_RSP PDU

Feature	Sub-Procedure	ATT Protocol Opcodes
Characteristic Value Write	Write Without Response	ATT_WRITE_CMD
	Signed Write Without Response	ATT_WRITE_CMD
	Write Characteristic Value	ATT_WRITE_REQ ATT_WRITE_RSP ATT_ERROR_RSP PDU
	Write Long Characteristic Values	ATT_PREPARE_WRITE_REQ ATT_PREPARE_WRITE_RSP ATT_EXECUTE_WRITE_REQ ATT_EXECUTE_WRITE_RSP ATT_ERROR_RSP PDU
	Characteristic Value Reliable Writes	ATT_PREPARE_WRITE_REQ ATT_PREPARE_WRITE_RSP ATT_EXECUTE_WRITE_REQ ATT_EXECUTE_WRITE_RSP ATT_ERROR_RSP PDU
Characteristic Value Notification	Single Notifications	ATT_HANDLE_VALUE_NTF
	Multiple Variable Length Notifications	ATT_MULTIPLE_HANDLE_VALUE_NTF
Characteristic Value Indication	Indications	ATT_HANDLE_VALUE_IND ATT_HANDLE_VALUE_CFM
Characteristic Descriptor Value Read	Read Characteristic Descriptors	ATT_READ_REQ ATT_READ_RSP ATT_ERROR_RSP PDU
	Read Long Characteristic Descriptors	ATT_READ_BLOB_REQ ATT_READ_BLOB_RSP ATT_ERROR_RSP PDU
Characteristic Descriptor Value Write	Write Characteristic Descriptors	ATT_WRITE_REQ ATT_WRITE_RSP ATT_ERROR_RSP PDU
	Write Long Characteristic Descriptors	ATT_PREPARE_WRITE_REQ ATT_PREPARE_WRITE_RSP ATT_PREPARE_WRITE_REQ ATT_PREPARE_WRITE_RSP ATT_ERROR_RSP PDU

Table 4.2: GATT procedure mapping to ATT protocol opcodes

4.14. Procedure timeouts

GATT procedures are protected from failure with an Attribute Protocol transaction timeout.

If the Attribute Protocol transaction times out, the procedure shall be considered to have failed, and the local higher layer shall be notified. No further GATT procedures shall be performed on that ATT bearer. A new GATT procedure shall only be performed on another ATT bearer.

5. L2CAP interoperability requirements

The following default values shall be used by an implementation of this profile. The default values used may be different depending on the physical channel that the Attribute Protocol is being sent over.

5.1. BR/EDR L2CAP interoperability requirements

When using an Unenhanced ATT bearer, L2CAP connection-oriented channels over BR/EDR not in Enhanced Flow Control Mode can be used to transmit Attribute Protocol PDUs. These channels use the channel establishment procedure from L2CAP using the ATT fixed PSM (see [1[generic-attribute-profile--gatt-.html#UUID-c32d82db-0074-99ba-c52a-307a41944215_bibliomixed-idm13391355391050]]) including the configuration procedure to determine the ATT_MTU (see [Vol 3] Part A, Section 5.1[logical-link-control-and-adaptation-protocol-specification.html#UUID-f83ad3d3-8a5d-73cd-153c-67ec4867fda3]). Therefore, the ATT bearer (or the logical link as referred to in the Attribute Protocol) is, in this case, an established L2CAP connection-oriented channel.

5.1.1. ATT_MTU

At the end of the L2CAP configuration phase, upon transition to the OPEN state, the ATT_MTU for this ATT bearer shall be set to the minimum of the negotiated Maximum Transmission Unit configuration options.

Note: The minimum ATT_MTU for BR/EDR is 48 octets, as defined by L2CAP in [Vol 3] Part A, Section 5.1[logical-link-control-and-adaptation-protocol-specification.html#UUID-f83ad3d3-8a5d-73cd-153c-67ec4867fda3].

5.1.2. BR/EDR channel requirements

All Attribute Protocol messages sent by GATT over an L2CAP channel are sent using a dynamic channel ID derived by connecting using a fixed PSM. The use of a fixed PSM allows rapid reconnection of the L2CAP channel for Attribute Protocol as a preliminary SDP query is not required.

All packets sent on this L2CAP channel shall be Attribute PDUs.

PDUs shall be reliably sent.

The flow specification for the Attribute Protocol shall be best effort.

If operating in Basic L2CAP mode, the information payload of the L2CAP B-frame shall be a single Attribute PDU.

The channel shall be encrypted. The Key_Type shall be either an Unauthenticated Combination Key or an Authenticated Combination Key.

The L2CAP connection may be initiated by the client or by the server.

5.1.3. [This section is no longer used]

5.2. LE L2CAP interoperability requirements

When using an Unenhanced ATT bearer, the channel used to carry Attribute Protocol PDUs over LE is the Attribute L2CAP fixed channel.

To terminate the ATT bearer, the physical channel has to be disconnected.

5.2.1. ATT_MTU

Both GATT Client and GATT Server implementations shall support an ATT_MTU not less than the default value.

Default Value	Value for LE
ATT_MTU	23

Table 5.1: LE L2CAP ATT_MTU

5.2.2. LE channel requirements

L2CAP fixed CID 0x0004 shall be used for the Attribute Protocol. All packets sent on this fixed channel shall be Attribute Protocol PDUs.

The flow specification for the Attribute Protocol shall be best effort.

PDUs shall be reliably sent, and not flushed.

The retransmission and flow control mode for this channel shall be Basic L2CAP mode

The default parameters for the payload of the L2CAP B-frame shall be a single Attribute PDU.

Parameter	Value
MTU	23
Flush Timeout	0xFFFF (Infinite)
QoS	Best Effort
Mode	Basic Mode

Table 5.2: Attribute Protocol fixed channel configuration parameters

5.3. Enhanced ATT bearer L2CAP interoperability requirements

When using an Enhanced ATT bearer over BR/EDR, L2CAP connection-oriented channels in Enhanced Retransmission Mode are used to transmit Attribute Protocol PDUs. Such channels are established using L2CAP_CONNECTION_REQ packets.

When using an Enhanced ATT bearer over LE, L2CAP connection-oriented channels in Enhanced Credit Based Flow Control Mode are used to transmit Attribute Protocol PDUs. Such channels are established using L2CAP_CREDIT_BASED_CONNECTION_REQ packets.

In both cases, the EATT fixed PSM [1[generic-attribute-profile--gatt-.html#UUID-c32d82db-0074-99ba-c52a-307a41944215_bibliomixed-idm13391355391050]] is used and the ATT bearer is the established L2CAP connection-oriented channel.

Multiple L2CAP channels can be established between a client and a server.

5.3.1. ATT_MTU

The ATT_MTU for the Enhanced ATT bearer shall be set to the minimum of the MTU field values of the two devices; these values come from the L2CAP_CREDIT_BASED_CONNECTION_REQ and L2CAP_CREDIT_BASED_CONNECTION_RSP signaling packets or the latest L2CAP_CREDIT_-BASED_RECONFIGURE_REQ packets.

Note: The minimum ATT_MTU for an Enhanced ATT bearer is 64 octets.

5.3.2. Channel Requirements

All Attribute Protocol messages sent by GATT over an L2CAP Enhanced Credit Based Flow Control Mode channel are sent using one of the dynamic channel IDs derived by connecting using a fixed PSM.

All packets sent on this L2CAP channel shall be Attribute PDUs.

The flow specification for the Attribute Protocol shall be best effort.

The information payload of the L2CAP K-frame shall be a single Attribute PDU.

The channel shall be encrypted.

5.4. L2CAP collision mitigation

If both devices request L2CAP connections simultaneously and both devices have limited resources, a device may reject the incoming request and find its own request is also rejected. In this situation, the Central may retry immediately but the Peripheral shall wait a minimum of 100 ms before retrying; on LE connections, the Peripheral shall wait at least 2 × (*connPeripheralLatency* + 1) × *connInterval* if that is longer.

5.5. Bearer support

A GATT implementation supporting bearers over BR/EDR shall support at least one of Unenhanced and Enhanced ATT bearers over BR/EDR and may support both.

A GATT implementation supporting bearers over LE shall support Unenhanced ATT bearers over LE and may support Enhanced ATT bearers over LE.

Note: A GATT implementation supporting bearers over both BR/EDR and LE therefore may support any combination of bearers provided that it supports Unenhanced ATT bearers over LE and at least one type of ATT bearer over BR/EDR.

6. GAP interoperability requirements

6.1. BR/EDR GAP interoperability requirements

6.1.1. Connection Establishment

To establish an Unenhanced ATT bearer, the Channel Establishment procedure (as defined in [Vol 3] Part C, Section 7.2[generic-access-profile.html#UUID-66e26fe4-1dca-77b1-e1f5-96965a52bf53]) shall be used with the PSM set to ATT.

Either device may establish an ATT bearer at any time.

To establish an Enhanced ATT bearer, the Section 5.3[generic-attribute-profile--gatt-.html#UUID-2f352749-51c3-45d3-fb4e-63d05aab9263] procedure shall be used with the fixed PSM set to EATT. A device shall not attempt to establish an Enhanced ATT bearer with a peer unless it is aware that the peer supports Enhanced ATT bearers, for example by using an out of band method, via a higher layer protocol, or because the device has checked the peer's Server Supported Features characteristic (see Section 7.4[generic-attribute-profile--gatt-.html#UUID-d42e9f3c-fbec-1849-cac4-127fc7de6c4c]) or its SDP record.

Either device may terminate an ATT bearer at any time.

No idle mode procedures or modes are defined by this profile.

6.2. LE GAP interoperability requirements

6.2.1. Connection Establishment

To establish an Unenhanced ATT bearer, the Connection Establishment procedure (as defined in [Vol 3] Part C, Section 9.3.5[generic-access-profile.html#UUID-0d008ea0-c2ae-7196-3cce-5f548a281331] to Section 9.3.8[generic-access-profile.html#UUID-c6c4fc0c-7b55-28a9-c370-c796fb94511a]) shall be used.

To establish an Enhanced ATT bearer, the Section 5.3[generic-attribute-profile--gatt-.html#UUID-2f352749-51c3-45d3-fb4e-63d05aab9263] procedure shall be used with the fixed PSM set to EATT.

Either device may terminate an ATT bearer at any time.

No idle mode procedures or modes are defined by this profile.

Note: Unlike BR/EDR, it is not necessary to check the Server Supported Features characteristic before attempting to establish an Enhanced ATT bearer.

6.2.2. Profile roles

This profile can be used in the following profile roles (as defined in [Vol 3] Part C, Section 2.2.2[generic-access-profile.html#UUID-d61411b2-8e14-ec23-af7c-e2cc5b807025]):

- Central
- Peripheral

6.3. Disconnected events

6.3.1. Notifications and indications while disconnected

If a client has configured the server to send a notification or indication to the client, it shall be configured to allow re-establishment of the connection when it is disconnected.

If the client is disconnected, but intends to become a Central in the connection it shall perform a GAP connection establishment procedure. If the client is disconnected, but intends to become a Peripheral in the connection it shall go into a GAP connectable mode.

A server shall re-establish a connection with a client when an event or trigger operation causes a notification or indication to a client.

If the server is disconnected, but intends to become a Peripheral in the connection it shall go into a GAP connectable mode. If the server is disconnected, but intends to become a Central in the connection it shall perform a GAP connection establishment procedure.

If the server cannot re-establish a connection, then the notification or indication for this event shall be discarded and no further connection re-establishment shall occur, until another event occurs.

7. Defined Generic Attribute Profile service

All characteristics defined within this section shall be contained in a primary service with the service UUID set to «GATT Service» as defined in Section 3.1[generic-attribute-profile--gatt-.html#UUID-2148a3e6-91f7-e758-750f-8b14377cab6e]. Only one instance of the GATT service shall be exposed on a GATT Server.

Table 7.1[generic-attribute-profile--gatt-.html#UUID-25a99b97-5f9a-6d55-c6c0-381a55b667a9_table-idm13358912909944] lists characteristics that may be present in the server and the characteristics that may be supported by the client.

Characteristic	Ref.	Support in Client	Support in Server
Service Changed	7.1[generic-attribute-profile--gatt-.html#UUID-6ee92321-3db4-dad2-554e-946a80ff7435]	M	C.1
Client Supported Features	7.2[generic-attribute-profile--gatt-.html#UUID-8481d6cb-91d1-ee20-e40b-9e2bcc6e60f5]	O	C.2
Database Hash	7.3[generic-attribute-profile--gatt-.html#UUID-cc9d0db2-8c48-0611-ff58-4b9323fc4d12]	O	O

Characteristic	Ref.	Support in Client	Support in Server
Server Supported Features	7.4[generic-attribute-profile--gatt-.html#UUID-d42e9f3c-fbec-1849-cac4-127fc7de6c4c]	O	C.3

Table 7.1: GATT Profile service characteristic support

C.1: Mandatory if service definitions on the server can be added, changed, or removed; otherwise optional

C.2: Mandatory if the *Database Hash* and *Service Changed* characteristics are supported or if Enhanced ATT Bearer or Multiple Variable Length Notification are supported; otherwise excluded

C.3: Mandatory if any of the features in Table 7.11[generic-attribute-profile--gatt-.html#UUID-d42e9f3c-fbec-1849-cac4-127fc7de6c4c_N1680884813042] are supported, otherwise optional

The assigned UUIDs for these characteristics are defined in Assigned Numbers[https://www.bluetooth.com/specifications/assigned-numbers] [1[generic-attribute-profile--gatt-.html#UUID-c32d82db-0074-99ba-c52a-307a41944215_bibliomixed-idm13391355391050]].

7.1. Service Changed

The «Service Changed» characteristic is a control-point attribute (as defined in [Vol 3] Part F, Section 3.2.6[attribute-protocol--att-.html#UUID-320db6f0-7699-6e2f-4acd-e1bf2718e009]) that shall be used to indicate to connected devices that services have changed (i.e., added, removed or modified). The characteristic shall be used to indicate to clients that have a trusted relationship (i.e. bond) with the server when GATT based services have changed when they re-connect to the server. See Section 2.5.2[generic-attribute-profile--gatt-.html#UUID-542d3f08-c238-c464-d662-b3982828e5bc].

This *Characteristic Value* shall be configured to be indicated using the Client Characteristic Configuration descriptor by a client. Indications caused by changes to the Service Changed Characteristic Value shall be considered lost if the client has erroneously not enabled indications in the Client Characteristic Configuration descriptor (see [Vol 3] Part F, Section 3.3.3[attribute-protocol--att-.html#UUID-55e35307-dcf7-8b68-15b4-fbfcd86799a3]).

Attribute Handle	Attribute Type	Attribute Value			Attribute Permission
0xNNNN	0x2803 – UUID for «Characteristic»	Characteristic Properties = 0x20	0xMMMM = Handle of Characteristic Value	0x2A05 – UUID for «Service Changed»	No Authentication, No Authorization

Table 7.2: Service Changed Characteristic declaration

The *Service Changed Characteristic Value* is two 16-bit *Attribute Handles* concatenated together indicating the beginning and ending *Attribute Handles* affected by an addition, removal, or modification to a GATT-based service on the server. A change to a characteristic value is not considered a modification of the service. If a change has been made to any of the GATT service definition characteristic values other than the *Service Changed* characteristic value and the *Client Supported Features* characteristic value, the range shall also include the beginning and ending *Attribute Handle* for the GATT service definition.

Attribute Handle	Attribute Type	Attribute Value		Attribute Permission
0xMMMM	0x2A05 – UUID for «Service Changed»	0xSSSS – Start of Affected Attribute Handle Range	0xTTTT – End of Affected Attribute Handle Range	No Authentication, No Authorization, Not Readable, Not Writable

Table 7.3: Service Changed Characteristic Value declaration

There shall be only one instance of the *Service Changed* characteristic within the GATT service definition. A *Service Changed* characteristic value shall exist for each client with a trusted relationship.

If the list of GATT based services and the service definitions cannot change for the lifetime of the device then this characteristic shall not exist, otherwise this characteristic shall exist.

If the *Service Changed* characteristic exists on the server, the *Characteristic Value Indication* support on the server is mandatory.

The client shall support *Characteristic Value Indication* of the *Service Changed* characteristic.

The *Service Changed* characteristic *Attribute Handle* on the server shall not change if the server has a trusted relationship with any client.

7.2. Client Supported Features

The *Client Supported Features* characteristic is used by the client to inform the server which features are supported by the client. If the characteristic exists on the server, the client may update the *Client Supported Features* bit field. If a client feature bit is set by a client and the server supports that feature, the server shall fulfill all requirements associated with this feature when communicating with this client. If a client feature bit is not set by a client, then the server shall not use any of the features associated with that bit when communicating with this client.

Attribute Handle	**Attribute Type**	**Attribute Value**			**Attribute Permission**
0xNNNN	0x2803 – UUID for «Characteristic»	Characteristic Properties = 0x0A	0xMMMM = Handle of Characteristic Value	0x2B29 – UUID for «Client Supported Features»	Read only, No Authentication, No Authorization

Table 7.4: Client Supported Features characteristic declaration

The format of the *Client Supported Features* characteristic is defined in Table 7.5[generic-attribute-profile--gatt-.html#UUID-8481d6cb-91d1-ee20-e40b-9e2bcc6e60f5_table-idm13359011484318].

Attribute Handle	**Attribute Type**	**Attribute Value**	**Attribute Permission**
0xMMMM	0x2B29 - UUID for «Client Supported Features»	0xXX...XX (variable length) - Client Features	Readable, Writable, No Authentication, No Authorization

Table 7.5: Client Supported Features value declaration

The *Client Supported Features* characteristic value is an array of octets, each of which is a bit field. The allocation of these bits is specified in Table 7.6[generic-attribute-profile--gatt-.html#UUID-8481d6cb-91d1-ee20-e40b-9e2bcc6e60f5_table-idm13359011479034]. All bits not listed are reserved for future use. The array should not have any trailing zero octets.

If any octet number in Table 7.6[generic-attribute-profile--gatt-.html#UUID-8481d6cb-91d1-ee20-e40b-9e2bcc6e60f5_table-idm13359011479034] does not appear in the attribute value because it is too short, the server shall behave as if that octet were present with a value of zero.

Client Features	Octet	Bit	Ref.	Description
Robust Caching	0	0	2.5.2.1[generic-attribute-profile--gatt-.html#UUID-787b0d4d-3112-9b07-6bef-89dda173a489]	The client supports robust caching
Enhanced ATT bearer	0	1	5.3[generic-attribute-profile--gatt-.html#UUID-2f352749-51c3-45d3-fb4e-63d05aab9263]	The client supports Enhanced ATT bearer
Multiple Handle Value Notifications	0	2	4.10.2[generic-attribute-profile--gatt-.html#UUID-ef60fab0-0495-034a-7f7d-574b6721bb69]	The client supports receiving ATT_MULTIPLE_HANDLE_VALUE_NTF PDUs

Table 7.6: Client Supported Features bit assignments

The default value for the *Client Supported Features* characteristic value shall be all bits set to zero.

There shall be only one instance of the *Client Supported Features* characteristic within the GATT service definition.

A *Client Supported Features* characteristic value shall exist for each connected client. For clients with a trusted relationship, the characteristic value shall be persistent across connections. For clients without a trusted relationship the characteristic value shall be set to the default value at each connection.

The Attribute Handle of the *Client Supported Features* characteristic on the server shall not change during a connection or if the server has a trusted relationship with any client.

A client shall not clear any bits it has set. The server shall respond to any such request with the Error Code parameter set to *Value Not Allowed* (0x13).

7.3. Database Hash

The *Database Hash* characteristic contains the result of a hash function applied to the service definitions in the GATT database. The client may read the characteristic at any time to determine if services have been added, removed, or modified. If any of the input fields to the hash function (as listed in Section 7.3.1[generic-attribute-profile--gatt-.html#UUID-d010b640-e044-bf72-03ef-617e329d3992]) have changed, the server shall calculate a new Database Hash and update the characteristic value.

The *Database Hash* characteristic is a read-only attribute.

Attribute Handle	Attribute Type	Attribute Value			Attribute Permission
0xNNNN	0x2803 – UUID for «Characteristic»	Characteristic Properties = 0x02	0xMMMM = Handle of Characteristic Value	0x2B2A – UUID for «Database Hash»	Read only, No Authentication, No Authorization

Table 7.7: Database Hash characteristic declaration

The characteristic value is a 128-bit unsigned integer number. The calculation of the Database Hash is specified in Section 7.3.1[generic-attribute-profile--gatt-.html#UUID-d010b640-e044-bf72-03ef-617e329d3992].

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xMMMM	0x2B2A - UUID for «Database Hash»	uint128 - Database Hash	Read only, No Authentication, No Authorization

Table 7.8: Database Hash characteristic value declaration

There is only one instance of the *Database Hash* characteristic within the GATT service definition. The same *Database Hash* value is used for all clients, whether a trusted relationship exists or not.

In order to read the value of this characteristic the client shall always use the *GATT Read Using Characteristic UUID* sub-procedure. The *Starting Handle* should be set to 0x0001 and the *Ending Handle* should be set to 0xFFFF.

If a client reads the value of this characteristic while the server is re-calculating the hash following a change to the database, the server shall return the new hash, delaying its response until it is available.

7.3.1. Database Hash calculation

The Database Hash shall be calculated according to RFC-4493 [2[generic-attribute-profile--gatt-.html#UUID-c32d82db-0074-99ba-c52a-307a41944215_bibliomixed-idm13391355404606]]. This RFC defines the Cipher-based Message Authentication Code (CMAC) using AES-128 as the block cipher function, also known as AES-CMAC.

The inputs to AES-CMAC are:

> *m* is the variable length data to be hashed
> *k* is the 128-bit key, which shall be all zero
> (0x00000000_00000000_00000000_00000000)

The 128-bit Database Hash is generated as follows:

Database Hash = AES-CMAC$_k$(m), where *m* is calculated as follows:

> In ascending order of attribute handles, starting with the first handle, concatenate the fields *Attribute Handle*, *Attribute Type*, and *Attribute Value* if the attribute has one of the following types: «Primary Service», «Secondary Service», «Included Service», «Characteristic», or «Characteristic Extended Properties», concatenate the fields *Attribute Handle* and *Attribute Type* if the attribute has one of the following types: «Characteristic User Description», «Client Characteristic Configuration», «Server Characteristic Configuration», «Characteristic Presentation Format», or «Characteristic Aggregate Format», and ignore the attribute if it has any other type (such attributes are not part of the concatenation).
>
> For each *Attribute Handle*, the fields shall be concatenated in the order given above. The byte order used for each field or sub-field value shall be little-endian. If a field contains sub-fields, the subfields shall be concatenated in the order they appear in Section 3[generic-attribute-profile--gatt-.html#UUID-84c0a288-84cc-29aa-4074-6c3a0713b2d5] (Service Interoperability Requirements). For instance, a characteristic declaration value of {0x02, 0x0005, 0x2A00} is represented after concatenation as 02 05 00 00 2A.

The formats of the fields *Attribute Handle*, *Attribute Type*, and *Attribute Value* for the Attribute Types listed above are defined in Section 3[generic-attribute-profile--gatt-.html#UUID-84c0a288-84cc-29aa-4074-6c3a0713b2d5] (Service Interoperability Requirements).

If the length of *m* is not a multiple of the AES-CMAC block length of 128 bits, padding shall be applied as specified in RFC-4493 Section 2.4.

7.4. Server Supported Features

The *Server Supported Features* characteristic is a read-only characteristic that shall be used to indicate support for server features. The server shall set a bit only if the corresponding feature is supported.

Attribute Handle	Attribute Type	Attribute Value			Attribute Permission
0xNNNN	0x2803 – UUID for «Characteristic»	Characteristic Properties = 0x02	0xMMMM = Handle of Characteristic Value	0x2B3A – UUID for «Server Supported Features»	Read Only, No Authentication, No Authorization

Table 7.9: Server Supported Features characteristic declaration

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xMMMM	0x2B3A – UUID for «Server Supported Features»	0xuu - Server Supported Features	Readable

Table 7.10: Server Supported Features value declaration

The *Server Supported Features* characteristic is an array of octets, each of which is a bit field. The allocation of these bits is specified in Table 7.11[generic-attribute-profile--gatt-.html#UUID-d42e9f3c-fbec-1849-cac4-127fc7de6c4c_N1680884813042]. All bits not listed are reserved for future use. The array should not have any trailing zero octets.

Server Supported Features	Octet	Bit	Ref.	Description
EATT Supported	0	0	5.3[generic-attribute-profile--gatt-.html#UUID-2f352749-51c3-45d3-fb4e-63d05aab9263]	Enhanced ATT bearer supported

Table 7.11: Server Supported Features bit assignments

If any octet number in Table 7.11[generic-attribute-profile--gatt-.html#UUID-d42e9f3c-fbec-1849-cac4-127fc7de6c4c_N1680884813042] does not appear in the attribute value because it is too short, the client shall behave as if that octet were present with the value of zero.

There shall be only one instance of the *Server Supported Features* characteristic within the GATT service definition.

8. Security considerations

8.1. Authentication requirements

Authentication in the GATT Profile is applied to each characteristic independently. Authentication requirements are specified in this profile, related higher layer specifications or are implementation specific if not specified otherwise.

The GATT Profile procedures are used to access information that may require the client to be authenticated and have an encrypted connection before a characteristic can be read or written.

If such a request is issued when the physical link is unauthenticated or unencrypted, the server shall send an ATT_ERROR_RSP PDU. The client wanting to read or write this characteristic can then request that the physical link be authenticated using the GAP authentication procedure, and once this has been completed, send the request again.

The list of services and characteristics that a device supports is not considered private or confidential information, and therefore the Service and Characteristic Discovery procedures shall always be permitted. This implies that an Error Code parameter set to *Insufficient Authentication* (0x05) shall not be used in an ATT_ERROR_RSP PDU for a *Find Information Request*.

Note: A characteristic may be allowed to be read by any device, but only written by an authenticated device. An implementation should take this into account, and not assume that if it can read a *Characteristic Value*, it will also be able to write the *Characteristic Value*. Similarly, if a characteristic can be written, it does not mean the characteristic can also be read. Each individual characteristic could have different security properties.

Once sufficient authentication of the client has been established to allow access to one characteristic within a service definition, a server may also allow access to other characteristics within the service definition depending on the higher level or implementation specific requirements.

A server may allow access to most characteristics within a service definition once sufficient authentication has been performed, but restrict access to other characteristics within the same service definition. This may result due to some characteristics requiring stronger authentication requirements than currently enabled.

Once a server has authenticated a client for access to characteristics in one service definition, it may automatically allow access to characteristics in other service definitions.

8.2. Authorization requirements

Authorization in the GATT Profile is applied to each characteristic independently. Authorization requirements may be specified in this profile, related higher layer specifications or are implementation specific if not specified otherwise.

The GATT Profile can be used to access information that may require authorization before a characteristic can be read or written.

If such a request is issued to a characteristic contained in a service definition that is not authorized, the responder shall send an ATT_ERROR_RSP PDU with the Error Code parameter set to *Insufficient Authorization* (0x08).

Once a server has authorized a client for access to characteristics in one group or service definition, it may automatically allow access to characteristics in other service definitions.

9. SDP interoperability requirements

A device that supports GATT over BR/EDR and only one of ATT or EATT shall publish the SDP record shown in Table 9.1[generic-attribute-profile--gatt-.html#UUID-748be175-5220-ddd2-0722-d73c541e0533_table-idm13359013046580]; if both ATT and EATT are supported, the device shall publish the SDP record shown in Table 9.2[generic-attribute-profile--gatt-.html#UUID-748be175-5220-ddd2-0722-d73c541e0533_table-idm13359013396072]. The GATT Service Start Handle shall be set to the attribute handle of the «GATT Service» service declaration. The GATT Service End Handle shall be set to the attribute handle of the last attribute within the «GATT Service» service definition group.

Item	Type	Value	Meaning
Attribute ID	uint16	0x0001	ServiceClassIDList
Attribute Value	Data element sequence (1 item)		
Service Class	UUID	«GATT Service»	
Attribute ID	uint16	0x0004	ProtocolDescriptorList
Attribute Value	Data element sequence (2 items)		
Protocol Descriptor	Data element sequence (2 items)		
Protocol	UUID	«L2CAP»	
Parameter 0	uint16	0x001F *or* 0x0027	PSM = ATT *or* PSM = EATT
Protocol Descriptor	Data element sequence (3 items)		
Protocol	UUID	«ATT»	

Item		Type	Value	Meaning
	Parameter 0	uint16	0xHHHH	GATT service start handle
	Parameter 1	uint16	0xHHHH	GATT service end handle
Attribute ID		uint16	0x0005	BrowseGroupList
Attribute Value		Data element sequence (1 item)		
	Group	UUID	«PublicBrowseRoot»	

Table 9.1: SDP record for the Generic Attribute Profile, if only one of ATT or EATT is supported

Item				Type	Value	Meaning
Attribute ID				uint16	0x0001	ServiceClassIDList
Attribute Value				Data element sequence (1 item)		
	Service class			UUID	«GATT Service»	
Attribute ID				uint16	0x0004	ProtocolDescriptorList
Attribute Value				Data element sequence (2 items)		
	Protocol Descriptor			Data element sequence (2 items)		
		Protocol		UUID	«L2CAP»	
		Parameter 0		uint16	0x001F	PSM = ATT
	Protocol Descriptor			Data element sequence (3 items)		
		Protocol		UUID	«ATT»	
		Parameter 0		uint16	0xHHHH	GATT service start handle
		Parameter 1		uint16	0xHHHH	GATT service end handle
Attribute ID				uint16	0x000D	AdditionalProtocolDescriptorList
Attribute Value				Data element sequence (1 item)		
	Protocol Descriptor List			Data element sequence (2 items)		
		Protocol Descriptor		Data element sequence (2 items)		
			Protocol	UUID	«L2CAP»	
			Parameter 0	uint16	0x0027	PSM = EATT
		Protocol Descriptor		Data element sequence (3 items)		
			Protocol	UUID	«ATT»	
			Parameter 0	uint16	0xHHHH	GATT service start handle
			Parameter 1	uint16	0xHHHH	GATT service end handle
Attribute ID				uint16	0x0005	BrowseGroupList
Attribute Value				Data element sequence (1 item)		
	Group			UUID	«PublicBrowseRoot»	

Table 9.2: SDP record for the Generic Attribute Profile, if both ATT and EATT are supported

10. References

[1] Assigned Numbers Specification: https://www.bluetooth.com/specifications/assigned-numbers[https://www.bluetooth.com/specifications/assigned-numbers]

[2] RFC-4493: http://www.ietf.org/rfc/rfc4493.txt[http://www.ietf.org/rfc/rfc4493.txt]

[3] Core Specification Supplement, Part A, Data Types Specification

Appendix A. Example ATT Server contents

Table A.1[generic-attribute-profile--gatt-.html#UUID-077c8eb0-10f5-8c69-63d6-603fee0dc762_table-idm13359013605988] shows an example ATT Server and the attributes contained on the server.

Note: This example does not necessarily use UUIDs or services defined by the Bluetooth SIG or in adopted profiles.

Handle	**Attribute Type**	**Attribute Value**
0x0001	«Primary Service»	«GAP Service»
0x0004	«Characteristic»	{0x02, 0x0006, «Device Name»}
0x0006	«Device Name»	“Example Device”
0x0010	«Primary Service»	«GATT Service»
0x0011	«Characteristic»	{0x26, 0x0012, «Service Changed»}
0x0012	«Service Changed»	0x0000, 0x0000
0x0100	«Primary Service»	«Battery State Service»
0x0106	«Characteristic»	{0x02, 0x0110, «Battery State»}
0x0110	«Battery State»	0x04
0x0200	«Primary Service»	«Thermometer Humidity Service»
0x0201	«Include»	{0x0500, 0x0504, «Manufacturer Service»}
0x0202	«Include»	{0x0550,0x0568}
0x0203	«Characteristic»	{0x02, 0x0204, «Temperature»}
0x0204	«Temperature»	0x028A
0x0205	«Characteristic Presentation Format»	{0x0E, 0xFE, «degrees Celsius», 0x01, «Outside»}
0x0206	«Characteristic User Description»	“Outside Temperature”
0x0210	«Characteristic»	{0x02, 0x0212, «Relative Humidity»}
0x0212	«Relative Humidity»	0x27
0x0213	«Characteristic Presentation Format»	{0x04, 0x00, «Percent», «Bluetooth SIG», «Outside»}
0x0214	«Characteristic User Description»	“Outside Relative Humidity”

Handle	Attribute Type	Attribute Value
0x0280	«Primary Service»	«Weight Service»
0x0281	«Include»	0x0505, 0x0509, «Manufacturer Service»}
0x0282	«Characteristic»	{0x02, 0x0283, «Weight kg»}
0x0283	«Weight kg»	0x00005582
0x0284	«Characteristic Presentation Format»	{0x08, 0xFD, «kilogram», «Bluetooth SIG», «Hanging»}
0x0285	«Characteristic User Description»	“Rucksack Weight”
0x0300	«Primary Service»	«Position Service»
0x0301	«Characteristic»	{0x02, 0x0302, «Latitude Longitude»}
0x0302	«Latitude Longitude»	0x28BEAFA40B320FCE
0x0304	«Characteristic»	{0x02, 0x0305, «Latitude Longitude Elevation»}
0x0305	«Latitude Longitude Elevation»	0x28BEAFA40B320FCE0176
0x0400	«Primary Service»	«Alert Service»
0x0401	«Characteristic»	{0x0E, 0x0402, «Alert Enumeration»}
0x0402	«Alert Enumeration»	0x00
0x0500	«Secondary Service»	«Manufacturer Service»
0x0501	«Characteristic»	{0x02, 0x0502, «Manufacturer Name»}
0x0502	«Manufacturer Name»	“ACME Temperature Sensor”
0x0503	«Characteristic»	{0x02, 0x0504, «Serial Number»}
0x0504	«Serial Number»	“237495-3282-A”
0x0505	«Secondary Service»	«Manufacturer Service»
0x0506	«Characteristic»	{0x02, 0x0507, «Manufacturer Name»}
0x0507	«Manufacturer Name»	“ACME Weighing Scales”
0x0508	«Characteristic»	{0x02, 0x0509, «Serial Number»}
0x0509	«Serial Number»	“11267-2327A00239”
0x0550	«Secondary Service»	«Vendor Specific Service»
0x0560	«Characteristic»	{0x02, 0x0568, «Vendor Specific Type»}
0x0568	«Vendor Specific Type»	0x56656E646F72

Table A.1: Examples of ATT Server contents

As can be seen, the ATT Server indicates support for ten services: GAP Service, GATT Service, Battery State Service, Thermometer Humidity Service, Weight Service, Position Service, Alert Service, two Manufacturer Services, and a Vendor Specific Service.

The server contains the following information about each of the services:

- The characteristic containing the name of the device is “Example Device”.

- The characteristic indicating the server supports all the attribute opcodes, and supports two prepared write values.
- The characteristic containing the battery state with a value of 0x04, meaning it is discharging.
- The characteristic containing the outside temperature with a value of 6.5 °C.
- The characteristic containing the outside relative humidity with a value of 39%.
- The characteristic containing the weight hanging off the device with a value of 21.89 kg.
- The characteristic containing the position of this device with the value of 68.3585444 degrees north, 18.7830222 degrees east, with an elevation of 374 meters.
- The characteristic containing the temperature sensor manufacturer with the value of ACME Temperature Sensor.
- The characteristic containing the serial number for the temperature sensor with a value of 237495-3282-A.
- The characteristic containing the weighing sensor is manufacturer with a value of ACME Weight Scales.
- The characteristic containing the serial number for the weighing sensor with a value of 11267-2327A00239.

The device is therefore on the side of the Abisko Turiststation, Norrbottens Län, Sweden, with a battery in good state, measuring a relatively warm day, with low humidity, and a heavy rucksack.

Appendix B. Example Database Hash

Table B.1[generic-attribute-profile--gatt-.html#UUID-108023e0-a27e-d7ef-85ee-24cc931b63e6_table-idm13359013642564] shows how the variable length data *m* for calculating the Database Hash is constructed from an example GATT database. The column *Included in Hash* indicates whether the *Attribute Handle* (H), *Attribute Type* (T), or *Attribute Value* (V) are included in *m*.

Attribute Handle	**Attribute Type**	**Attribute Value**	**Notes**	**Included in Hash**	**Data Included in m**
0x0001	0x2800	0x1800	Primary Service: GAP Service	HTV	01 00 00 28 00 18
0x0002	0x2803	{0x0A, 0x0003, 0x2A00}	Characteristic (Read, Write): Device Name	HTV	02 00 03 28 0A 03 00 00 2A
0x0003	0x2A00	*any*	Characteristic Value: Device Name	No	*none*
0x0004	0x2803	{0x02, 0x0005, 0x2A01}	Characteristic (Read): Appearance	HTV	04 00 03 28 02 05 00 01 2A
0x0005	0x2A01	*any*	Characteristic Value: Appearance	No	*none*
0x0006	0x2800	0x1801	Primary Service: GATT Service	HTV	06 00 00 28 01 18
0x0007	0x2803	{0x20, 0x0008, 0x2A05}	Characteristic (Indicate): Service Changed	HTV	07 00 03 28 20 08 00 05 2A
0x0008	0x2A05	*any*	Characteristic Value: Service Changed	No	*none*
0x0009	0x2902	0x0002	Client Characteristic Configuration Descriptor	HT	09 00 02 29
0x000A	0x2803	{0x0A, 0x000B, 0x2B29}	Characteristic (Read, Write): Client Supported Features	HTV	0A 00 03 28 0A 0B 00 29 2B
0x000B	0x2B29	*any*	Characteristic Value: Client Supported Features	No	*none*
0x000C	0x2803	{0x02, 0x000D, 0x2B2A}	Characteristic (Read): Database Hash	HTV	0C 00 03 28 02 0D 00 2A 2B

Attribute Handle	Attribute Type	Attribute Value	Notes	Included in Hash	Data Included in m
0x000D	0x2B2A	*any*	Characteristic Value: Database Hash	No	*none*
0x000E	0x2800	0x1808	Primary Service: Glucose Service	HTV	0E 00 00 28 08 18
0x000F	0x2802	{0x0014, 0x0016, 0x180F}	Included Service: Battery Service	HTV	0F 00 02 28 14 00 16 00 0F 18
0x0010	0x2803	{0xA2, 0x0011, 0x2A18}	Characteristic (Read, Indicate, Extended Properties): Glucose Measurement	HTV	10 00 03 28 A2 11 00 18 2A
0x0011	0x2A18	*any*	Characteristic Value: Glucose Measurement	No	*none*
0x0012	0x2902	0x0002	Client Characteristic Configuration Descriptor	HT	12 00 02 29
0x0013	0x2900	0x0000	Extended Properties	HTV	13 00 00 29 00 00
0x0014	0x2801	0x180F	Secondary Service: Battery Service	HTV	14 00 01 28 0F 18
0x0015	0x2803	{0x02, 0x0016, 0x2A19}	Characteristic (Read): Battery Level	HTV	15 00 03 28 02 16 00 19 2A
0x0016	0x2A19	*any*	Characteristic Value: Battery Level	No	*none*

Table B.1: Example database

The resulting variable length data m divided into blocks M0 to M6 is:

```
M0: 01000028 00180200 03280A03 00002A04
M1: 00032802 0500012A 06000028 01180700
M2: 03282008 00052A09 0002290A 0003280A
M3: 0B00292B 0C000328 020D002A 2B0E0000
M4: 2808180F 00022814 0016000F 18100003
M5: 28A21100 182A1200 02291300 00290000
M6: 14000128 0F181500 03280216 00192A
```

The resulting Database Hash is (MSB to LSB):

Database Hash = AES-CMAC$_k$(m) = F1 CA 2D 48 EC F5 8B AC 8A 88 30 BB B9 FB A9 90

Note: The bytes in M0 to M6 and the Database Hash are ordered from the most significant on the left to the least significant on the right.

In this section