HomeKit Accessory Protocol Specification

Release R13

Developer

Contents

List of Figures

List of Tables

1 Introduction

NOTICE OF PROPRIETARY PROPERTY: THIS DOCUMENT AND THE INFORMATION CONTAINED HEREIN IS THE PROPRIETARY PROPERTY OF APPLE INC. THE POSSESSOR AGREES TO THE FOLLOWING: (I) TO MAINTAIN THIS DOCUMENT IN CONFIDENCE, (II) NOT TO REPRODUCE OR COPY IT, (III) NOT TO REVEAL OR PUBLISH IT IN WHOLE OR IN PART, (IV) ALL RIGHTS RESERVED.

ACCESS TO THIS DOCUMENT AND THE INFORMATION CONTAINED HEREIN IS GOVERNED BY THE TERMS OF THE MFi LICENSE AGREEMENT. ALL OTHER USE SHALL BE AT APPLE'S SOLE DISCRETION.

1.1 Requirements, Recommendations, and Permissions

This specification contains statements that are incorporated by reference into legal agreements between Apple and its licensees. The use of the words *must*, *must not*, *required*, *shall*, *shall not*, *should*, *should not*, *recommended*, *not recommended*, *may*, *optional*, and *deprecated* in a statement have the following meanings:

- *must*, *shall*, or *required* means the statement is an absolute requirement.
- *must not*, *shall not* or *prohibited* means the statement is an absolute prohibition.
- *should* or *recommended* means the full implications must be understood before choosing a different course.
- *should not* or *not recommended* means the full implications must be understood before choosing this course.
- *may* or *optional* means the statement is truly optional, and its presence or absence cannot be assumed.
- *deprecated* means the statement is provided for historical purposes only and is equivalent to 'must not'.

The absence of requirements, recommendations, or permissions for a specific accessory design in this specification must not be interpreted as implied approval of that design. Developers are strongly encouraged to ask Apple for feedback on accessory designs that are not explicitly mentioned in this specification.

1.2 Terminology

1.2.1 Accessory, Device, and Product

Throughout this specification:

- The term *device* or *controller* is used to refer to an Apple iPod, iPhone, iPad or Watch (typically running iOS or watchOS, Apple's mobile operating system).
- The term *accessory* is used to refer to any product intended to interface with a device via the means described in this specification.
- The term *product* is used to refer generically to either a Mac (Apple computers that run macOS) or an aforementioned *device*.

1.2.2 Component

An accessory is defined as a collection of functional units called *components*. Examples of a component include, but are not limited to, the following:

- Data transport
- Power source
- Human Interface Device (HID) control set

1.2.3 Feature

All accessories must support one or more accessory interface *features*. Each feature may have associated accessory design requirements and recommendations; an accessory must comply with all feature-specific requirements to properly support the feature. Some features require other features to be implemented.

Accessory design must take into account the possibility that an Apple device may not support all of the accessory's implemented features and react appropriately.

2 Core Concepts

The HomeKit Accessory Specification defines how an Apple device communicates with an accessory using HomeKit Accessory Protocol, or HAP. For example, an iOS device uses HAP to discover, explore, and interact with HomeKit accessories such as lights, door locks, and garage door openers.

2.1 Transports

HomeKit Accessory Protocol supports two transports: Bluetooth LE and IP. HomeKit accessories may support either transport or both transports. The transport-specific protocols are covered in "6 HomeKit Accessory Protocol for IP Accessories" (page 66) and "7 HomeKit Accessory Protocol for Bluetooth LE Accessories" (page 101).

2.2 Security

HomeKit Accessory Protocol sessions are end-to-end encrypted and mutually authenticated. Sessions also feature perfect forward secrecy, meaning that a new, unique key is generated for every session. The foundation of this security is provided by Pairing.

2.2.1 Pairing

Pairing establishes a cryptographic relationship between an iOS device and an accessory. There are two components to Pairing: Pair Setup and Pair Verify.

2.2.1.1 Pair Setup

Pair Setup is a one-time operation that creates a valid pairing between an iOS device and an accessory by securely exchanging public keys with an iOS device and an accessory. Pair Setup requires the customer to enter an eight-digit Setup Code on their iOS device. The Setup Code is provided by the accessory via a label or display.

2.2.1.2 Pair Verify

Pair Verify is performed for every HomeKit Accessory Protocol session. Pair Verify verifies the pairing between an iOS device and an accessory and establishes an ephemeral shared secret used to secure the HomeKit Accessory Protocol session.

2.2.2 Session Keys

Encryption and authentication keys are derived from the ephemeral shared secret established during Pair Verify.

2.3 Attributes

HomeKit Accessory Protocol uses two types of attributes, services and characteristics, to model the capabilities of an accessory.

2.3.1 Accessories

Accessories are comprised of services and characteristics. An example of an accessory is a ceiling fan with a light and a mister that sprays cool water.

2.3.2 Services

Services group functionality in order to provide context. In the aforementioned example accessory there are three services: a fan service to interact with the ceiling fan, a light service to interact with the light, and a mister service to interact with the spray mister.

2.3.2.1 Service Naming

Not all services provide user-visible or user-interactive functionality. Services which provide either user-visible or user-interactive functionality must include the Name characteristic, see "10.24 Name" (page 187); all other services must not include this characteristic. This convention is used by iOS controllers to determine which services to display to users.

In the aforementioned ceiling fan example, the fan, light, and mister services would all include a Name characteristic. There may be an additional service providing firmware update capabilities. The characteristics contained in this service are not meant to be user-visible or user-interactive, thus this service would not contain a Name characteristic.

Note that the Accessory Information service, see "9.1 Accessory Information" (page 153), is an exception and always includes the Name characteristic even though it is not typically user-visible or user-interactive.

2.3.2.2 Extending Services

To maintain backward compatibility with earlier clients, any characteristics added in later versions of a service must be optional. Later versions of a service must not change behaviors defined in previous versions of the service.

2.3.2.3 Primary Service

Accessories should list one of its services as the primary service. The primary service must match the primary function of the accessory and must also match with the accessory category. An accessory must expose only one primary service from its list of available services.

2.3.2.4 Hidden Service

Accessories may specify the services that are to be hidden from users by a generic HomeKit application. Accessories may expose several services that could be used to configure the accessory or to update firmware on the accessory, these services should be marked as hidden. When all characteristics in a service are marked hidden then the service must also be marked as hidden.

2.3.2.5 Linked Service

Linked services allows accessories to specify logical relationship between services. A service can link to one or more services. A service must not link to itself. Service links have context and meaning only to the first level of services that it links to. For example if Service A links to Service B, and service B links to Service C, this does not imply any relation between Service A to Service C. If Service A also relates to Service C then Service A’s linked services must include both Service B and Service C. Linked services allows applications to display logically grouped accessory controls in the UI.

2.3.3 Characteristics

A characteristic is a feature that represents data or an associated behavior of a service. The characteristic is defined by a universally unique type, and has additional properties that determine how the value of the characteristic can be accessed.

- The properties of ”perms” and ”ev”, for example, indicate read/write/notify permissions, and event notifications.
- The characteristic may also have other properties that further describe the characteristic. Properties like ”format” describe the format of the characteristic’s value such as int or string, ”description” contains a description string, and ”minValue”, ”maxValue”, and ”minStep” refer to minimum, maximum, and step limits that also apply to the characteristic’s value.
- For example, the ”9.5 Lightbulb” (page 154) service contains a ”10.4 Brightness” (page 178) characteristic of type ”public.hap.characteristic.brightness” with permissions of paired read (”pr”) and paired write (”pw”). The ’ ”10.4 Brightness” (page 178) characteristic may have additional properties describing the value. For example, a ”format” property can indicate that the value is an int number, and a ”unit” property can indicate that the value is in percentage. Furthermore, other properties can be used to indicate a minimum value of 0, maximum value of 100, and step value of 1 via the ”minValue”, ”maxValue”, and ”minStep” properties.

In the aforementioned example accessory the fan service may have the following characteristics:

- ”10.26 On” (page 188): This characteristic is a Boolean value that represents the power state of the fan: a `true` value indicates the fan is turned on and a `false` value indicates the fan is turned off.
- ”10.29 Rotation Speed” (page 189): This characteristic is an Integer value that represents the percentage of rotation speed the fan is currently using.
- ”10.28 Rotation Direction” (page 189): This characteristic is an Enumerated value that represents the rotation direction of the fan. A `0` value indicates the fan is rotating clockwise and a `1` value indicates the fan is rotating counter-clockwise.

The "9.5 Lightbulb" (page 154) may have the following characteristics:

- "10.26 On" (page 188): This characteristic is a Boolean value that represents the power state of the light: a `true` value indicates the light is turned on and a `false` value indicates the light is turned off.

Note that the same "10.26 On" (page 188) characteristic is used in both the fan and the light services. The protocol-level interaction between an iOS device and an accessory is the same but the different services provide context for the characteristic.

2.3.3.1 Valid Characteristic Values

Accessory characteristics that support only a sub-set of the Apple Defined enum values can indicate the supported values as part of the characteristic's metadata.

2.3.3.2 Additional Authorization Data

Certain types of characteristics may support additional authorization data for write requests by default. Additional authorization data is controller-provided data that the accessory may use to validate that the controller is authorized to perform a requested operation. The contents of the authorization data are manufacturer specific. The additional authorization data is provided by a third-party app to iOS and stored by iOS on the controller, so it must not be required to be unique per write request as the controller will not construct or retrieve unique authorization data for each request. Additional authorization data may change periodically, e.g. once per month, or when user permissions change.

The characteristic and service combination that support additional authorization data by default are "10.32 Target Door State" (page 191) in the "9.4 Garage Door Opener" (page 154) and the "10.19 Lock Target State" (page 186) in the "9.4 Garage Door Opener" (page 154) and "9.7 Lock Mechanism" (page 155).

For example, a HomeKit accessory implementing an exterior lock, see "9.7 Lock Mechanism" (page 155), may wish to allow control of the lock by all primary users of the home at any time. This user group has additional authorization data 'A'. Another group of secondary users are only allowed control of the lock during specific times of the day, for instance 9:00 AM to 6:00 PM; this user group has additional authorization data 'B'. When a primary user's controller writes to the Target Lock State, "10.19 Lock Target State" (page 186), the controller will include the additional authorization data 'A' along with the write request. The lock can verify that the controller is authorized to perform the write operation at any time of the day based on the inclusion of additional authorization data 'A'. However, if, for instance, a secondary user's controller attempts to write to the Target Lock State at 10:00 PM, this write request will include the additional authorization data 'B', and the accessory could then reject the write as not authorized at that time based on the inclusion of additional authorization data 'B'.

2.3.4 Additional HomeKit Accessory Protocol Requirements

The following is a summary of additional HomeKit Accessory Protocol requirements

Table 2-1: Additional HomeKit Accessory Protocol Requirements

Feature	Requirement	Description
Primary Service	Optional	Accessories may choose to indicate a primary service
Linked Service	Optional	
Hidden Service	Conditional	If the accessory exposes custom services for proprietary controls on the accessory then it must include a Hidden service
Valid Values / Valid Values range	Conditional	When a characteristic supports only a subset of the Apple defined enum values then it must include Valid Values / Valid Values Range
Accessory Flags	Conditional	If accessory requires additional setup or configuration using the accessory manufacturer's app then it must include Accessory Flags

2.4 Profiles

Profiles define the appropriate services and characteristics to be used to provide consistent behavior. For example, a Light profile uses the "9.5 Lightbulb" (page 154) that requires a "10.26 On" (page 188) characteristic. The Light profile takes services a step further by mandating that a light must stop illuminating entirely when the "10.26 On" (page 188) is set to a value of `false`.

2.5 Roles

2.5.1 HAP Client

- Always the controller.
- Send requests and receive responses from HAP accessory servers.
- Register for and receive notifications from HAP accessory servers.

2.5.2 HAP Accessory Server

An HAP accessory server is a device that supports HomeKit Accessory Protocol and exposes a collection of accessories to the HAP controller(s). An HAP accessory server represents one endpoint of the pairing relationship established with HAP Pairing, as described in "5 Pairing" (page 38) and exposes at least one HAP accessory object.

- Always the accessory.
- Expose attributes that can be accessed by HAP clients.
- Accept incoming requests from clients and send responses.
- Send notifications to registered clients.

2.5.3 HAP Accessory Objects

An HAP accessory object represents a physical accessory on an HAP accessory server. For example, a thermostat would expose a single HAP accessory object that represents the user-addressable functionality of the thermostat.

2.5.3.1 Required Services

An HAP accessory object must include at least one service, public.hap.service.accessory-information, as defined in the "11 Apple-defined Profiles" (page 255).

2.5.3.2 Bridges

A bridge is a special type of HAP accessory server that bridges HomeKit Accessory Protocol and different RF/transport protocols, such as ZigBee or Z-Wave. A bridge must expose all the user-addressable functionality supported by its connected devices as HAP accessory objects to the HAP controller(s). A bridge must ensure that the instance ID assigned to the HAP accessory objects exposed on behalf of its connected devices do not change for the lifetime of the server/client pairing.

For example, a bridge that bridges three lights would expose four HAP accessory objects: one HAP accessory object that represents the bridge itself that may include a "firmware update" service, and three additional HAP accessory objects that each contain a "lightbulb" service.

A bridge must not expose more than 150 HAP accessory objects.

Exposing a new accessory from an existing bridge requires an MFi Product Plan update.

Any accessories, regardless of transport, that enable physical access to the home, such as door locks, must not be bridged. Accessories that support IP transports, such as Wi-Fi, must not be bridged. Accessories that support Bluetooth LE that can be controlled, such as a light bulb, must not be bridged. Accessories that support Bluetooth LE that only provide data, such as a temperature sensor, and accessories that support other transports, such as a ZigBee light bulb or a proprietary RF sensor, may be bridged.

2.5.3.3 Primary HAP Accessory Object

The HAP accessory object with an instance ID of 1 is considered the primary HAP accessory object. For bridges, this must be the bridge itself.

2.5.3.4 Colocation

The services contained within an HAP accessory object must be collocated. For example, a fan with a light on it would expose single HAP accessory object with three services: the required "accessory information" service, a "fan" service, and a "lightbulb" service. Conversely, a bridge that bridges two independent lights that may be in different physical locations must expose an HAP accessory object for each independent light.

2.6 Accessory Attribute Database

The accessory attribute database is a list of HAP Accessory objects, Service objects, and Characteristic objects.

2.6.1 Instance IDs

Instance IDs are numbers with a range of [1, 18446744073709551615] for IP accessories (see "7.4.4.2 Instance IDs" (page 135) for BLE accessories). These numbers are used to uniquely identify HAP accessory objects within an HAP accessory server, or uniquely identify services, and characteristics within an HAP accessory object. The instance ID for each object must be unique for the lifetime of the server/client pairing.

2.6.1.1 Accessory Instance IDs

Accessory instance IDs, "aid", are assigned from the same number pool that is global across entire HAP Accessory Server. For example, if the first Accessory object has an instance ID of "1" then no other Accessory object can have an instance ID of "1" within the Accessory Server.

2.6.1.2 Service and Characteristic Instance IDs

Service and Characteristic instance IDs, "iid", are assigned from the same number pool that is unique within each Accessory object. For example, if the first Service object has an instance ID of "1" then no other Service or Characteristic objects can have an instance ID of "1" within the parent Accessory object. The Accessory Information service must have a service instance ID of 1.

After a firmware update services and characteristics types that remain unchanged must retain their previous instance ids, newly added services and characteristics must not reuse instance ids from services and characteristics that were removed in the firmware update.

3 Requirements

3.1 Overview

Accessories that support HomeKit Accessory Protocol (HAP) must conform to the following requirements along with any feature specific requirements contained in their respective chapters.

3.2 General Requirements

An accessory that supports HAP:

- must incorporate an Apple Authentication Coprocessor, version 2.0C, 3.0 or "3.6 Software Authentication" (page 30). Earlier versions of the Apple Authentication Coprocessor are not allowed.
- must comply with the latest version of the *Accessory Interface Specification* when the accessory supports additional MFi licensed technologies.
- must be able to be setup using Apple Home app out of the box and after every factory reset without requiring additional steps.
  - Some accessories may require an additional physical step for setup through "10.84 Accessory Flags" (page 227).Use of Accessory Flags requires written approval by Apple in advance.
- must support accessory setup as defined in "4 Accessory Setup" (page 32).
- may expose other interfaces, such as a public API.
- must enable the end user to access and use its HomeKit functionality without requiring the end user to pay an additional fee to the accessory manufacturer or developer, or any other third party, after the point of sale of the accessory.

3.3 Wi-Fi Requirements

Accessories which use Wi-Fi for HAP communication:

- must be listed as a Wi-Fi CERTIFIED product with the Wi-Fi Alliance.
- must be compatible with the current Apple AirPort Base Station and Time Capsule products.
- must support Wi-Fi Accessory Configuration 2 (WAC2) to get Wi-Fi credentials.

3.3.1 WAC2

3.3.1.1 Requirements

WAC2 requires iOS 11.3 or later.

Accessories must support the following additional requirements in addition to those specified in *Accessory Interface Specification*:

Table 3-1: Additional Apple Device Information Element (IE) Flags

Value	Bit	Requirement	Description
`0x40`	1	Required	Accessory is not Configured.
`0x20`	2	Not Allowed	Supports MFi Configuration v1.
`0x000040`	17	Required	Supports HomeKit Accessory Protocol v1.
`0x000008`	20	Required	Supports WAC2 for HomeKit Accessories.
`0x000004`	21	Required, if bit 22 is not set.	Supports Apple Authentication Coprocessor.
`0x000002`	22	Required, if bit 21 is not set.	Supports Software-based Authentication.

3.3.1.2 Procedure

1. Accessories must support the following Apple Device IE elements (Apple Device IE elements are defined in *Apple Device Information Elements* section of the *Accessory Interface Specification*):
    - *0x09* (Setup Hash). The Setup Hash element is defined as Type:0x09, Length:4, Value: Setup Hash (see "4.2.3 Setup Hash" (page 33)). *Note: Element ID 0x09 is currently not defined in Accessory Interface Specification.*
2. Accessories must use `_hap._tcp` "6.4 Discovery" (page 71) Bonjour service instead of `_mfi-config._tcp`.
3. The `deviceid` used in the Bonjour service during WAC must be same as the `Device ID` used in the Bonjour service used for HomeKit discovery and must conform to the requirements in "5.4 Device ID" (page 38).
4. "5.6 Pair Setup" (page 41) is used to establish a secure WAC pairing session using "4.2.1 Setup Code" (page 32).
5. Accessories which support software authentication, must do the software authentication procedures as defined in "5.15 Software Authentication Procedure" (page 58).
6. After "5.6 Pair Setup" (page 41), "5.7 Pair Verify" (page 49) is used to start a secure session.
7. Accessories must apply the session keys derived from "5.7 Pair Verify" (page 49) to encrypt/decrypt all traffic for the existing accessory server connection, which includes Wi-Fi Configuration 'config TLVs' and continue with the WAC procedure as defined in section *Wi-Fi Accessory Configuration Setup Experience* of the *Accessory Interface Specification* to join the new Wi-Fi network.
8. Upon completion of WAC procedure, accessory must clear bit 2 of Bonjour TXT Status Flag as defined in "6-8 Bonjour TXT Status Flags" (page 72).

3.3.1.3 Reconfiguration

Accessories should provide a mechanism for the user to reset Wi-Fi configuration without removing HomeKit pairing. When the user requests to reset Wi-Fi configuration without removing HomeKit pairing the accessory must enter WAC mode in unconfigured state i.e.,accessory must set bit 2 of Bonjour TXT Status Flag as defined in "6-8 Bonjour TXT Status Flags" (page 72). It must also set *Device is paired to a HomeKit controller* flag as bit 9 (0x0040) and follow "3.3.1.2 Procedure" (page 29) from step 4.

3.4 Firmware Updates

- Accessories must support OTA firmware updates.
- All firmware images must be authenticated and verified by the accessory using a non-trivial mechanism.
- Updated firmware must complete MFi self-certification requirements prior to release.
- Accessories must not allow a firmware image to be downgraded after a successful firmware update.
- Accessories must increment the `config number (CN)` (For IP accessories refer to "6.4 Discovery" (page 71) and for Bluetooth accessories refer to "7.4.2.1 HAP BLE Regular Advertisement Format" (page 131))after a firmware update.
- Accessories must maintain the instance ids for services and characteristics that remain unchanged after a firmware update.
- Accessories must not reuse an instance id from a service or characteristics that was removed by the current firmware update.

3.5 Coalescing Requirements

An accessory that supports multiple transports (eg. HomeKit Accessory Protocol for IP, HomeKit Accessory Protocol for Bluetooth LE, HomeKit Accessory Protocol over iCloud) must ensure that the following information is same across the different transports:

- The set of accessories and the accessory instance IDs of each of the accessories.
- The set of services and the service instance IDs of each of the services with in an accessory.
- The set of HAP characteristics and the HAP characteristic instance IDs of each of the characteristics with in a service.

3.6 Software Authentication

Software token-based authentication requires iOS 11.3 or later.

- Refer to *HomeKit Software Authentication Server Specification* on how to obtain software tokens.

- Software tokens must be provisioned in the accessory via Factory Provisioning (at the time of manufacturing) or In-Field Provisioning (through a firmware update).
- The token must be decoded using base 64 and provisioned in raw data format in the accessory.
- The UUID generated by the Licensee server for each token provisioned must be registered with Apple Server as defined in *HomeKit Software Authentication Server Specification* before provisioning in the accessory in little endian format (16 bytes).
- Software tokens must be stored in secure storage of the accessory such as secure element or secure enclave or "5.2 Cryptographic Key Storage" (page 38). A minimum of 1024 bytes of memory must be allocated for storing a token.
- Provisioned software token must persist through factory reset including manual reset by user.
- Provisioned software token will be activated as part of the HomeKit Pairing process, as defined in "5 Pairing" (page 38).

4 Accessory Setup

4.1 Overview

This chapter describes how the HomeKit setup payload information is generated, stored, displayed and delivered by the accessory to the controller for pairing purposes.

4.2 Setup Payload

4.2.1 Setup Code

The Setup Code must conform to the format XXXXXXXX where each X is a 0-9 digit - for example, `10148005`. For the purposes of generating accessory SRP verifier (see "5.6 Pair Setup" (page 41)), the setup code must be formatted as `XXX–XX–XXX` (including dashes). In this example, the format of setup code used by the SRP verifier must be `101–48–005`.

4.2.1.1 Generation of Setup Code

If an accessory generates a random Setup Code, for eg. to display, it must conform to the following rules:

- The accessory must generate a new Setup Code each time it is needed.
- The accessory must generate Setup Codes from a cryptographically secure random number generator.
- Setup codes must not be derived from public information, such as a serial number, manufacturer date, MAC address, region of origin, etc.

If an accessory cannot generate a dynamic Setup Code, then it must conform to the following rules:

- A random Setup Code must be generated for each individual accessory.
- The manufacturing process must use a cryptographically secure random number generator.
- The accessory must be manufactured with an SRP verifier for the Setup Code rather than the raw Setup Code.
- Setup codes must not be derived from public information, such as a serial number, date of manufacture, MAC address, region of origin, etc.

4.2.1.2 Invalid Setup Codes

The following are examples of Setup Codes that must not be used due to their trivial, insecure nature:

- 00000000

- 11111111
- 22222222
- 33333333
- 44444444
- 55555555
- 66666666
- 77777777
- 88888888
- 99999999
- 12345678
- 87654321

4.2.2 Setup ID

This identifier is an alphanumeric string of 4(0-9, A-Z) characters. This identifier is persistent across reboots and factory reset of the accessory. This identifier must be different than the DeviceID, serial number, model or accessory name and must be random for each accessory instance manufactured by an accessory manufacturer.

4.2.3 Setup Hash

The Setup Hash must be included in the transport specific advertisement data (WiFi - "3.3 Wi-Fi Requirements" (page 28), IP - Table 6-7 (page 72), BLE - "7.4.2.1 HAP BLE Regular Advertisement Format" (page 131)) of the accessories. This hash is derived from the "4.2.2 Setup ID" (page 33) and "5.4 Device ID" (page 38). The procedure for setup hash generation is described in "4.2.3.1 Setup Hash Generation" (page 33).

4.2.3.1 Setup Hash Generation

The "5.4 Device ID" (page 38) should be converted to uppercase for the computation of the setup hash. SHA-512 is used to generate the setup hash from a concatenation of the "4.2.2 Setup ID" (page 33) followed by the "5.4 Device ID" (page 38). The setup hash is the first 4 bytes of the SHA-512 digest.

Examples: The SHA-512 digest for an accessory with Setup ID "7OSX" and a deviceID: E1:91:1A:70:85:AA is:

```
C9FE1BCFB89B86E218AB56C5F1986EE9B9CC954A06C3769AC6433D472793EA7189DE46F
2A0E043363D5E1E4401D066F17AAC6D9C9A993F439879A331CF55F6CB
```

The Setup Hash is:

```
C9FE1BCF
```

If the deviceID (after accessory factory reset) changes to: C8:D8:58:C6:63:F5 then the SHA-512 digest is:

```
EF5D8E9BD0635D10CBF528F1DC968D7E7A9380BD7AA216A0AFFD264D36FCF632CA58B4
F8618D738E81FE01B3F7D03BBAB03C59553EC378720312FF66B49A58A3
```

In this case, the Setup Hash is:

```
    EF5D8E9B
```

BLE and WAC must include the setup hash as a binary representation in the advertisement data whereas IP advertisement data includes this in base64-encoded form.

4.3 Format of Setup Payload

The information encoded in the setup payload is a URI with the following format:

```
X-HM://<Setup Payload encoding>
```

The setup payload encoding contains the following information:

```
<VersionCategoryFlagsAndSetupCode><SetupID>
```

`<VersionCategoryFlagsAndSetupCode>` is of 9 characters and represents the following 46-bit value encoded as a base-36 string:

- Bits 45-43 is a 3-bit version field - must be set to 0'b000
- Bits 42-39 is a 4-bit reserved field - must be set to 0'b0000
- Bits 38-31 is a 8-bit accessory category (the lower 8-bits of the accessory category, "14-1 Accessory Categories" (page 271))
- Bit 30 - Accessory supports Wi-Fi Accessory Configuration (WAC) for configuring wireless credentials
- Bit 29 - Accessory supports HAP over BLE transport
- Bit 28 - Accessory supports HAP over IP transport; MUST be ON if Bit 30 is ON
- Bit 27 - Accessory is paired with a controller (only for accessories using programmable NFC tags to advertise the setup payload)
- Bits 26-0 - 27-bit setupCode, "4.2.1 Setup Code" (page 32) (maximum of 27-bits to encode 8-digit setup code)

`<SetupID>` is of 4 alphanumeric characters (0-9, A-Z), "4.2.2 Setup ID" (page 33)

An example of enhanced setup payload encoding is:

```
X-HM://007JNU5AE7OSX
```

This example describes an accessory with the following information:

- WAC support: YES
- IP support: (MUST be ON) YES
- Accessory Category: 7 (Outlet)
- Setup Code: 518-08-582 (bit representation is 3168946)
- SetupID: 7OSX

An accessory with the same setup parameters but only supporting HAP over BLE transport will result in setup payload of:

```
X-HM://0076CDMX27OSX
```

4.4 Retrieval of Setup Payload

The setup payload must be retrievable by the controller from the accessory using either "4.4.1 QR Codes" (page 35) or "4.4.2 NFC" (page 35).

4.4.1 QR Codes

QR code may be used to store the setup payload.

4.4.1.1 Requirements

- The QR code generated must be of Version 1.0 or higher
- QR code design is defined in ISO/IEC 180004 First Edition 2000-06-15
- Must use level M or higher ECC.

Please refer to *Works with Apple HomeKit Identity Guidelines* for more details on identity guidelines for including QR codes on the accessory.

4.4.2 NFC

NFC tags may be used to store the setup payload. The user can tap the iOS controller to the accessory to obtain the setup payload.

4.4.2.1 Requirements

- NFC tags will use the NFC Data Exchange Format (NDEF) as defined by *NFC Forum™* in *NDEF 1.0 NFCForum-TS-NDEF 1.0*
- An NDEF message is defined as a group of individual NDEF records as defined by *NFC Forum™* in NFC Record Type Definition (RTD) RTD 1.0 *NFCForum-TS-RTD 1.0*
- Setup payload for NFC tags will use NDEF URI Record Type Definition as defined by *NFC Forum™* in URI Record Type Definition RTD-URI 1.0 *NFCForum-TS-RTD URI 1.0*. The URI for defined in section 3.3 Format
- Minimum payload that must be supported is 30 bytes
- NFC tag types must be Type 2 or greater
- Setup payload should be scannable from the NFC tags when the accessory is not powered on
- NFC tag should not be scannable when the HomeKit Accessory is still in the packaging
- The setup payload must be scannable when holding the top of the iOS controller near the center of the NFC tag on the accessory. Recommended NFC tag performance guidelines are defined by *NFC Forum™* in *Tag Performance Requirements Document*
- The center of the NFC tag must be indicated on the accessory using the *Apple HomeKit Identity Guidelines*
- One of the following NFC tags must be used based on different "14-1 Accessory Categories" (page 271).
  - Read only: tag data programmed at manufacturer's facility and cannot be modified once programmed.
  - Programmable Tag
    * Not in pairing mode:
      - The "4.2.2 Setup ID" (page 33) and the "4.2.1 Setup Code" (page 32) must not be encoded in the URI.
      - If the accessory is not paired and not in pairing mode it must not advertise the "4.2.2 Setup ID" (page 33) and "4.2.1 Setup Code" (page 32) in the URI. e.g., `X-HM://007ISZPJ40000` for the full data blob `X-HM://007JNU5AE7OSX`.
    * Powered on and pairing mode:
      - Complete data blob must be presented. e.g., `X-HM://007JNU5AE7OSX`
      - The accessory must terminate the pairing mode after 5 minutes of inactivity and move to 'Not in pairing mode'. Pair-setup can still be allowed after the 5 minute timer expires.
      - If pairing completes, it must move to 'Paired'.
    * Paired:
      - Once HomeKit pairing is completed the NFC tag must not encode the SetupID and Setup Code in the payload. When the accessory is paired, bit 27 is set to 1 in the URI. e.g `X-HM://007L0WGLC0000`.
    * Programming of NFC Tag must not happen over wireless link.
    * The setup code must not change when the NFC tag is re-programmed.
    * The accessory must require users to explicitly trigger NFC pairing mode via a physical interaction on the accessory.
    * Accessories implementing programmable NFC must not implicitly be ready to pair once all pairings have been removed.

4.4.2.2 Example

Following is the NDEF payload for the URI:

```
X-HM://007JNU5AE7OSX
```

Table 4-1: NDEF Representation

Offset	Content	Description
0	0xD1	SR =1, TNF = 0x01, MB=1, ME=1
1	0x01	Length go the Record Type (1 byte)
2	0x15	Length of the payload (20 bytes)
3	0x55	The Record Name ("U")
4	0x00	No abbreviation
5	0x58 0x2D 0x48 0x4D 0x3A 0x2F 0x2F 0x30 0x30 0x37 0x4A 0x4E 0x55 0x35 0x41 0x45 0x37 0x4F 0x53 0x58	The string: X-HM://007JNU5AE7OSX

Please refer to *Works with Apple HomeKit Identity Guidelines* for more details on identity guidelines for including NFC tags on the accessory.

5 Pairing

5.1 Overview

This chapter describes the process of cryptographically pairing devices. It has the following features:

- Provides end-to-end security without exposing secret information to external entities.
- Secret keys are generated on the accessory and must not leave the accessory where they are used.
- Does not rely on link layer security (i.e. safe to run over open Wi-Fi networks).
- Is transport neutral (e.g. can work over Wi-Fi/IP, Bluetooth, etc.).
- Is practical to implement and is efficient even in resource-constrained accessories.
- Keys are revokable without cooperation from the peer.

5.2 Cryptographic Key Storage

Keys must be stored in a secure manner to prevent unauthorized access. Keys must not be accessible in any way from outside the device. The recommended mechanism is to generate, store, and operate on keys only within a Secure Element. The only operation requiring a secret key is signing and that can be performed inside the Secure Element to minimize attacks. Pairing also requires storing the public key of each paired peer. These keys, although public, must also be stored securely because they are used to verify pairing relationships. For example, insecure storage could allow a pairing to be added without going through the pair setup process or deny legitimate use of a pairing. If a device is physically reset, all cryptographic keys must be erased.

5.3 Admins

Admins are pairings that have the 'admin bit' set. Admins are exclusively authorized to add, remove, and list pairings. The Apple ID associated with the first iOS device to perform Pair Setup with a new, unpaired accessory automatically has the admin bit set.

5.4 Device ID

This identifier of the accessory must be a unique random number generated at every factory reset and must persist across reboots.

5.5 Secure Remote Password (SRP)

5.5.1 SRP Modifications

Pairing uses Stanford's Secure Remote Password protocol with the following modifications:

- SHA-512 is used as the hash function, replacing SHA-1. If the SRP reference implementation provided by Stanford is being used then the function that generates the Session Key, K, from the Premaster Secret, S, must be changed from Mask Generation Function 1, `t_mgf1()`, to the specified hash function, SHA-512.
- The Modulus, N, and Generator, g, are specified by the 3072-bit group of RFC 5054.

5.5.2 SRP Test Vectors

The following test vectors demonstrate calculation of the Verifier (v), Premaster Secret (S), and Session Key (K).

```
# Modulus (N), as specified by the 3072-bit group of RFC 5054
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08 8A67CC74
020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B 302B0A6D F25F1437
4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D C2007CB8 A163BF05
98DA4836 1C55D39A 69163FA8 FD24CF5F 83655D23 DCA3AD96 1C62F356 208552BB
9ED52907 7096966D 670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B
E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9 DE2BCBF6 95581718
3995497C EA956AE5 15D22618 98FA0510 15728E5A 8AAAC42D AD33170D 04507A33
A85521AB DF1CBA64 ECFB8504 58DBEF0A 8AEA7157 5D060C7D B3970F85 A6E1E4C7
ABF5AE8C DB0933D7 1E8C94E0 4A25619D CEE3D226 1AD2EE6B F12FFA06 D98A0864
D8760273 3EC86A64 521F2B18 177B200C BBE11757 7A615D6C 770988C0 BAD946E2
08E24FA0 74E5AB31 43DB5BFC E0FD108E 4B82D120 A93AD2CA FFFFFFFF FFFFFFFF

# Generator (g), as specified by the 3072-bit group of RFC 5054
05

# Username (I), as an ASCII string without quotes
"alice"

# Password (p), as an ASCII string without quotes
"password123"

# A private (a)
60975527 035CF2AD 1989806F 0407210B C81EDC04 E2762A56 AFD529DD DA2D4393

# A public (A)
FAB6F5D2 615D1E32 3512E799 1CC37443 F487DA60 4CA8C923 0FCB04E5 41DCE628
0B27CA46 80B0374F 179DC3BD C7553FE6 2459798C 701AD864 A91390A2 8C93B644
ADBF9C00 745B942B 79F9012A 21B9B787 82319D83 A1F83628 66FBD6F4 6BFC0DDB
```

```
2E1AB6E4 B45A9906 B82E37F0 5D6F97F6 A3EB6E18 2079759C 4F684783 7B62321A
C1B4FA68 641FCB4B B98DD697 A0C73641 385F4BAB 25B79358 4CC39FC8 D48D4BD8
67A9A3C1 0F8EA121 70268E34 FE3BBE6F F89998D6 0DA2F3E4 283CBEC1 393D52AF
724A5723 0C604E9F BCE583D7 613E6BFF D67596AD 121A8707 EEC46944 95703368
6A155F64 4D5C5863 B48F61BD BF19A53E AB6DAD0A 186B8C15 2E5F5D8C AD4B0EF8
AA4EA500 8834C3CD 342E5E0F 167AD045 92CD8BD2 79639398 EF9E114D FAAAB919
E14E8509 89224DDD 98576D79 385D2210 902E9F9B 1F2D86CF A47EE244 635465F7
1058421A 0184BE51 DD10CC9D 079E6F16 04E7AA9B 7CF7883C 7D4CE12B 06EBE160
81E23F27 A231D184 32D7D1BB 55C28AE2 1FFCF005 F57528D1 5A88881B B3BBB7FE

# B private (b)
E487CB59 D31AC550 471E81F0 0F6928E0 1DDA08E9 74A004F4 9E61F5D1 05284D20

# B public (B)
40F57088 A482D4C7 733384FE 0D301FDD CA9080AD 7D4F6FDF 09A01006 C3CB6D56
2E41639A E8FA21DE 3B5DBA75 85B27558 9BDB2798 63C56280 7B2B9908 3CD1429C
DBE89E25 BFBD7E3C AD3173B2 E3C5A0B1 74DA6D53 91E6A06E 465F037A 40062548
39A56BF7 6DA84B1C 94E0AE20 8576156F E5C140A4 BA4FFC9E 38C3B07B 88845FC6
F7DDDA93 381FE0CA 6084C4CD 2D336E54 51C464CC B6EC65E7 D16E548A 273E8262
84AF2559 B6264274 215960FF F47BDD63 D3AFF064 D6137AF7 69661C9D 4FEE4738
2603C88E AA098058 1D077584 61B777E4 356DDA58 35198B51 FEEA308D 70F75450
B71675C0 8C7D8302 FD7539DD 1FF2A11C B4258AA7 0D234436 AA42B6A0 615F3F91
5D55CC3B 966B2716 B36E4D1A 06CE5E5D 2EA3BEE5 A1270E87 51DA45B6 0B997B0F
FDB0F996 2FEE4F03 BEE780BA 0A845B1D 92714217 83AE6601 A61EA2E3 42E4F2E8
BC935A40 9EAD19F2 21BD1B74 E2964DD1 9FC845F6 0EFC0933 8B60B6B2 56D8CAC8
89CCA306 CC370A0B 18C8B886 E95DA0AF 5235FEF4 393020D2 B7F30569 04759042

# Salt (s)
BEB25379 D1A8581E B5A72767 3A2441EE

# Verifier (v)
9B5E0617 01EA7AEB 39CF6E35 19655A85 3CF94C75 CAF2555E F1FAF759 BB79CB47
7014E04A 88D68FFC 05323891 D4C205B8 DE81C2F2 03D8FAD1 B24D2C10 9737F1BE
BBD71F91 2447C4A0 3C26B9FA D8EDB3E7 80778E30 2529ED1E E138CCFC 36D4BA31
3CC48B14 EA8C22A0 186B222E 655F2DF5 603FD75D F76B3B08 FF895006 9ADD03A7
54EE4AE8 8587CCE1 BFDE3679 4DBAE459 2B7B904F 442B041C B17AEBAD 1E3AEBE3
CBE99DE6 5F4BB1FA 00B0E7AF 06863DB5 3B02254E C66E781E 3B62A821 2C86BEB0
D50B5BA6 D0B478D8 C4E9BBCE C2176532 6FBD1405 8D2BBDE2 C33045F0 3873E539
48D78B79 4F0790E4 8C36AED6 E880F557 427B2FC0 6DB5E1E2 E1D7E661 AC482D18
E528D729 5EF74372 95FF1A72 D4027717 13F16876 DD050AE5 B7AD53CC B90855C9
39566483 58ADFD96 6422F524 98732D68 D1D7FBEF 10D78034 AB8DCB6F 0FCF885C
C2B2EA2C 3E6AC866 09EA058A 9DA8CC63 531DC915 414DF568 B09482DD AC1954DE
C7EB714F 6FF7D44C D5B86F6B D1158109 30637C01 D0F6013B C9740FA2 C633BA89

# Random Scrambling Parameter (u)
03AE5F3C 3FA9EFF1 A50D7DBB 8D2F60A1 EA66EA71 2D50AE97 6EE34641 A1CD0E51
C4683DA3 83E8595D 6CB56A15 D5FBC754 3E07FBDD D316217E 01A391A1 8EF06DFF
```

```
# Premaster Secret (S)
F1036FEC D017C823 9C0D5AF7 E0FCF0D4 08B009E3 6411618A 60B23AAB BFC38339
72682312 14BAACDC 94CA1C53 F442FB51 C1B027C3 18AE238E 16414D60 D1881B66
486ADE10 ED02BA33 D098F6CE 9BCF1BB0 C46CA2C4 7F2F174C 59A9C61E 2560899B
83EF6113 1E6FB30B 714F4E43 B735C9FE 6080477C 1B83E409 3E4D456B 9BCA492C
F9339D45 BC42E67C E6C02C24 3E49F5DA 42A869EC 855780E8 4207B8A1 EA6501C4
78AAC0DF D3D22614 F531A00D 826B7954 AE8B14A9 85A42931 5E6DD366 4CF47181
496A9432 9CDE8005 CAE63C2F 9CA4969B FE840019 24037C44 6559BDBB 9DB9D4DD
142FBCD7 5EEF2E16 2C843065 D99E8F05 762C4DB7 ABD9DB20 3D41AC85 A58C05BD
4E2DBF82 2A934523 D54E0653 D376CE8B 56DCB452 7DDDC1B9 94DC7509 463A7468
D7F02B1B EB168571 4CE1DD1E 71808A13 7F788847 B7C6B7BF A1364474 B3B7E894
78954F6A 8E68D45B 85A88E4E BFEC1336 8EC0891C 3BC86CF5 00978801 78D86135
E7287234 58538858 D715B7B2 47406222 C1019F53 603F0169 52D49710 0858824C

# Session Key (K)
5CBC219D B052138E E1148C71 CD449896 3D682549 CE91CA24 F098468F 06015BEB
6AF245C2 093F98C3 651BCA83 AB8CAB2B 580BBF02 184FEFDF 26142F73 DF95AC50
```

5.6 Pair Setup

Pair Setup requires the user to enter the accessory's password on the iOS device to bidirectionally authenticate. The Pair Setup process follows the Secure Remote Password protocol. This process assumes that the accessory is unpaired and in a mode that enables pairing to be initiated.

All Pairing Identifiers in Pair Setup are case-sensitive.

Every accessory must support a manufacturer-defined mechanism to restore itself to a "factory reset" state where all pairing information is erased and restored to factory default settings. This mechanism should be easily accessible to a user, e.g, a physical button or a reset code.

This protocol incorporates Apple Authentication Coprocessor signatures to ensure the accessory has completed MFi certification.

Note: The C functions referenced in this section refer to the SRP C API defined by `http://srp.stanford.edu`. If you are using a different SRP implementation then you'll need to use the equivalent functions in your library.

5.6.1 M1: iOS Device -> Accessory – 'SRP Start Request'

When the iOS device performs authentication as part of Pair Setup it sends a request to the accessory with the following TLV items:

```
kTLVType_State   <M1>
kTLVType_Method  <Pair Setup with Authentication>
```

When the iOS device performs Pair Setup with a seperate optional authentication procedure, it sends a request to the accessory with the following TLV items:

```
kTLVType_State   <M1>
kTLVType_Method  <Pair Setup>
kTLVType_Flags   <Pairing Type Flags> Table 5-18 (page 65)
```

5.6.2 M2: Accessory -> iOS Device – 'SRP Start Response'

When the accessory receives <M1>, it must perform the following steps:

1. If the accessory is already paired, it must respond with the following TLV items:

   ```
   kTLVType_State   <M2>
   kTLVType_Error   <kTLVError_Unavailable>
   ```

2. If the accessory has received more than 100 unsuccessful authentication attempts, it must respond with the following TLV items:

   ```
   kTLVType_State   <M2>
   kTLVType_Error   <kTLVError_MaxTries>
   ```

3. If the accessory is currently performing a Pair Setup operation with a different controller, it must respond with the following TLV items:

   ```
   kTLVType_State   <M2>
   kTLVType_Error   <kTLVError_Busy>
   ```

4. Create new SRP session with `SRP_new( SRP6a_server_method() )`.

5. Set SRP username to `Pair-Setup` with `SRP_set_username()`.

6. Generate 16 bytes of random salt and set it with `SRP_set_params()`.

7. If the accessory received the M1 (SRP Start Request) without `kTLVType_Flags` or if the `kTLVType_Flags` were set as kPairingFlag_Transient and kPairingFlag_Split then

   - If the accessory can display a random setup code then it must generate a random setup code save the SRP verifier for the Setup Code and use it for the next Pair-Setup with kPairingFlag_Split and set it with `SRP_set_auth_password()`.
   - If the accessory cannot display a random setup code then it must retrieve the SRP verifier for the Setup Code, e.g. from an EEPROM, and set the verifier with `SRP_set_authenticator()`.
   - The accessory must include the received `kTLVType_Flags` in its M2 response.

   If the accessory received the M1 (SRP Start Request) with the `kTLVType_Flags` set as kPairingFlag_Split then

   - If the accessory has saved SRP verifier it must retrieve the saved SRP verifier for the Setup Code, e.g. from an EEPROM, and set the verifier with `SRP_set_authenticator()`. The accessory must also include the received `kTLVType_Flags` in its M2 response.
   - If the accessory does not have a saved SRP verifier it mush respond with the following TLV items:

     ```
     kTLVType_State   <M2>
     kTLVType_Error   <kTLVError_Authentication>
     ```

   The Setup Code must conform to the format XXX-XX-XXX where each X is a 0-9 digit and dashes are required. See Section "4.2.1 Setup Code" (page 32) for more information.

8. If the accessory has generated a setup code then it must present the Setup Code to the user, e.g. display it on the accessory's screen. If the accessory doesn't have a screen then the Setup Code may be on a printed label. If using software token authentication to enable HAP on an accessory through firmware update the setup code must be presented in the iOS App.

9. Generate an SRP public key with SRP_gen_pub().

10. Respond to the iOS device's request with the following TLV items:

kTLVType_State	<M2>
kTLVType_PublicKey	<Accessory's SRP public key>
kTLVType_Salt	<16 byte salt generated in Step 6>
kTLVType_Flags	<Pairing Type Flags> (Optional as per Step 7)

5.6.3 M3: iOS Device -> Accessory – 'SRP Verify Request'

When the iOS device receives <M2>, it will check for kTLVType_Error. If present, the iOS device will abort the setup process and report the error to the user. If kTLVType_Error is not present and the controller is performing only a Split Pair Setup (i.e. kPairingFlag_Split was set and kPairingFlag_Transient was not set in M2) then the coltroller will reuse the setup code that was used in the previous Transient + Split Setup session, otherwise the user is prompted to enter the Setup Code provided by the accessory. Once the Setup Code is avaliable, the iOS device performs the following steps:

1. Create a new SRP session with SRP_new(SRP6a_client_method()).

2. Set the SRP username to Pair-Setup with SRP_set_username().

3. Set salt provided by the accessory in the <M2> TLV with SRP_set_params().

4. Generate its SRP public key with SRP_gen_pub().

5. Set the Setup Code as entered by the user with SRP_set_auth_password().

6. Compute the SRP shared secret key with SRP_compute_key().

7. Generate iOS device-side SRP proof with SRP_respond().

8. Send a request to the accessory with the following TLV items:

kTLVType_State	<M3>
kTLVType_PublicKey	<iOS device's SRP public key>
kTLVType_Proof	<iOS device's SRP proof>

5.6.4 M4: Accessory -> iOS Device – 'SRP Verify Response'

When the accessory receives <M3>, it must perform the following steps:

1. Use the iOS device's SRP public key to compute the SRP shared secret key with SRP_compute_key().

2. Verify the iOS device's SRP proof with SRP_verify(). If verification fails, the accessory must respond with the following TLV items:

kTLVType_State	<M4>
kTLVType_Error	kTLVError_Authentication

3. Generate the accessory-side SRP proof with SRP_respond().

4. If the accessory is performing Pair Setup with Authentication (i.e. kTLVType_Method is <Pair Setup with Authentication>)

(a) Generate the MFi challenge, `MFiChallenge`, from the SRP shared secret by using HKDF−SHA−512 with the following parameters:

```
InputKey    = <SRP shared secret>
Salt        = "MFi-Pair-Setup-Salt"
Info        = "MFi-Pair-Setup-Info"
OutputSize  = 32 bytes
```

(b) Derive the MFi Challenge Data, `MFiChallengeData`, by SHA-256 (for Apple Authentication Coprocessor 3.0) or SHA-1 (for Apple Authentication Coprocessor 2.0C) hashing the `MFiChallenge`.

(c) Generate the MFi proof, `MFiProof`, by performing challenge-response generation with the Apple Authentication Coprocessor. A value of `20` (for Apple Authentication Coprocessor 2.0C only) must be written to the "Challenge Data Length" register of the Apple Authentication Coprocessor and the bytes of `MFiChallengeData` must be written to the "Challenge Data" register of the Apple Authentication Coprocessor.

Once the accessory has performed challenge-response generation, the bytes contained in the "Challenge Response Data" register must be provided as the `MFiProof`.

(d) Read the Accessory Certificate, `AccessoryCertificate`, from the Apple Authentication Coprocessor. The data will be the accessory's X.509 certificate in a DER-encoded, PKCS#7 message.

(e) Construct a sub-TLV with the following TLV items:

```
kTLVType_Signature     <MFiProof>
kTLVType_Certificate   <AccessoryCertificate>
```

(f) Derive the symmetric session encryption key, `SessionKey`, from the SRP shared secret by using HKDF−SHA−512 with the following parameters:

```
InputKey    = <SRP shared secret>
Salt        = "Pair-Setup-Encrypt-Salt"
Info        = "Pair-Setup-Encrypt-Info"
OutputSize  = 32 bytes
```

(g) Encrypt the sub-TLV, `encryptedData`, and generate the 16 byte auth tag, `authTag`. This uses the ChaCha20-Poly1305 AEAD algorithm with the following parameters:

```
encryptedData, authTag = ChaCha20-Poly1305(SessionKey, Nonce="PS-Msg04",
AAD=<none>, Msg=<Sub-TLV>)
```

If the accessory is performing a Transient Pair Setup (i.e. `kTLVType_Method` is `<Pair Setup>` and the kPairingFlag_Transient is set in `kTLVType_Flags`)

(a) Derive the Pair-Setup session encryption keys, `Pair Setup SessionKeys` for all subsequent messages, from the SRP shared secret by using HKDF−SHA−512

Messages from the Controller to Accessory shall be encrypted with ControllerEncryptKey =

HKDF-SHA-512 of

```
InputKey    = <SRP shared secret>
Salt        = "SplitSetupSalt"
Info        = "ControllerEncrypt-Control"
OutputSize  = 32 bytes
```

Messages from Accessory to Controller shall be encrypted with AccessoryEncryptKey =

HKDF-SHA-512 of

```
InputKey    = <SRP shared secret>
Salt        = "SplitSetupSalt"
Info        = "AccessoryEncrypt-Control"
OutputSize  = 32 bytes
```

All subsequent messages must be secured with the AEAD algorithm AEAD_CHACHA20_POLY1305 as specified in Section 2.8 of RFC 7539. The accessory and controller shall use seperate nonces and shall be reset to 0. The 32-bit fixed-common part of the 96-bit nonce is all zeros: `00 00 00 00`.

If the accessory is performing only a Split Pair Setup (i.e. kTLVType_Method is <Pair Setup> and the kPairingFlag_Split is set, and kPairingFlag_Transient is not set in kTLVType_Flags)

(a) Derive the symmetric session encryption key, SessionKey, from the SRP shared secret by using HKDF-SHA-512 with the following parameters:

```
InputKey    = <SRP shared secret>
Salt        = "Pair-Setup-Encrypt-Salt"
Info        = "Pair-Setup-Encrypt-Info"
OutputSize  = 32 bytes
```

5. Construct the response with the following TLV items:

```
kTLVType_State          <M4>
kTLVType_Proof          <Accessory's SRP proof>
kTLVType_EncryptedData  <encryptedData with authTag appended> (Optional as per Step 4)
```

6. Send the response to the iOS device.

7. If the accessory is performing a transient Pair Setup (i.e. kTLVType_Method is <Pair Setup> and the kPairingFlag_Transient is set in kTLVType_Flags) then Pair Setup is complete for the accessory and the accessory must enable session security with the Pair-Setup session keys generated in Step 4.

5.6.5 M5: iOS Device -> Accessory – 'Exchange Request'

5.6.5.1 <M4> Verification

When the iOS device receives <M4>, it performs the following steps:

1. Check for kTLVType_Error. If present and it's set to kTLVError_Authentication, the user will be prompted that the Setup Code was incorrect and be allowed to try again. If kTLVType_Error is set to any other error code, then the setup process will be aborted and an error will be reported to the user. The accessory resets to <M1> for Pair Setup.

2. Verify accessory's SRP proof with SRP_verify(). If this fails, the setup process will be aborted and an error will be reported to the user.

3. If the accessory is performing Pair Setup with Authentication (i.e. kTLVType_Method is <Pair Setup with Authentication>)

   (a) Derive the symmetric session encryption key, SessionKey, from the SRP shared secret using the same method as the accessory.

   (b) Verify the sub-TLV's authTag from encryptedData. If this fails, the setup process will be aborted and an error will be reported to the user.

   (c) Decrypt the sub-TLV in encryptedData. If this fails, the setup process will be aborted and an error will be reported to the user.

   (d) Verify the AccessoryCertificate contained in the sub-TLV. The AccessoryCertificate must be issued by the MFi Sub-Certificate Authority, chained to the Apple root certificate, and have a class between 0x00 and 0x0A.

   (e) Derive the MFiChallenge from the SRP shared secret using the same method as the accessory.

(f) Verify the `MFiProof` using the public key contained in the `AccessoryCertificate`. If this fails, the setup process will be aborted and an error will be reported to the user.

If the accessory is performing a transient Pair Setup (i.e. `kTLVType_Method` is <`Pair Setup`> and the kPairingFlag_Transient is set in `kTLVType_Flags`)

(a) Derive the Pair-Setup session encryption keys, `Pair Setup SessionKeys` for subsequent messages, from the SRP shared secret by using HKDF−SHA−512

Messages from the Controller to Accessory shall be encrypted with ControllerEncryptKey =

HKDF-SHA-512 of	`InputKey`	= <`SRP shared secret`>
	`Salt`	= `"SplitSetupSalt"`
	`Info`	= `"ControllerEncrypt−Control"`
	`OutputSize`	= `32 bytes`

Messages from Accessory to Controller shall be encrypted with AccessoryEncryptKey =

HKDF-SHA-512 of	`InputKey`	= <`SRP shared secret`>
	`Salt`	= `"SplitSetupSalt"`
	`Info`	= `"AccessoryEncrypt−Control"`
	`OutputSize`	= `32 bytes`

All subsequent messages must be secured with the AEAD algorithm AEAD_CHACHA20_POLY1305 as specified in Section 2.8 of RFC 7539. The accessory and controller shall use seperate nonces and shall be reset to 0. The 32-bit fixed-common part of the 96-bit nonce is all zeros: `00 00 00 00`.

(b) Pair Setup is now complete on the controller and the controller shall enable session security with the Pair-Setup session keys.

5.6.5.2 <M5> Request Generation

Once <M4> Verification is complete, and the controller is performing a non-transient pair-setup the iOS device performs the following steps to generate the <M5> request:

1. Generate its Ed25519 long-term public key, `iOSDeviceLTPK`, and long-term secret key, `iOSDeviceLTSK`, if they don't exist.

2. Derive `iOSDeviceX` from the SRP shared secret by using HKDF−SHA−512 with the following parameters:

```
InputKey    = <SRP shared secret>
Salt        = "Pair−Setup−Controller−Sign−Salt"
Info        = "Pair−Setup−Controller−Sign−Info"
OutputSize  = 32 bytes
```

3. Concatenate `iOSDeviceX` with the iOS device's Pairing Identifier, `iOSDevicePairingID`, and its long-term public key, `iOSDeviceLTPK`. The data must be concatenated in order such that the final data is `iOSDeviceX, iOSDevicePairingID, iOSDeviceLTPK`. The concatenated value will be referred to as `iOSDeviceInfo`.

4. Generate `iOSDeviceSignature` by signing `iOSDeviceInfo` with its long-term secret key, `iOSDeviceLTSK`, using Ed25519.

5. Construct a sub-TLV with the following TLV items:

`kTLVType_Identifier`	<`iOSDevicePairingID`>
`kTLVType_PublicKey`	<`iOSDeviceLTPK`>
`kTLVType_Signature`	<`iOSDeviceSignature`>

6. Encrypt the sub-TLV, `encryptedData`, and generate the 16 byte auth tag, `authTag`. This uses the ChaCha20-Poly1305 AEAD algorithm with the following parameters:
   `encryptedData, authTag = ChaCha20-Poly1305(SessionKey, Nonce="PS-Msg05", AAD=<none>, Msg=<Sub-TLV>)`

7. Send the request to the accessory with the following TLV items:
```
kTLVType_State          <M5>
kTLVType_EncryptedData  <encryptedData with authTag appended>
```

5.6.6 M6: Accessory -> iOS Device – 'Exchange Response'

5.6.6.1 <M5> Verification

When the accessory receives <M5>, it must perform the following steps:

1. Verify the iOS device's `authTag`, which is appended to the `encryptedData` and contained within the `kTLVType_EncryptedData` TLV item, from `encryptedData`. If verification fails, the accessory must respond with the following TLV items:
```
kTLVType_State   <M6>
kTLVType_Error   kTLVError_Authentication
```

2. Decrypt the sub-TLV in `encryptedData`. If decryption fails, the accessory must respond with the following TLV items:
```
kTLVType_State   <M6>
kTLVType_Error   kTLVError_Authentication
```

3. Derive `iOSDeviceX` from the SRP shared secret by using HKDF-SHA-512 with the following parameters:
```
InputKey     = <SRP shared secret>
Salt         = "Pair-Setup-Controller-Sign-Salt"
Info         = "Pair-Setup-Controller-Sign-Info"
OutputSize   = 32 bytes
```

4. Construct `iOSDeviceInfo` by concatenating `iOSDeviceX` with the iOS device's Pairing Identifier, `iOSDevicePairingID`, from the decrypted sub-TLV and the iOS device's long-term public key, `iOSDeviceLTPK` from the decrypted sub-TLV. The data must be concatenated in order such that the final data is `iOSDeviceX, iOSDevicePairingID, iOSDeviceLTPK`.

5. Use Ed25519 to verify the signature of the constructed `iOSDeviceInfo` with the `iOSDeviceLTPK` from the decrypted sub-TLV. If signature verification fails, the accessory must respond with the following TLV items:
```
kTLVType_State   <M6>
kTLVType_Error   kTLVError_Authentication
```

6. Persistently save the `iOSDevicePairingID` and `iOSDeviceLTPK` as a pairing. If the accessory cannot accept any additional pairings, it must respond with the following TLV items:
```
kTLVType_State   <M6>
kTLVType_Error   kTLVError_MaxPeers
```

5.6.6.2 <M6> Response Generation

Once <M5> Verification is complete, the accessory must perform the following steps to generate the <M6> response:

1. Generate its Ed25519 long-term public key, `AccessoryLTPK`, and long-term secret key, `AccessoryLTSK`, if they don't exist.

2. Derive `AccessoryX` from the SRP shared secret by using `HKDF-SHA-512` with the following parameters:

   ```
   InputKey     = <SRP shared secret>
   Salt         = "Pair-Setup-Accessory-Sign-Salt"
   Info         = "Pair-Setup-Accessory-Sign-Info"
   OutputSize   = 32 bytes
   ```

3. Concatenate `AccessoryX` with the accessory's Pairing Identifier, `AccessoryPairingID`, and its long-term public key, `AccessoryLTPK`. The data must be concatenated in order such that the final data is `AccessoryX, AccessoryPairingID, AccessoryLTPK`. The concatenated value will be referred to as `AccessoryInfo`.

4. Use Ed25519 to generate `AccessorySignature` by signing `AccessoryInfo` with its long-term secret key, `AccessoryLTSK`.

5. Construct the sub-TLV with the following TLV items:

`kTLVType_Identifier`	`<AccessoryPairingID>`
`kTLVType_PublicKey`	`<AccessoryLTPK>`
`kTLVType_Signature`	`<AccessorySignature>`

6. Encrypt the sub-TLV, `encryptedData`, and generate the 16 byte auth tag, `authTag`. This uses the ChaCha20-Poly1305 AEAD algorithm with the following parameters:
   `encryptedData, authTag = ChaCha20-Poly1305(SessionKey, Nonce="PS-Msg06", AAD=<none>, Msg=<Sub-TLV>)`

7. Send the response to the iOS device with the following TLV items:

   ```
   kTLVType_State           <M6>
   kTLVType_EncryptedData   <encryptedData with authTag appended>
   ```

5.6.6.3 <M6> Verification by iOS Device

When the iOS device receives <M6>, it performs the following steps:

1. Verifies `authTag`, which is appended to the `encryptedData` and contained within the `kTLVType_EncryptedData` TLV item, from `encryptedData`. If this fails, the setup process will be aborted and an error will be reported to the user.

2. Decrypts the sub-TLV in `encryptedData`. If this fails, the setup process will be aborted and an error will be reported to the user.

3. Uses Ed25519 to verify the signature of `AccessoryInfo` using `AccessoryLTPK`. If this fails, the setup process will be aborted and an error will be reported to the user.

4. Persistently saves `AccessoryPairingID` and `AccessoryLTPK` as a pairing.

Pair Setup is now complete.

5.7 Pair Verify

Once a pairing has been established, it must be verified each time it is used. Pair Verify uses the Station-to-Station protocol to perform bidirectional authentication and results in a mutually authenticated shared secret for future session security.

All Pairing Identifiers in pair verify are case-sensitive.

The following describes the flow of messages to verify pairing.

5.7.1 M1: iOS Device -> Accessory – 'Verify Start Request'

The iOS device generates a new, random Curve25519 key pair and sends a request to the accessory with the following TLV items:

```
kTLVType_State      <M1>
kTLVType_PublicKey  <iOS device's Curve25519 public key>
```

5.7.2 M2: Accessory -> iOS Device – 'Verify Start Response'

When the accessory receives <M1>, it must perform the following steps:

1. Generate new, random Curve25519 key pair.

2. Generate the shared secret, SharedSecret, from its Curve25519 secret key and the iOS device's Curve25519 public key.

3. Construct AccessoryInfo by concatenating the following items in order:

   (a) Accessory's Curve25519 public key.

   (b) Accessory's Pairing Identifier, AccessoryPairingID.

   (c) iOS device's Curve25519 public key from the received <M1> TLV.

4. Use Ed25519 to generate AccessorySignature by signing AccessoryInfo with its long-term secret key, AccessoryLTSK.

5. Construct a sub-TLV with the following items:

   ```
   kTLVType_Identifier   <AccessoryPairingID>
   kTLVType_Signature    <AccessorySignature>
   ```

6. Derive the symmetric session encryption key, SessionKey, from the Curve25519 shared secret by using HKDF-SHA-512 with the following parameters:

   ```
   InputKey     = <Curve25519 shared secret>
   Salt         = "Pair-Verify-Encrypt-Salt"
   Info         = "Pair-Verify-Encrypt-Info"
   OutputSize   = 32 bytes
   ```

7. Encrypt the sub-TLV, encryptedData, and generate the 16-byte auth tag, authTag. This uses the ChaCha20-Poly1305 AEAD algorithm with the following parameters:

   ```
   encryptedData, authTag = ChaCha20-Poly1305(SessionKey, Nonce="PV-Msg02",
   AAD=<none>, Msg=<Sub-TLV>)
   ```

8. Construct the response with the following TLV items:
```
kTLVType_State           <M2>
kTLVType_PublicKey       <Accessory's Curve25519 public key>
kTLVType_EncryptedData   <encryptedData with authTag appended>
```

9. Send the response to the iOS device.

5.7.3 M3: iOS Device -> Accessory – 'Verify Finish Request'

When the iOS device receives <M2>, it performs the following steps:

1. Generate the shared secret, `SharedSecret`, from its Curve25519 secret key and the accessory's Curve25519 public key.

2. Derive the symmetric session encryption key, `SessionKey`, in the same manner as the accessory.

3. Verify the 16-byte auth tag, `authTag`, against the received `encryptedData`. If this fails, the setup process will be aborted and an error will be reported to the user.

4. Decrypt the sub-TLV from the received `encryptedData`.

5. Use the accessory's Pairing Identifier to look up the accessory's long-term public key, `AccessoryLTPK`, in its list of paired accessories. If not found, the setup process will be aborted and an error will be reported to the user.

6. Use Ed25519 to verify `AccessorySignature` using `AccessoryLTPK` against `AccessoryInfo`. If this fails, the setup process will be aborted and an error will be reported to the user.

7. Construct `iOSDeviceInfo` by concatenating the following items in order:

   (a) iOS Device's Curve25519 public key.

   (b) iOS Device's Pairing Identifier, `iOSDevicePairingID`.

   (c) Accessory's Curve25519 public key from the received <M2> TLV.

8. Use Ed25519 to generate `iOSDeviceSignature` by signing `iOSDeviceInfo` with its long-term secret key, `iOSDeviceLTSK`.

9. Construct a sub-TLV with the following items:
```
kTLVType_Identifier   <iOSDevicePairingID>
kTLVType_Signature    <iOSDeviceSignature>
```

10. Encrypt the sub-TLV, `encryptedData`, and generate the 16-byte auth tag, `authTag`. This uses the ChaCha20-Poly1305 AEAD algorithm with the following parameters:
```
encryptedData, authTag = ChaCha20-Poly1305(SessionKey, Nonce="PV-Msg03",
AAD=<none>, Msg=<Sub-TLV>)
```

11. Construct the request with the following TLV items:
```
kTLVType_State           <M3>
kTLVType_EncryptedData   <encryptedData with authTag appended>
```

12. Send the request to the accessory.

5.7.4 M4: Accessory -> iOS Device – 'Verify Finish Response'

When the accessory receives <M3>, it must perform the following steps:

1. Verify the iOS device's authTag, which is appended to the encryptedData and contained within the kTLVType_EncryptedData TLV item, against encryptedData. If verification fails, the accessory must respond with the following TLV items:

   ```
   kTLVType_State   <M4>
   kTLVType_Error   kTLVError_Authentication
   ```

2. Decrypt the sub-TLV in encryptedData. If decryption fails, the accessory must respond with the following TLV items:

   ```
   kTLVType_State   <M4>
   kTLVType_Error   kTLVError_Authentication
   ```

3. Use the iOS device's Pairing Identifier, iOSDevicePairingID, to look up the iOS device's long-term public key, iOSDeviceLTPK, in its list of paired controllers. If not found, the accessory must respond with the following TLV items:

   ```
   kTLVType_State   <M4>
   kTLVType_Error   kTLVError_Authentication
   ```

4. Use Ed25519 to verify iOSDeviceSignature using iOSDeviceLTPK against iOSDeviceInfo contained in the decrypted sub-TLV. If decryption fails, the accessory must respond with the following TLV items:

   ```
   kTLVType_State   <M4>
   kTLVType_Error   kTLVError_Authentication
   ```

5. Send the response to the iOS device with the following TLV items: kTLVType_State <M4>

When the iOS device receives <M4>, the Pair Verify process is complete. If a subsequent Pair Verify request from another controller occurs in the middle of a Pair Verify transaction the accessory must honor both Pair Verify requests and maintain separate secure sessions for each controller. If a subsequent Pair Verify request from the same controller occurs in the middle of the Pair Verify process then the accessory must immediately tear down the existing session with the controller and must accept the newest request.

5.8 Fragmentation and Reassembly

Some transports, like Bluetooth LE, have limits on the maximum amount of data that can be exchanged per transaction. To support these transports, pairing requests and responses may need to be fragmented when sent and reassembled when received. All fragments except the last must use the kTLVType_FragmentData TLV type. The last fragment must use the kTLVType_FragmentLast TLV type.

The "value" of these TLV items is the next portion of data from the original pairing TLV that will fit in the current MTU. When a non-last fragment is received, it is appended to the reassembly buffer and a kTLVType_FragmentData item is sent back with an empty "value" section. This acknowledges the fragment so the next fragment can be sent.

When the last fragment is received, the reassembled TLV is processed normally. Fragments must be sent and received in order. Fragmentation may occur on a per-request/response basis. Requests or responses that fit within the MTU of the transport must be sent as-is and not fragmented.

5.9 AEAD Algorithm

When data is encrypted or decrypted, it uses the ChaCha20-Poly1305 AEAD algorithm as defined in RFC 7539. This document uses the 64-bit nonce option where the first 32 bits of 96-bit nonce are 0.

5.10 Add Pairing

This is used to exchange long-term public keys to establish a pairing relationship for an additional controller. Add Pairing can only be performed by admin controllers, that have established a secure session with the accessory using the "5.7 Pair Verify" (page 49) procedure. Authenticated encryption is used for all encrypted data.

The minimum number of pairing relationships that an accessory must support is **16**.

5.10.1 M1: iOS Device -> Accessory – 'Add Pairing Request'

The iOS device performs the following steps:

1. Get the Pairing Identifier, `AdditionalControllerPairingIdentifier`, and the Ed25519 long-term public key of the additional controller to pair, `AdditionalControllerLTPK`, via an out-of-band mechanism.

2. Construct the request TLV with the following items:
   ```
   kTLVType_State          <M1>
   kTLVType_Method         <Add Pairing>
   kTLVType_Identifier     AdditionalControllerPairingIdentifier
   kTLVType_PublicKey      AdditionalControllerLTPK
   kTLVType_Permissions    AdditionalControllerPermissions
   ```

3. Send the TLV over the HAP session established via "5.7 Pair Verify" (page 49), which provides bidirectional, authenticated encryption.

5.10.2 M2: Accessory -> iOS Device – 'Add Pairing Response'

When the accessory receives the request, it must perform the following steps:

1. Validate the received data against the established HAP session as described in the transport-specific chapters.

2. Verify that the controller sending the request has the admin bit set in the local pairings list. If not, accessory must abort and respond with the following TLV items:
   ```
   kTLVType_State    <M2>
   kTLVType_Error    kTLVError_Authentication
   ```

3. If a pairing for `AdditionalControllerPairingIdentifier` exists, it must perform the following steps:

   (a) If the `AdditionalControllerLTPK` does not match the stored long-term public key for `AdditionalControllerPairingIdentifier`, respond with the following TLV items:
   ```
   kTLVType_State    <M2>
   kTLVType_Error    kTLVError_Unknown
   ```

   (b) Update the permissions of the controller to match `AdditionalControllerPermissions`.

4. Otherwise, if a pairing for `AdditionalControllerPairingIdentifier` does not exist, it must perform the following steps:

   (a) Check if the accessory has space to support an additional pairing; the minimum number of supported pairings is 16 pairings. If not, accessory must abort and respond with the following TLV items:

   ```
   kTLVType_State   <M2>
   kTLVType_Error   kTLVError_MaxPeers
   ```

   (b) Save the additional controller's `AdditionalControllerPairingIdentifier`, `AdditionalControllerLTPK` and `AdditionalControllerPermissions` to a persistent store. If an error occurs while saving, accessory must abort and respond with the following TLV items:

   ```
   kTLVType_State   <M2>
   kTLVType_Error   kTLVError_Unknown
   ```

5. Construct a response with the following TLV items:

   ```
   kTLVType_State   <M2>
   ```

6. Send the response over the HAP session established via "5.7 Pair Verify" (page 49), which provides bidirectional, authenticated encryption.

When the iOS device receives this response, it performs the following steps:

1. Validate the received data against the established HAP session.

2. Validate that the received TLV contains no errors.

3. Send the accessory's long-term public key and Pairing Identifier to the additional controller via an out-of-band mechanism.

5.11 Remove Pairing

This is used to remove a previously established pairing. Remove Pairing can only be performed by admin controllers, that have established a secure session with the accessory using the "5.7 Pair Verify" (page 49) procedure. Authenticated encryption is used for all encrypted data.

After a remove pairing is completed, the accessory must tear down any existing connections with the removed controller within 5 seconds. Until those existing connections are closed, the accessory must refuse any requests, by returning the appropriate transport specific HAP Status codes in the response:

1. IP: -70401 (Table 6-11 (page 81))

2. BLE: 0x03 (Table 7-37 (page 123))

3. HAP over iCloud: 101

If the last remaining admin controller pairing is removed, all pairings on the accessory must be removed.

5.11.1 M1: iOS Device -> Accessory – 'Remove Pairing Request'

The iOS device performs the following steps:

1. Get the Pairing Identifier of the additional controller to remove, `RemovedControllerPairingIdentifier`, via an out-of-band mechanism.

2. Construct the request TLV with the following items:
   ```
   kTLVType_State        <M1>
   kTLVType_Method       <Remove Pairing>
   kTLVType_Identifier   RemovedControllerPairingIdentifier
   ```

3. Send the TLV over the HAP session established via "5.7 Pair Verify" (page 49), which provides bidirectional, authenticated encryption.

5.11.2 M2: Accessory -> iOS Device – 'Remove Pairing Response'

When the accessory receives the request, it must perform the following steps:

1. Validate the received data against the established HAP session as described in the transport-specific chapters.

2. Verify that the controller sending the request has the admin bit set in the local pairings list. If not, accessory must abort and respond with the following TLV items:
   ```
   kTLVType_State   <M2>
   kTLVType_Error   kTLVError_Authentication
   ```

3. If the pairing exists, remove `RemovedControllerPairingIdentifier` and its corresponding long-term public key from persistent storage. If a pairing for `RemovedControllerPairingIdentifier` does not exist, the accessory must return success. Otherwise, if an error occurs during removal, accessory must abort and respond with the following TLV items:
   ```
   kTLVType_State   <M2>
   kTLVType_Error   kTLVError_Unknown
   ```

4. Construct a response with the following TLV items: `kTLVType_State    <M2>`

5. Send the response over the HAP session established via "5.7 Pair Verify" (page 49), which provides bidirectional, authenticated encryption.

6. If the controller requested the accessory to remove its own pairing the accessory must invalidate the HAP session immediately after the response is sent.

7. If there are any established HAP sessions with the controller that was removed, then these connections must be immediately torn down and any associated data stream (e.g RTP, HDS) must be stopped and removed.

5.12 List Pairings

This is used to read a list of all the currently established pairings. List Pairings can only be performed by controllers that have been through "5.6 Pair Setup" (page 41) with the accessory and have established a shared secret via "5.7 Pair Verify" (page 49). Authenticated encryption is used for all encrypted data.

This is used to read a list of all the currently established pairings. It's a read with a response over an HAP session established via "5.7 Pair Verify" (page 49), which provides bidirectional, authenticated encryption. The response is a group of TLV items with separator items to delineate them. Each pairing entry must be comprised of the following TLV items:

```
kTLVType_Identifier    <Pairing Identifier of Controller 1>
kTLVType_PublicKey     <Ed25519 long-term public key of Controller 1>
kTLVType_Permissions   <Bit value describing permissions of Controller 1>
```

5.12.1 M1: iOS Device -> Accessory – 'List Pairings Request'

The iOS device performs the following steps:

1. Construct the request TLV with the following items:

   ```
   kTLVType_State    <M1>
   kTLVType_Method   <List Pairings>
   ```

2. Send the TLV over the HAP session established via "5.7 Pair Verify" (page 49), which provides bidirectional, authenticated encryption.

5.12.2 M2: Accessory -> iOS Device – 'List Pairings Response'

When the accessory receives the request, it must perform the following steps:

1. Validate the received data against the established HAP session as described in the transport-specific chapters.

2. Verify that the controller sending the request has the admin bit set in the local pairings list. If not, abort and respond with the following TLV items:

   ```
   kTLVType_State   <M2>
   kTLVType_Error   kTLVError_Authentication
   ```

3. Construct a response with the following TLV items:

   ```
   kTLVType_State         <M2>
   kTLVType_Identifier    <Pairing Identifier of Controller 1>
   kTLVType_PublicKey     <Ed25519 long-term public key of Controller 1>
   kTLVType_Permissions   <Bit value describing permissions of Controller 1>
   ```

   If another pairing follows a pairing then it must be separated using a separator item:

   ```
   kTLVType_Separator   <No value>
   ```

   Additional pairings must contain the following TLV items:

   ```
   kTLVType_Identifier    <Pairing Identifier of Controller N>
   kTLVType_PublicKey     <Ed25519 long-term public key of Controller N>
   kTLVType_Permissions   <Bit value describing permissions of Controller N>
   ```

4. Send the response over the HAP session established via "5.7 Pair Verify" (page 49), which provides bidirectional, authenticated encryption.

5.13 Pairing over Bluetooth LE

When pairing over Bluetooth LE, the accessory advertises a pairing service. Pairing requests are performed by writing to characteristics of this service. Pairing responses are performed by subsequently reading characteristics of this service. The read response must be sent within 30 seconds of the read request. The maximum payload size for an HAP Characteristic value over HAP Bluetooth LE must not exceed 512 bytes "7.4.1.7 Maximum Payload Size" (page 129). Pairing payloads with certificates will exceed 512 bytes and require payload fragmentation to be delivered, therefore accessories must support fragmentation per "5.8 Fragmentation and Reassembly" (page 51).

5.13.1 Pairing Service

Defines characteristics to support pairing between a controller and an accessory.

Property	Value
UUID	00000055-0000-1000-8000-0026BB765291
Type	public.hap.service.pairing
Required Characteristics	"5.13.1.1 Pair Setup" (page 56) "5.13.1.2 Pair Verify" (page 56) "5.13.1.3 Pairing Features" (page 57) "5.13.1.4 Pairing Pairings" (page 57)

5.13.1.1 Pair Setup

Accessories must accept reads and writes to this characteristic to perform Pair Setup.

Property	Value
UUID	0000004C-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.pairing.pair-setup
Permissions	Read, Write
Format	tlv8

5.13.1.2 Pair Verify

Accessories must accept reads and writes to this characteristic to perform Pair Verify.

Property	Value
UUID	0000004E-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.pairing.pair-verify
Permissions	Read, Write
Format	tlv8

5.13.1.3 Pairing Features

Read-only characteristic that exposes pairing features must be supported by the accessory. See Table 5-15 (page 63).

Property	Value
UUID	0000004F-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.pairing.features
Permissions	Read
Format	uint8

5.13.1.4 Pairing Pairings

Accessories must accept reads and writes to this characteristic to add, remove, and list pairings.

Property	Value
UUID	00000050-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.pairing.pairings
Permissions	Paired Read, Paired Write
Format	tlv8

5.14 Pairing over IP

When pairing over an IP network, HTTP is used as the transport. Pairing requests are performed by sending a POST request to the accessory's HTTP server unless otherwise noted. The URL resource specifies the pairing operation to perform. The body of the HTTP request contains the TLV data. Pairing responses are delivered as TLV data in the body of the HTTP response. The MIME type for requests and responses is `application/pairing+tlv8`. Discovery of the HTTP server's IP address, port, and supported features is required to be supported via Bonjour, please see the "6 HomeKit Accessory Protocol for IP Accessories" (page 66).

Table 5-1: HTTP Status Codes

Status Code	**Description**
200 OK	Success.
400 Bad Request	Generic error for a problem with the request, e.g. bad TLV, state error, etc.
405 Method Not Allowed	Wrong HTTP request method, e.g. GET when expecting POST.
429 Too Many Requests	Server cannot handle any more requests of this type, e.g. attempt to pair while already pairing.
470 Connection Authorization Required	Request to secure resource made without establishing security, e.g. didn't perform Pair Verify.
500 Internal Server Error	Server had a problem, e.g. ran out of memory.

Table 5-2: HTTP URLs

Status Code	**Description**
/pair-setup	Used for Pair Setup.
/pair-verify	Used for Pair Verify.
/pairings	Used for adding, removing, and listing pairings. Always sends HTTP POST with TLV8 payloads defined in "5.10 Add Pairing" (page 52), "5.11 Remove Pairing" (page 53), and "5.12 List Pairings" (page 54).
/secure-message	Used to exchange HAP PDU messages after a secure session is established. "5.15 Software Authentication Procedure" (page 58)

5.15 Software Authentication Procedure

Software authentication must be performed after a secure session (i.e either a Pair-Setup secure session or after a Pair-Verify secure session) is established. These procedures use the HAP-PDU formats as defined in "7.3.3 HAP PDU Format" (page 102).

Each request / response message defined in this section is independent of one another and shall be performed in any order and at any point in time as long as there is a secure session. The accessory must accept and respond to these requests whenever requested by the controller over a secure session. For IP accessories, HTTP is used as the transport and the authentication procedures are performed by sending a POST request to the accessory's HTTP server. The URL /secure-message is used for the authentication procedures. The body of the HTTP request contains the HAP-PDU. Responses are delivered as TLV data in the body of the HTTP response. The MIME type for requests and responses is `application/octet-stream`.

For IP accessories instance Id in the request shall be set to 0, for BLE accessories the instance id shall be Instance Id of the Service Signature Charactersitic "7.4.3.2 Service Signature Characteristic" (page 134) in the Protocol Information Service "7.4.3 HAP-BLE Protocol Information Service" (page 134).

Table 5-3: HAP Software Authentication Opcode Description

HAP Opcode	Description
0x10	HAP-Token-Request / Response
0x11	HAP-Token-Update-Request / Response
0x12	HAP-Info-Request / Response

Table 5-4: Software Authentication TLV types

TLV Type	Description
0x01	UUID (The matching UUID for the initial token provisioned on the accessory)
0x02	Software Token

5.15.1 HAP-Token-Request

Table 5-5: HAP-Token-Request

PDU Header			
Control Field (1 Byte)	HAP Opcode (1 Byte)	TID (1 Byte)	CharID (2 Bytes)
0b 0000 0000	0x10	0xXX	0xXX 0xXX

5.15.2 HAP-Token-Response

Table 5-6: HAP-Token-Response

PDU Header			PDU Body		
Control Field (1 Byte)	TID (1 Byte)	Status (1 Byte)	Body length (2 Bytes)	UUID TLV	Software Token TLV
0b 0000 0010	0xXX	0xXX	0xXXXX	<0x01,0x10,0xXX,...,0xXX>	<0x02,0xYY,0xXX,...,0xXX>

5.15.3 HAP-Token-Update-Request

Table 5-7: HAP-Token-Update-Request

PDU Header				PDU Body	
Control Field (1 Byte)	HAP Opcode (1 Byte)	TID (1 Byte)	CharID (2 Bytes)	Body length (2 Bytes)	Software Token TLV
0b 0000 0000	0x11	0xXX	0xXX 0xXX	0xXX 0xXX	<0x02, 0xYY, 0xXX, ..., 0xXX>

5.15.4 HAP-Token-Update-Response

Table 5-8: HAP-Token-Update-Response

PDU Header		
Control Field (1 Byte)	TID (1 Byte)	Status (1 Byte)
0b 0000 0010	0xXX	0xXX

Table 5-9: Info Parameter Types

Info Parameter Type	Description
0x01	HAP-Param-Current-State-Number
0x02	HAP-Param-Current-Config-Number
0x03	HAP-Param-Device-Identifier
0x04	HAP-Param-Feature-Flags
0x05	HAP-Param-Model-Name
0x06	HAP-Param-Protocol-Version
0x07	HAP-Param-Status-Flag
0x08	HAP-Param-Category-Identifier
0x09	HAP-Param-Setup-Hash

5.15.5 HAP-Info-Request

The HAP-Info-Request PDU is used to request all the protocol paramaters for the accessory as defined in "5.15.6 HAP-Info-Response" (page 61).

Table 5-10: HAP-Info-Request

PDU Header			
Control Field (1 Byte)	HAP PDU Type (1 Byte)	TID (1 Byte)	CharID (2 Bytes)
0b 0000 0000	0x12	0xXX	0xXXXX

5.15.6 HAP-Info-Response

Table 5-11: HAP-Info-Response(With Protocol Parameters) 1

PDU Header			PDU Body		
Control Field (1 Byte)	TID (1 Byte)	Status (1 Byte)	Body Length (2 Bytes)	TLV (State Number)	TLV (Config Number)
0b 0000 0010	0xXX	0xXX	0xXXXX	<0x01,0x02, 0xXXXX>	<0x02,0x02, 0xXXXX>

Table 5-12: HAP-Info-Response(With Protocol Parameters) 2

PDU Body (continued)				
TLV (Device Identifier)	TLV (Feature Flags)	TLV (Model Name)	TLV (Protocol Version)	TLV (Status Flag)
<0x03, 0x06, 0xXX XX XX XX XX XX>	<0x04 0x01, 0xXX>	<0x05, 0xXX, 0xXX...0xYY>	<0x06, 0xXX, 0xXX...0xYY>	<0x07, 0x01, 0xXX>

Table 5-13: HAP-Info-Response(With Protocol Parameters) 3

PDU Body (continued)	
TLV (Category Identifier)	TLV (Setup Hash)
<0x08, 0x02, 0xXXXX>	<0x09, 0x04, 0xXX....0xYY >

5.15.7 Transient Pair Setup with Software Authentication Procedure

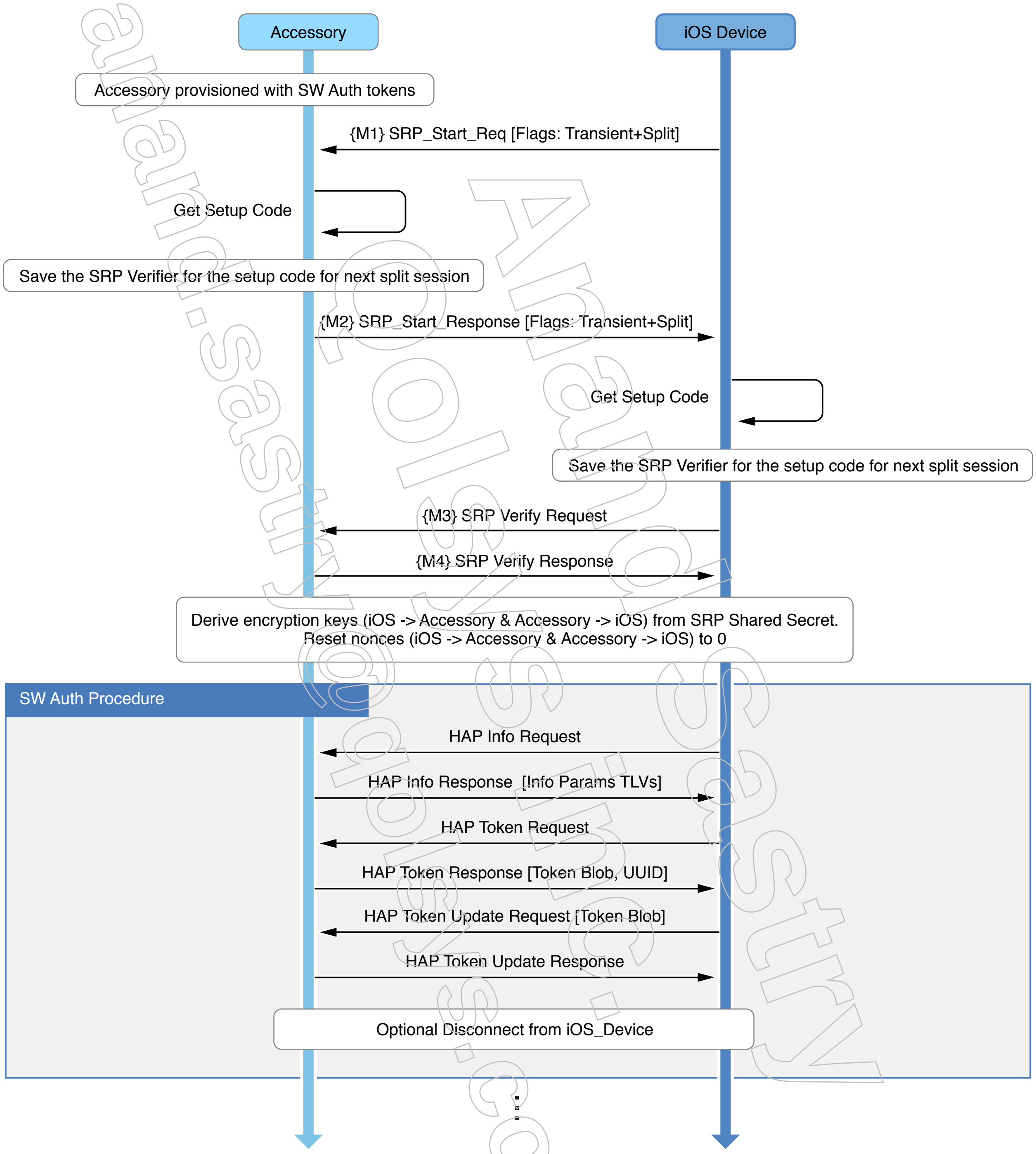

Accessory
iOS Device
Accessory provisioned with SW Auth tokens
{M1} SRP_Start_Req [Flags: Transient+Split]
Get Setup Code
Save the SRP Verifier for the setup code for next split session
{M2} SRP_Start_Response [Flags: Transient+Split]
Get Setup Code
Save the SRP Verifier for the setup code for next split session
{M3} SRP Verify Request
{M4} SRP Verify Response
Derive encryption keys (iOS -> Accessory & Accessory -> iOS) from SRP Shared Secret.
Reset nonces (iOS -> Accessory & Accessory -> iOS) to 0
SW Auth Procedure
HAP Info Request
HAP Info Response [Info Params TLVs]
HAP Token Request
HAP Token Response [Token Blob, UUID]
HAP Token Update Request [Token Blob]
HAP Token Update Response
Optional Disconnect from iOS_Device

Figure 5-1: Transient Pair Setup with Software Authentication Procedure (1)

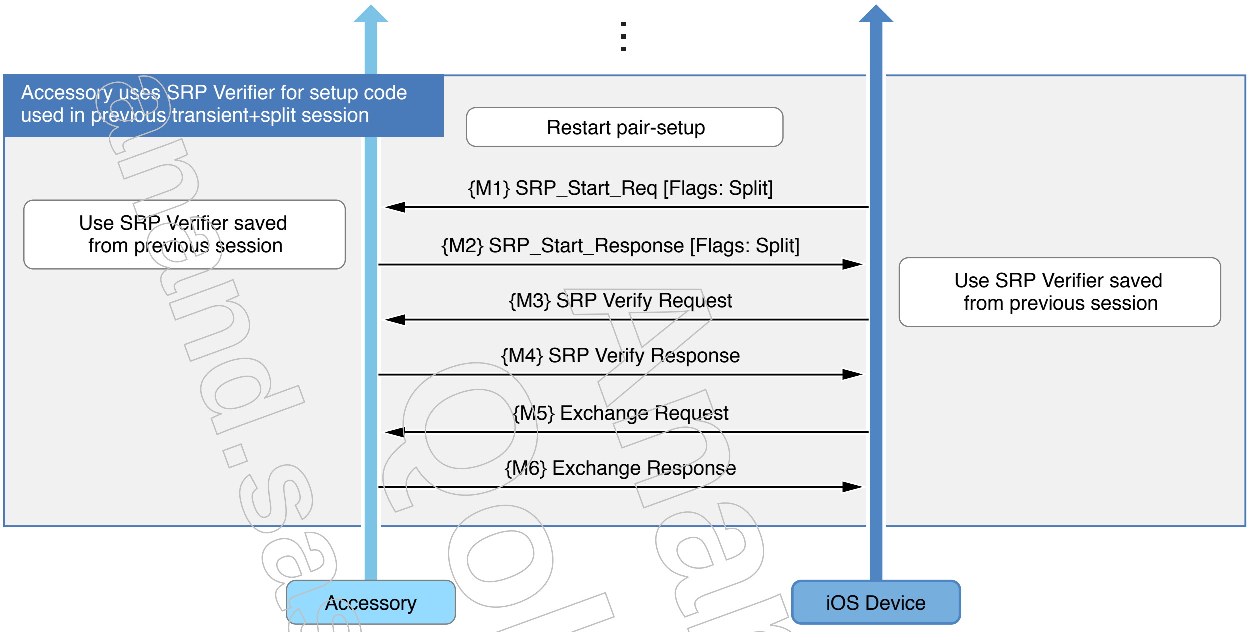

⋮
Accessory uses SRP Verifier for setup code used in previous transient+split session
Restart pair-setup
Use SRP Verifier saved from previous session
{M1} SRP_Start_Req [Flags: Split]
{M2} SRP_Start_Response [Flags: Split]
Use SRP Verifier saved from previous session
{M3} SRP Verify Request
{M4} SRP Verify Response
{M5} Exchange Request
{M6} Exchange Response
Accessory
iOS Device

Figure 5-2: Transient Pair Setup with Software Authentication Procedure (2)

5.16 Methods, Error Codes, and TLV Values

Table 5-14: Methods

Value	Description
0	Pair Setup.
1	Pair Setup with Auth.
2	Pair Verify.
3	Add Pairing.
4	Remove Pairing.
5	List Pairings.
6-255	Reserved.

Table 5-15: Pairing Feature Flags

Mask	Bit	Description
`0x01`	1	Supports Apple Authentication Coprocessor.
`0x02`	2	Supports Software Authentication.
`0x04-0x80`	3-8	Reserved.

Table 5-16: Error Codes

Value	Name	Description
`0x00`	n/a	Reserved.
`0x01`	`kTLVError_Unknown`	Generic error to handle unexpected errors.
`0x02`	`kTLVError_Authentication`	Setup code or signature verification failed.
`0x03`	`kTLVError_Backoff`	Client must look at the retry delay TLV item and wait that many seconds before retrying.
`0x04`	`kTLVError_MaxPeers`	Server cannot accept any more pairings.
`0x05`	`kTLVError_MaxTries`	Server reached its maximum number of authentication attempts.
`0x06`	`kTLVError_Unavailable`	Server pairing method is unavailable.
`0x07`	`kTLVError_Busy`	Server is busy and cannot accept a pairing request at this time.
`0x08–0xFF`	n/a	Reserved.

Table 5-17: TLV Values

Type	Name	Format	Description
`0x00`	kTLVType_Method	integer	Method to use for pairing. See Table 5-14 (page 63).
`0x01`	kTLVType_Identifier	UTF-8	Identifier for authentication.
`0x02`	kTLVType_Salt	bytes	16+ bytes of random salt.
`0x03`	kTLVType_PublicKey	bytes	Curve25519, SRP public key, or signed Ed25519 key.
`0x04`	kTLVType_Proof	bytes	Ed25519 or SRP proof.
`0x05`	kTLVType_EncryptedData	bytes	Encrypted data with auth tag at end.
`0x06`	kTLVType_State	integer	State of the pairing process. 1=M1, 2=M2, etc.
`0x07`	kTLVType_Error	integer	Error code. Must only be present if error code is not 0. See Table 5-16 (page 64).
`0x08`	kTLVType_RetryDelay	integer	Seconds to delay until retrying a setup code.
`0x09`	kTLVType_Certificate	bytes	X.509 Certificate.
`0x0A`	kTLVType_Signature	bytes	Ed25519 or Apple Authentication Coprocessor signature.
`0x0B`	kTLVType_Permissions	integer	Bit value describing permissions of the controller being added. None (`0x00`) : Regular user Bit 1 (`0x01`) : Admin that is able to add and remove pairings against the accessory.
`0x0C`	kTLVType_FragmentData	bytes	Non-last fragment of data. If length is 0, it's an ACK.
`0x0D`	kTLVType_FragmentLast	bytes	Last fragment of data.
`0x13`	kTLVType_Flags	integer	Pairing Type Flags (32 bit unsigned integer). See Table 5-18 (page 65)
`0xFF`	kTLVType_Separator	null	Zero-length TLV that separates different TLVs in a list.

Table 5-18: Pairing Type Flags

Mask	Bit	Description
`0x00000010`	4	BitMask (1 « 4) Transient Pair-Setup (kPairingFlag_Transient) Pair Setup M1 - M4 without exchanging public keys
`0x01000000`	24	BitMask (1 « 24) Split-Pair Setup (kPairingFlag_Split) When Set with kPairingFlag_Transient save the SRP Verifier used in this session, and when only kPairingFlag_Split is set, use the saved SRP verifier from previous session.
`others`	others	Reserved.

6 HomeKit Accessory Protocol for IP Accessories

6.1 Overview

This chapter describes the HomeKit Accessory Protocol (HAP) for IP accessories.

6.2 Requirements

6.2.1 IP Requirements

- HAP accessory servers must support IPv4.
- HAP accessory servers must act as a DHCPv4 client.
- HAP accessory servers may support static IPv4 addresses.
- HAP accessory servers must support link-local IPv4 addressing as described in RFC 3927.
- HAP accessory servers must support link-local IPv6 addressing as described in RFC 4862.
- HAP accessory servers must support simultaneous IPv4 and IPv6 connections.

6.2.2 Bonjour Requirements

- HAP accessory servers must support Multicast DNS host names, e.g. "lights.local".
- HAP accessory servers must handle name collisions as described in Section 9 of RFC 6762.
- HAP accessory servers must support advertising a DNS service that includes a TXT record with multiple keys as described in Section 6 of RFC 6763.
- HAP accessory servers must pass the Bonjour Conformance Test, which is available at `https://developer.apple.com/softwarelicensing/agreements/bonjour.php`. The HAP accessory server must pass the following tests and all of the associated subtests:
  - Link-local Address Allocation
  - IPv4 Multicast-DNS
  - IPv6 Multicast-DNS
  - Mixed-network Interoperability
- Refer to the Bonjour for Developers webpage for Bonjour source code and specifications.

6.2.3 TCP Requirements

- HAP accessory servers must support eight simultaneous TCP connections.
- HAP accessory servers must always be able to accommodate new incoming TCP connections.

6.2.4 HTTP Server Requirements

- The HTTP server must be HTTP/1.1 compliant as described in RFC 7230 and RFC 7231.
- The HTTP server must support persistent connections, i.e. multiple HTTP requests through single TCP connection, as described in Section 6.3 of RFC 7230.
- The HTTP server must support the HTTP URLs summarized in Table 6-14 (page 93) to service HAP clients.

6.2.5 Application Requirements

- HAP accessory servers must be able to receive and process HAP requests at any time once the HAP Bonjour service has been registered.
- HAP accessory servers must support parsing and generating JavaScript Object Notation (JSON) as described in RFC 7159.
- HAP accessory servers must encode JSON in UTF-8.
- HAP accessory servers must support multiple connections.

6.2.6 General Requirements

- HAP accessory servers must support firmware update via all IP interfaces. For example, if an accessory server supports both Ethernet and Wi-Fi it must be able to update its firmware over both Ethernet and Wi-Fi.

6.3 HAP Objects

6.3.1 Accessory Objects

HAP Accessory Objects have the following required properties.

Table 6-1: Properties of HAP Accessory Objects in JSON

Property	Key	JSON Type	Description
Accessory Instance ID	”aid”	Number	Integer assigned by the HAP Accessory Server to uniquely identify the HAP Accessory object, see ”2.6.1 Instance IDs” (page 27).
Services	”services”	Array	Array of Service objects. Must not be empty. The maximum number of services must not exceed 100.

6.3.2 Service Objects

Service objects have the following required properties:

Table 6-2: Properties of Service Objects in JSON

Property	Key	JSON Type	Description
Type	”type”	String	String that defines the type of the service. See ”6.6.1 Service and Characteristic Types” (page 74).
Instance ID	”iid”	Number	Integer assigned by the HAP Accessory Server to uniquely identify the HAP Service object, see ”2.6.1 Instance IDs” (page 27).
Characteristics	“characteristics”	Array	Array of Characteristic objects. Must not be empty. The maximum number of characteristics must not exceed 100, and each characteristic in the array must have a unique type.
Hidden Services	”hidden”	Boolean	When set to True, this service is not visible to user.
Primary Services	”primary”	Boolean	When set to True, this is the primary service on the accessory.
Linked Services	”linked”	Array	An array of Numbers containing the instance ids of the services that this service links to.

6.3.3 Characteristic Objects

Characteristic objects have the following defined properties.

After an accessory has a pairing, only the values of the Value (”value”) and Event Notifications (”ev”) properties are allowed to change.

Table 6-3: Properties of Characteristic Objects in JSON

Property	Key	JSON Type	Required?	Description
Type	”type”	String	Yes	String that defines the type of the characteristic. See ”6.6.1 Service and Characteristic Types” (page 74).
Instance ID	”iid”	Number	Yes	Integer assigned by the HAP Accessory Server to uniquely identify the HAP Characteristic object, see ”2.6.1 Instance IDs” (page 27).

Value	”value”	<TYPE>	Yes	The value of the characteristic, which must conform to the ”format” property. The literal value `null` may also be used if the characteristic has no value. This property must be present if and only if the characteristic contains the `Paired Read` permission, see Table 6-4 (page 70).
Permissions	”perms”	Array	Yes	Array of permission strings describing the capabilities of the characteristic. See Table 6-4 (page 70).
Event Notifications	”ev”	Boolean	No	Boolean indicating if event notifications are enabled for this characteristic.
Description	”description”	String	No	String describing the characteristic on a manufacturer-specific basis, such as an indoor versus outdoor temperature reading.
Format	”format”	String	Yes	Format of the value, e.g. ”float”. See Table 6-5 (page 71).
Unit	”unit”	String	No	Unit of the value, e.g. ”celsius”. See Table 6-6 (page 71).
Minimum Value	”minValue”	Number	No	Minimum value for the characteristic, which is only appropriate for characteristics that have a format of ”int” or ”float”.
Maximum Value	”maxValue”	Number	No	Maximum value for the characteristic, which is only appropriate for characteristics that have a format of ”int” or ”float”.
Step Value	”minStep”	Number	No	Minimum step value for the characteristic, which is only appropriate for characteristics that have a format of ”int” or ”float”. For example, if this were 0.15, the characteristic value can be incremented from the minimum value in multiples of 0.15. For ”float”, the ”Value” needs to be rounded on the accessory side to the closest allowed value per the ”Step Value” (e.g. a value of 10.150001 received on the accessory side with a ”Step Value” of 0.15 needs to be interpreted as 10.15).
Max Length	”maxLen”	Number	No	Maximum number of characters if the format is ”string”. If this property is omitted for ”string” formats, then the default value is 64. The maximum value allowed is 256.

Max Data Length	”maxDataLen”	Number	No	Maximum number of characters if the format is ”data”. If this property is omitted for ”data” formats, then the default value is 2097152.
Valid Values	”valid-values”	Array	No	An array of Numbers where each element represents a valid value.
Valid Values Range	”valid-values-range”	Array	No	A 2 element array representing the starting value and ending value of the range of valid values.
TTL	”TTL”	Number	No	Specified TTL in milliseconds the controller requests the accessory to securely execute a write command. Maximum value of this is 9007199254740991
PID	”pid”	Number	No	64-bit unsigned integer assigned by the controller to uniquely identify the timed write transaction.

Table 6-4: Characteristic Properties

Property	JSON String	Description
Paired Read	”pr”	This characteristic can only be read by paired controllers.
Paired Write	”pw”	This characteristic can only be written by paired controllers.
Events	”ev”	This characteristic supports events. The HAP Characteristic object must contain the ”ev” key if it supports events.
Additional Authorization	”aa”	This characteristic supports additional authorization data
Timed Write	”tw”	This characteristic allows only timed write procedure
Hidden	”hd”	This characteristic is hidden from the user
Write Response	”wr”	This characteristic supports write response

Table 6-5: Characteristic Value Formats

Format JSON String	Description
"bool"	Boolean value expressed as one of the following: true, false, 0 (false), and 1 (true).
"uint8"	Unsigned 8-bit integer.
"uint16"	Unsigned 16-bit integer.
"uint32"	Unsigned 32-bit integer.
"uint64"	Unsigned 64-bit integer.
"int"	Signed 32-bit integer.
"float"	Signed 64-bit floating point number.
"string"	Sequence of zero or more Unicode characters, encoded as UTF-8. Maximum length is 64 bytes unless overridden by the "maxLen" property.
"tlv8"	Base64-encoded set of one or more TLV8's.
"data"	Base64-encoded data blob. Maximum length is 2,097,152 bytes unless overridden by the "maxDataLen" property.

Table 6-6: Characteristic Units

Format JSON String	Description
"celsius"	The unit is only "degrees Celsius".
"percentage"	The unit is in percentage "%".
"arcdegrees"	The unit is in arc degrees.
"lux"	The unit is in lux.
"seconds"	The unit is in seconds.

6.4 Discovery

Accessories advertise their presence on the network via Bonjour. HAP clients browse for them and if any are found and determined to be compatible, they are displayed to the user. The Bonjour service type is:

```
_hap._tcp
```

The name of the Bonjour service is the user-visible name of the accessory, e.g. "LED Bulb M123", and must match the name provided in the Accessory Information Service of the HAP Accessory object that has an instanceID of 1. The name may contain any Unicode character and is encoded using UTF-8. It has a maximum length of 63 bytes, which may be fewer than 63 characters as a single Unicode character may require multiple bytes. Additional data needed for discovery-time metadata is advertised via a TXT record associated with the Bonjour service. TXT record keys are intentionally short to minimize network traffic. Accessories must confirm to the TXT records as defined in Table 6-7 (page 72)

Table 6-7: _hap._tcp Bonjour TXT Record Keys

Key	Description
”c#”	Current configuration number. Required. Must update when an accessory, service, or characteristic is added or removed on the accessory server. Accessories must increment the config number after a firmware update. This must have a range of 1-65535 and wrap to 1 when it overflows. This value must persist across reboots, power cycles, etc.
”ff”	Pairing Feature flags (e.g. ”0x3” for bits 0 and 1). Required if non-zero. See Table 5-15 (page 63).
”id”	Device ID (”5.4 Device ID” (page 38)) of the accessory. The Device ID must be formatted as ”XX:XX:XX:XX:XX:XX”, where ”XX” is a hexadecimal string representing a byte. Required. This value is also used as the accessory's Pairing Identifier.
”md”	Model name of the accessory (e.g. ”Device1,1”). Required.
”pv”	Protocol version string ”X.Y” (e.g. ”1.0”). Required if value is not ”1.0”. (see ”6.6.3 IP Protocol Version” (page 75))
”s#”	Current state number. Required. This must have a value of ”1”.
”sf”	Status flags (e.g. ”0x04” for bit 3). Value should be an unsigned integer. See Table 6-8 (page 72). Required.
”ci”	Accessory Category Identifier. Required. Indicates the category that best describes the primary function of the accessory. This must have a range of 1-65535. This must take values defined in ”14-1 Accessory Categories” (page 271). This must persist across reboots, power cycles, etc.
”sh”	Setup Hash. See (”4.2.3 Setup Hash” (page 33)) Required if the accessory supports enhanced setup payload information.

Table 6-8: Bonjour TXT Status Flags

Mask	Bit	Description
0x01	1	Accessory has not been paired with any controllers.
0x02	2	Accessory has not been configured to join a Wi-Fi network.
0x04	3	A problem has been detected on the accessory.
0x08–0x80	4-8	Reserved.

6.5 Security for IP Accessories

Any connection using HomeKit Accessory Protocol must be secure. Authenticated encryption for HAP is provided by two mechanisms: pairing, which is defined in "5 Pairing" (page 38), and session security, which is defined below.

A connection that does not use HAP should also be secured, and security negotiation for the connection should use the HAP connection. For example, audio and video streams from an accessory to a controller should be secured using SRTP with the SRTP security configuration itself taking place over the HAP connection.

6.5.1 Pairing

"5 Pairing" (page 38) defines a secure mechanism used to pair a controller and an accessory. The pairing is established once and verified for every session. After the pairing has been verified, both sides have an ephemeral shared secret that can be used to encrypt the session.

In Pair Setup, the Pairing Identifier used for the `kTLVType_Identifier` must be the accessory's globally unique ID as defined by the "id" key in the Table 6-7 (page 72).

6.5.2 Session Security

Once the controller and accessory have established an authenticated ephemeral shared secret using Pair Verify, both the controller and the accessory use the shared secret to derive the read and write keys for session security:

```
AccessoryToControllerKey = HKDF-SHA-512 of
    InputKey   = <Pair Verify shared secret>
    Salt       = "Control-Salt"
    Info       = "Control-Read-Encryption-Key"
    OutputSize = 32 bytes

ControllerToAccessoryKey = HKDF-SHA-512 of
    InputKey   = <Pair Verify shared secret>
    Salt       = "Control-Salt"
    Info       = "Control-Write-Encryption-Key"
    OutputSize = 32 bytes
```

The controller and accessory use the derived keys in the following manner:

Table 6-9: Derived Session Key Usage

	Format JSON String	Description
Accessory	AccessoryToControllerKey	ControllerToAccessoryKey
Controller	ControllerToAccessoryKey	AccessoryToControllerKey

Each HTTP message must be secured with the AEAD algorithm AEAD_CHACHA20_POLY1305 as specified in Section 2.8 of RFC 7539. The 32-bit fixed-common part of the 96-bit nonce is all zeros: `00 00 00 00`.

Each HTTP message is split into frames no larger than 1024 bytes. Each frame has the following format:

```
<2:AAD for little endian length of encrypted data (n) in bytes>
     <n:encrypted data according to AEAD algorithm, up to 1024 bytes>
     <16:authTag according to AEAD algorithm>
```

Once session security has been established, if the accessory encounters a decryption failure then it must immediately close the connection used for the session.

6.6 IP Accessory Attribute Database

The accessory attribute database is a list of HAP Accessory objects, Service objects, and Characteristic objects serialized into JSON. As such, name/value pairs in JSON objects are unordered, and values in JSON arrays are ordered.

6.6.1 Service and Characteristic Types

Service and Characteristic types are UUIDs as defined by RFC 4122. Apple-defined service and characteristic types are based on the following HAP Base UUID:

```
00000000-0000-1000-8000-0026BB765291
```

When Apple-defined UUIDs are encoded as JSON strings, a short form must be used by including only the first 8 characters with leading zeros removed. When converting short form UUIDs back to full UUIDs, the process is reversed by prefixing the string with 0 or more ’0’ characters to expand it to 8 characters then ”-0000-1000-8000-0026BB765291” is appended to form the full 36-character UUID.

For example, the public.hap.service.accessory-information full UUID is ”0000003E-0000-1000-8000-0026BB765291”. When encoded as a JSON string, the first 8 characters are ”0000003E” and with leading zeros removed it becomes ”3E”. Converting back to a full UUID would prefix ”3E” with six ’0’ characters to get ”0000003E” then append ”-0000-1000-8000-0026BB765291” to form a full UUID of ”0000003E-0000-1000-8000-0026BB765291”. Other examples:

```
”00000001-0000-1000-8000-0026BB765291” -> ”1”
”00000F25-0000-1000-8000-0026BB765291” -> ”F25”
”0000BBAB-0000-1000-8000-0026BB765291” -> ”BBAB”
”010004FF-0000-1000-8000-0026BB765291” -> ”10004FF”
”FF000000-0000-1000-8000-0026BB765291” -> ”FF000000”
```

Custom service and characteristic types must be 128-bit UUIDs as defined by RFC 4122. Custom types must be encoded as the full UUID string in JSON. Custom types must not use the HAP Base UUID.

6.6.2 Getting the Accessory Attribute Database

A controller obtains the attribute database by sending the accessory an HTTP GET request to `/accessories`:

```
GET /accessories HTTP/1.1
Host: lights.local:12345
```

If the controller is paired with the accessory then the accessory will return the attribute database in the message body of the HTTP response.

Controllers should cache the attribute database and monitor the current configuration number, c# in Table 6-7 (page 72), for changes, which indicates that the controller should get the attribute database and update its cache. Unpaired controllers must not cache the attribute database.

If an updated attribute database shows that it is missing some services and characteristics, then these service and characteristics will be deleted from the home.

6.6.3 IP Protocol Version

IP accessories must include ”9.2 HAP Protocol Information” (page 153). For a bridge accessory, only the primary HAP accessory object must contain ”9.2 HAP Protocol Information” (page 153). The IP protocol version is a string with the following format: ”X.Y.Z”. Please see ”10.37 Version” (page 193) for detailed definition of the characteristic. This must be set to ”1.1.0” for this version of HAP IP. The ”pv” Key in Bonjour TXT record (Table 6-7 (page 72)) is truncated to ”X.Y”. It must be set to ”1.1” for this version of HAP IP.

6.6.4 Example Accessory Attribute Database in JSON

Below is an example HTTP response containing the HAP attribute database for a lightbulb bridge with two lightbulbs:

```
    HTTP/1.1 200 OK
    Content-Type: application/hap+json
    Content-Length: <length>

    {
        "accessories" : [
            {
                "aid" : 1,
                "services" : [
                    {
                        "type" : "3E",
                        "iid" : 1,
                        "characteristics" : [
                            {
                                "type" : "23",
                                "value" : "Acme Light Bridge",
                                "perms" : [ "pr" ],
                                "format" : "string",
                                "iid" : 2
                            },
                            {
                                "type" : "20",
                                "value" : "Acme",
                                "perms" : [ "pr" ],
                                "format" : "string",
                                "iid" : 3
                            },
                            {
                                "type" : "30",
                                "value" : "037A2BABF19D",
                                "perms" : [ "pr" ],
                                "format" : "string",
                                "iid" : 4
                            },
                            {
                                "type" : "21",
                                "value" : "Bridge1,1",
                                "perms" : [ "pr" ],
                                "format" : "string",
                                "iid" : 5
                            },
                            {
                                "type" : "14",
                                "value" : null,
                                "perms" : [ "pw" ],
                                "format" : "bool",
                                "iid" : 6
                            },
                            {
                                "type" : "52",
                                "value" : "100.1.1",
                                "perms" : [ "pr" ],
                                "format" : "string",
                                "iid" : 7
                            },
                        ]
                    },
                    {
                        "type" : "A2",
                        "iid" : 8,
                        "characteristics" : [
                            {
                                "type" : "37",
                                "value" : "01.01.00",
                                "perms" : [ "pr" ],
                                    ”format” : ”string”,
                                    ”iid” : 9
                                }
                            ]
                        }
                    ]
                },
                {
                    ”aid” : 2,
                    ”services” : [
                        {
                            ”type” : ”3E”,
                            ”iid” : 1,
                            ”characteristics” : [
                                {
                                    ”type” : ”23”,
                                    ”value” : ”Acme LED Light Bulb”,
                                    ”perms” : [ ”pr” ],
                                    ”format” : ”string”,
                                    ”iid” : 2
                                },
                                {
                                    ”type” : ”20”,
                                    ”value” : ”Acme”,
                                    ”perms” : [ ”pr” ],
                                    ”format” : ”string”,
                                    ”iid” : 3
                                },
                                {
                                    ”type” : ”30”,
                                    ”value” : ”099DB48E9E28”,
                                    ”perms” : [ ”pr” ],
                                    ”format” : ”string”,
                                    ”iid” : 4
                                },
                                {
                                    ”type” : ”21”,
                                    ”value” : ”LEDBulb1,1”,
                                    ”perms” : [ ”pr” ],
                                    ”format” : ”string”,
                                    ”iid” : 5
                                },
                                {
                                    ”type” : ”14”,
                                    ”value” : null,
                                    ”perms” : [ ”pw” ],
                                    ”format” : ”bool”,
                                    ”iid” : 6
                                }
                            ]
                        },
                        {
                            "type" : "43",
                            "iid" : 7,
                            "characteristics" : [
                                {
                                    "type" : "25",
                                    "value" : true,
                                    "perms" : [ "pr", "pw" ],
                                    "format" : "bool",
                                    "iid" : 8
                                },
                                {
                                    "type" : "8",
                                    "value" : 50,
                                    "perms" : [ "pr", "pw" ],
                                    "iid" : 9,
                                    "maxValue" : 100,
                                    "minStep" : 1,
                                    "minValue" : 20,
                                    "format" : "int",
                                    "unit" : "percentage"
                                }
                            ]
                        }
                    ]
                },
                {
                    "aid" : 3,
                    "services" : [
                        {
                            "type" : "3E",
                            "iid" : 1,
                            "characteristics" : [
                                {
                                    "type" : "23",
                                    "value" : "Acme LED Light Bulb",
                                    "perms" : [ "pr" ],
                                    "format" : "string",
                                    "iid" : 2
                                },
                                {
                                    "type" : "20",
                                    "value" : "Acme",
                                    "perms" : [ "pr" ],
                                    "format" : "string",
                            ″iid″ : 3
                        },
                        {
                            ″type″ : ″30″,
                            ″value″ : ″099DB48E9E28″,
                            ″perms″ : [ ″pr″ ],
                            ″format″ : ″string″,
                            ″iid″ : 4
                        },
                        {
                            ″type″ : ″21″,
                            ″value″ : ″LEDBulb1,1″,
                            ″perms″ : [ ″pr″ ],
                            ″format″ : ″string″,
                            ″iid″ : 5
                        },
                        {
                            ″type″ : ″14″,
                            ″value″ : null,
                            ″perms″ : [ ″pw″ ],
                            ″format″ : ″bool″,
                            ″iid″ : 6
                        }
                    ]
                },
                {
                    ″type″ : ″43″,
                    ″iid″ : 7,
                    ″characteristics″ : [
                        {
                            ″type″ : ″25″,
                            ″value″ : true,
                            ″perms″ : [ ″pr″, ″pw″ ],
                            ″format″ : ″bool″,
                            ″iid″ : 8
                        },
                        {
                            ″type″ : ″8″,
                            ″value″ : 50,
                            ″perms″ : [ ″pr″, ″pw″ ],
                            ″iid″ : 9,
                            ″maxValue″ : 100,
                            ″minStep″ : 1,
                            ″minValue″ : 20,
                            ″format″ : ″int″,
                            ″unit″ : ″percentage″
                        }
                    ]
```

```
                    }
                ]
            }
        ]
    }
```

6.7 Controlling IP Accessories

6.7.1 Handling HTTP Requests

The accessory must respond to HTTP requests with an HTTP response that includes an appropriate HTTP status code:

6.7.1.1 Successful HTTP Status Codes

- `200 OK` when the request was successfully executed and the response includes a body, e.g. after successfully reading characteristics.
- `204 No Content` when the request was successfully executed and the response has no body, e.g. after successfully writing characteristics.
- `207 Multi-Status` when the request was processed but was not executed (partially or completely), e.g. when writing two characteristics, one fails or both writes fails, when writing one characteristic and the write fails.

An accessory that can successfully process the request must return a Successful HTTP Status Code. The body of the response may be empty, or a JSON object, depending on the specification of the request.

6.7.1.2 Client Error HTTP Status Codes

- 400 Bad Request on HTTP client error, e.g. a malformed request.
- 404 Not Found on an invalid HTTP URL.
- 422 Unprocessable Entity for a well-formed request that contains invalid HTTP parameters.

6.7.1.3 Server Error Status HTTP Codes

- 500 Internal Server Error on an accessory server error, e.g. the operation timed out.
- 503 Service Unavailable if the accessory server is too busy to service the request, e.g. reached its maximum number of connections.

6.7.1.4 HAP Status Codes

If the HTTP Status Code is a Client Error (4xx) or Server Error (5xx), then the response must include an HAP Status Code property ("status") indicating the error code.

If the HTTP Status Code is `207 Multi-Status`, then each characteristic in the response body must include a "status" property indicating the success or failure HAP status code for each characteristic.

Table 6-10: HAP Status Code Property

Property	Key	JSON Type	Description
HAP Status Code	"status"	Number	A specific status code. See Table 6-11 (page 81) for more details on the supported status codes.

Table 6-11: HAP Status Codes

Status Code	Description
0	This specifies a success for the request.
-70401	Request denied due to insufficient privileges.
-70402	Unable to communicate with requested service, e.g. the power to the accessory was turned off.
-70403	Resource is busy, try again.
-70404	Cannot write to read only characteristic.
-70405	Cannot read from a write only characteristic.
-70406	Notification is not supported for characteristic.
-70407	Out of resources to process request.
-70408	Operation timed out.
-70409	Resource does not exist.
-70410	Accessory received an invalid value in a write request.
-70411	Insufficient Authorization.

6.7.2 Writing Characteristics

The accessory must support writing values to one or more characteristics via a single HTTP request.

To write one or more characteristic values, the controller sends an HTTP PUT request to `/characteristics`. The HTTP body is a JSON object that contains an array of Characteristic Write Request objects.

6.7.2.1 Characteristic Write Request Objects

Characteristic Write Request objects have the following defined properties:

Table 6-12: Properties of Characteristic Write Objects in JSON

Property	Key	JSON Type	Description
Accessory Instance ID	"aid"	Number	The instance ID of the accessory that contains the characteristic to be written. Required.
Instance ID	"iid"	Number	The instance ID of the characteristic to be written. If a provided instance ID is not a Characteristic object, the accessory must respond with an "Invalid Parameters" error. See Table 6-11 (page 81). Required.
Value	"value"	<TYPE>	Optional property that contains the value to be written to the characteristic.
Events	"ev"	Boolean	Optional property that indicates the state of event notifications for the characteristic
Authorization Data	"authData"	String	Optional property that contains a base 64 encoded string of the authorization data associated with the characteristic.
Remote	"remote"	Boolean	Optional property that indicates if remote access was used to send the request. A value of `true` indicates remote access was used.
Response	"r"	Boolean	Optional property, a Boolean value indicating whether a response value should be included in the response to a write request for a characteristic that supports Write Response.

At least one of `"value"` or `"ev"` will be present in the characteristic write request object. All characteristics write operations must complete with success or failure before sending a response or handling other requests. If no error occurs, the accessory must send an HTTP response with a `204 No Content` status code and an empty body.

6.7.2.2 Write a Single Characteristic

For example, to set the "10.26 On" (page 188) characteristic, which has an instance ID of "8", of a "9.5 Lightbulb" (page 154) service on an accessory with an instance ID of "2", the HTTP PUT request will be:

```
PUT /characteristics HTTP/1.1
Host: lights.local:12345
Content-Type: application/hap+json
Content-Length: <length>

{
    "characteristics" : [
        {
            "aid" : 2,
            "iid" : 8,
            "value" : true
        }
    ]
}
```

If no error occurs then the accessory must respond with a `204 No Content` HTTP Status Code and an empty body. The "`value`" key must never be included as part of the response to a write request.

```
HTTP/1.1 204 No Content
```

Otherwise, if there is an error, then the accessory must respond with a `207 Multi-Status` HTTP Status Code and an appropriate HAP status code indicating the error. For example, if the above write request was for a characteristic that was read-only (i.e. the value of the permissions property was only Paired Read), then the response would be the following:

```
HTTP/1.1 207 Multi-Status
Content-Type: application/hap+json
Content-Length: <length>

{
    "characteristics" : [
        {
            "aid" : 2,
            "iid" : 8,
            "status" : -70404
        }
    ]
}
```

6.7.2.3 Writing Multiple Characteristics

To write multiple characteristics, the controller will send additional Characteristic Write Request objects to the accessory. For example, to write a Boolean value to the "10.26 On" (page 188) characteristic of "lightA", which has a characteristic instance ID of "8" and an accessory instance ID of "2", and "lightB", which has a characteristic instance ID of "8" and an accessory instance ID of "3", the controller will send the following HTTP PUT request:

```
PUT /characteristics HTTP/1.1
Content-Type: application/hap+json
Content-Length: <length>

{
    "characteristics" : [
        {
            "aid" : 2,
            "iid" : 8,
            "value" : true
        },
        {
            "aid" : 3,
            "iid" : 8,
            "value" : true
        }
    ]
}
```

If no errors occur then the accessory must respond with a `204 No Content` HTTP Status Code and an empty body. The `"value"` key must never be included as part of the response to a write request.

```
HTTP/1.1 204 No Content
```

If an error occurs when attempting to write any characteristics, e.g. the physical devices represented by the characteristics to be written were unreachable, the accessory must respond with a `207 Multi-Status` HTTP Status Code and each response object must contain a `"status"` entry. Characteristics that were written successfully must have a `"status"` of `0` and characteristics that failed to be written must have a non-zero `"status"` entry. An example is listed below where one write failed and one succeeded:

```
HTTP/1.1 207 Multi-Status
Content-Type: application/hap+json
Content-Length: <length>

{
    "characteristics" : [
        {
            "aid" : 2,
            "iid" : 8,
            "status" : 0
        },
        {
            "aid" : 3,
            "iid" : 8,
            "status" : -70402
        }
    ]
}
```

6.7.2.4 Timed Write Procedures

Timed Write Procedure will be used to write to characteristic that require time sensitive actions. If a characteristic requires timed writes, the accessory must indicate that using HAP Characteristics properties.

An accessory must support timed writes to all characteristics even if the characteristic property does not require it.

Timed Write Procedure will be used to securely execute a write command within a specified TTL. The accessory starts the TTL timer after sending the response to a Prepare Write Request. The scheduled request must be executed only if the accessory receives an Execute Write Request before its TTL timer expires.

If the accessory receives an Execute Write Request after the TTL has expired it must ignore the request and respond with HAP status error code -70410 (HAPIPStatusErrorCodeInvalidWrite).

If the accessory receives consecutive Prepare Write Requests in the same session, the accessory must reset the timed write transaction with the TTL specified by the latest request.

If the accessory receives a standard write request on a characteristic which requires timed write, the accessory must respond with HAP status error code -70410 (HAPIPStatusErrorCodeInvalidWrite).

When writing multiple characteristics, the controller will send a Prepare Write Request if any of characteristic in the multiple characteristics Write Request requires timed write.

For example, to write a Boolean value to the ″Target Door State″ characteristic of ″Door″, which has a characteristic instance ID of ″6″ and an accessory instance ID of ″2″, ″Lock Target State″ characteristic which has a characteristic instance ID of ″7″ and an accessory instance ID of ″2″ and the ″Lock Management Auto Security Timeout″ characteristic which has a characteristic instance ID of ″8″ and an accessory instance ID of ″2″, the controller will send the following the HTTP PUT request to accessory with accessory instance ID of ″2″:

```
PUT /prepare HTTP/1.1
Host: door.local:12345
Content-Type: application/hap+json
Content-Length: <length>

{
    ″ttl″ : 2500
    ″pid″: 11122333

}
```

The accessory must respond with a 200 OK HTTP Status Code and include a HAP status code indicating if timed write procedure can be executed or not.

```
HTTP/1.1 200 OK
Content-Type: application/hap+json
Content-Length: <length>

{
    ″status″ : 0

}
```

After the controller has received the response to a prepare write request, it will send a commit write request within the TTL milliseconds. The HTTP PUT request will be:

```
PUT /characteristics HTTP/1.1
Content-Type: application/hap+json
Content-Length: <length>

{
”characteristics”: [{
”aid” : 2,
”iid” : 6,
”value” : 1

},
{
”aid” : 2,
”iid” : 7,
”value” : 3

}
{
”aid” : 2,
”iid” : 8,
”value” : 4

}]
”pid” : 11122333

}
```

If no errors occur then the accessory must respond with a 204 No Content HTTP Status Code and an empty body. The ”value” key must never be included as part of the response to a write request.

```
HTTP/1.1 204 No Content
```

If an error occurs when attempting to write any characteristics, the accessory must respond with a 207 Multi-Status HTTP Status Code and each response object must contain a ”status” entry. Characteristics that were written successfully must have a ”status” of 0 and characteristics that failed to be written must have a non-zero ”status” entry. An example is listed below where one write failed and two writes succeeded:

```
HTTP/1.1 207 Multi-Status
Content-Type: application/hap+json
Content-Length: <length>
{
”characteristics”: [{
”aid” : 2,
”iid” : 6,
”status” : 0
},
{
"aid" : 2,
"iid" : 7,
"status" : -70402
}
{
"aid" : 2,
"iid" : 8,
"status" : 0
}]
}
```

6.7.3 Writing a Characteristic with Write Response

Write response property is used for characteristics when a read response is required for a write operation. This property is typically applicable to write control point operations. The accessory must send the write response to write operation as part of the HTTP response using 207 Multi-Status HTTP Status code. In such a case, the controller may not perform an explicit read request to read the result of control point write operation.

6.7.4 Reading Characteristics

The accessory must support reading values from one or more characteristics via a single HTTP request.

To read the value of a characteristic, the controller sends an HTTP GET request to `/characteristics` with a query string that conforms to Section 3.4 of RFC 3986. The query string may have the following parameters:

Table 6-13: HAP GET Request URL Parameters

Parameter	Description
`id`	The identifiers for the characteristics to be read must be formatted as `<Accessory Instance ID>.<Characteristic Instance ID>`, as a comma-separated list. For example, to read the values of characteristics with instance ID "4" and "8" on an accessory with an instanceID of "1" the URL parameter would be `id=1.4,1.8`. `id` is required for all GET requests.
`meta`	Boolean value that determines whether or not the response should include metadata. If `meta` is not present it must be assumed to be "0". If `meta` is "1", then the response must include the following properties if they exist for the characteristic: "format", "unit", "minValue", "maxValue", "minStep", and "maxLen".
`perms`	Boolean value that determines whether or not the response should include the permissions of the characteristic. If `perms` is not present it must be assumed to be "0".
`type`	Boolean value that determines whether or not the response should include the type of characteristic. If `type` is not present it must be assumed to be "0".
`ev`	Boolean value that determines whether or not the "ev" property of the characteristic should be included in the response. If `ev` is not present it must be assumed to be "0".

Each parameter will be delimited by an & character.

6.7.4.1 Reading a Single Characteristic

An example HTTP GET request to read the value of a single characteristic with instance ID ”8”, on an accessory with instance ID ”2”, is the following:

```
GET /characteristics?id=2.8 HTTP/1.1
```

The response must contain a Characteristic object with the value, Accessory Instance ID, and the Characteristic Instance ID. In the example below, the value of the characteristic with instance ID ”8” on the accessory with instance ID ”2” is 'false':

```
HTTP/1.1 200 OK
Content-Type: application/hap+json
Content-Length: <length>

{
    ”characteristics” : [
        {
            ”aid” : 2,
            ”iid” : 8,
            ”value” : false,
        }
    ]
}
```

6.7.4.2 Reading Multiple Characteristics

To read multiple characteristics, for example, the values of all characteristics with instance ID ”8”, across accessories with instance IDs ”2” and ”3”, the HTTP GET request would be:

```
GET /characteristics?id=2.8,3.8 HTTP/1.1
```

If all reads succeed, the accessory must respond with a `200 OK` HTTP Status Code and a JSON body. The body must contain a JSON object with the value and instance ID of each characteristic. In the example below, the value of both characteristics is 'false':

```
HTTP/1.1 200 OK
Content-Type: application/hap+json
Content-Length: <length>

{
    ”characteristics” : [
        {
            "aid" : 2,
            "iid" : 8,
            "value" : false,
        },
        {
            "aid" : 3,
            "iid" : 8,
            "value" : false,
        }
    ]
}
```

If an error occurs when attempting to read any characteristics, e.g. the physical devices represented by the characteristics to be read were unreachable, the accessory must respond with a `207 Multi-Status` HTTP Status Code and each characteristic object must contain a `"status"` entry. Characteristics that were read successfully must have a `"status"` of `0` and `"value"`. Characteristics that were read unsuccessfully must contain a non-zero `"status"` entry and must not contain a `"value"` entry. An example is listed below where one read failed and one succeeded:

```
HTTP/1.1 207 Multi-Status
Content-Type: application/hap+json
Content-Length: <length>

{
    "characteristics" : [
        {
            "aid" : 2,
            "iid" : 8,
            "status" : 0,
            "value" : false
        },
        {
            "aid" : 3,
            "iid" : 8,
            "status" : -70402
        }
    ]
}
```

6.7.5 Control Point Characteristics

A control-point characteristic is a write-only characteristic used to request the accessory to perform actions such as execute commands or perform operations. The context of the request is passed as the value of the HTTP PUT request.

The value is defined by the vendor or Apple Accessory profile.

The identify characteristic is an example of a control-point characteristic.

6.7.6 Identify Routine

An identify routine must exist for all accessories. There are two ways that the identify routine may be invoked. Both of these methods must be supported:

- Via an Identify URL if the accessory is unpaired, as described in "6.7.7 Identify HTTP URL" (page 90).
- Via a characteristic write to the identify characteristic, public.hap.characteristic.identify.

The identify routine is used to locate the accessory. The routine should run no longer than five seconds.

For example, a lightbulb accessory may implement an identify routine as follows:

Blink the lightbulb (turn on to full brightness, then off) three times for 500 ms each time

6.7.7 Identify HTTP URL

The identify URL resides at `/identify`. A controller sends an HTTP POST request with an empty body to /identify on the accessory to cause it to run its identify routine:

```
POST /identify HTTP/1.1
```

The URL is only valid if the accessory is unpaired, i.e. it has no paired controllers. If the accessory has paired controllers then the accessory must return `400 Bad Request` for any HTTP requests to the /identify URL:

```
HTTP/1.1 400 Bad Request
Content-Type: application/hap+json
Content-Length: <length>

{
    "status" : -70401
}
```

Otherwise, for an unpaired accessory, it will fulfill the identify request and return a `204 No Content` HTTP status code:

```
HTTP/1.1 204 No Content
```

6.8 Notifications

Notifications allow an accessory to report state changes to a controller. Notifications must be explicitly configured for each characteristic. Network-based notifications must be coalesced by the accessory using a delay of no less than 1 second. The exception to this rule includes notifications for the following characteristics which must be delivered immediately.

- ”10.123 Button Event” (page 249)
- ”10.53 Programmable Switch Event” (page 201)

Additional restrictions, based on the notification method, are described below. Accessories must strictly follow these notification policies. Excessive or inappropriate notifications may result in the user being notified of a misbehaving accessory and/or termination of the pairing relationship. The following describes each notification method:

6.8.1 Accessory Updates State Number Internally

This internally tracks the state number within the accessory and provides it via an explicit request from the controller, but doesn’t send notifications. This method avoids multicast network traffic on each state change, but is the least responsive. This is the default unless the accessory has been configured by a controller to use a different method.

6.8.2 Accessory Sends Events to Controller

The accessory delivers notifications by sending an event message, which is an unsolicited HTTP response, over the TCP connection established by the controller. An event message has the same format as an HTTP response, but uses an protocol version of `EVENT/1.0`. Event messages must never interrupt a response that is already being sent. A response message must also never interrupt an event message. Either of these violations would result in the controller receiving a corrupt message. Event messages do not receive an HTTP response from the controller. Indications shall not be sent to the controller that changed the value of a characteristic if the change was due to a write from the controller.

6.8.2.1 Event Example

An example of bidirectional communication is when a controller writes the value of a ”target” characteristic, such as the ”target temperature” of a thermostat, and wants to be notified of changes in the ”current temperature” characteristic. The following steps outline how notifications are registered and delivered:

1. The controller registers for notifications against the ”current temperature” characteristic, which has an instance ID of 4, on an Accessory object with an instance ID of ”1”, by sending the following request to the Accessory Server:

```
PUT /characteristics HTTP/1.1
Host: thermostat.local:12345
Content-Type: application/hap+json
Content-Length: <length>

{
    ”characteristics” : [
        {
            ”aid” : 1,
            ”iid” : 4,
            ”ev” : true
        }
    ]
}
```

2. If the characteristic supports notifications then the accessory must respond with a `204 No Content` HTTP status code and an empty body. If the characteristic does not support notifications then the accessory must respond with an HTTP `207 Multi-Status` status code and a ”status” with code -70406:

```
HTTP/1.1 207 Multi-Status
Content-Type: application/hap+json
Content-Length: <length>

{
    "characteristics" : [
        {
            "aid" : 1,
            "iid" : 4,
            "status" : -70406
        }
    ]
}
```

3. When the value of the ”current temperature” value changes, the accessory sends the following unsolicited message to the controller:

```
EVENT/1.0 200 OK
Content-Type: application/hap+json
Content-Length: <length>

{
    "characteristics" : [
        {
            "aid" : 1,
            "iid" : 4,
            "value" : 23.0
        }
    ]
}
```

6.8.2.2 Event Policy

- Delivering notifications to controllers requires the controller to establish an HAP session with the accessory.
- The notification registration state of a characteristic must not persist across sessions.
- When a new HAP session is established the notification registration state of that controller must be ’false’ for all characteristics provided by the accessory.
- A write-only characteristic (i.e. the characteristic permissions only include Paired Write ”pw”) must not support event notifications.
- The accessory must support registering for notifications against multiple characteristics in a single request.

- The accessory should coalesce notifications whenever possible.
- The accessory must only deliver notifications to the controller for characteristics that the controller has registered to receive notifications against.
- At any point the controller may deregister for event notifications against a characteristic by setting the "ev" key to "false." The accessory must stop delivering notifications for the deregistered characteristic immediately after receiving the deregister request from the controller.

6.9 HTTP URLs

This section summarizes the HTTP URLs used in HAP.

Table 6-14: HTTP URLs

URL Path	HTTP Method	Description
/accessories	GET	Retrieve the accessory attribute database from the accessory. Only valid from paired controllers. See "6.6 IP Accessory Attribute Database" (page 74).
/characteristics	GET	Reads characteristic data. See "6.7.4 Reading Characteristics" (page 87).
/characteristics	PUT	Writes characteristic data. See "6.7.2 Writing Characteristics" (page 81).
/identify	POST	Request the accessory to run its identify routine. Only valid if the accessory is unpaired. See "6.7.7 Identify HTTP URL" (page 90).
/pair-setup	POST	Write a pair-setup request. See "5.6 Pair Setup" (page 41).
/pair-verify	POST	Write a pair-verify request. See "5.7 Pair Verify" (page 49).
/pairings	POST	Add, remove, or list pairings. See "5.10 Add Pairing" (page 52), "5.11 Remove Pairing" (page 53), and "5.12 List Pairings" (page 54).
/prepare	PUT	Prepare Write Request. See "6.7.2.4 Timed Write Procedures" (page 84).
/secure-message	POST	Used to exchange HAP PDU messages after a secure session is established. "5.15 Software Authentication Procedure" (page 58)
/resource	POST	Request the accessory to run the resource routine (e.g a snapshot is returned in response for an IP camera accessory). See "12.5 Image Snapshot" (page 261).

6.10 HomeKit Data Stream

This section specifies the requirements for HomeKit Data Stream (HDS). The HomeKit Data Stream protocol operates over a reliable byte stream (TCP), and uses an ecryption key negotiated out-of-band. All messages are encrypted and authenticated.

6.10.1 Data Stream Transport Security

For each new Data Stream Transport session, both the controller and accessory shall derive the read and write transport encryption keys using the following parameters, using the key salt values negotiated during the transport setup procedure.

```
AccessoryToControllerBulkTransferEncryptionKey = HKDF-SHA-512 of
InputKey = <Current HAP session shared secret>
Salt = <ControllerKeySalt><AccessoryKeySalt>
Info = ”HDS-Read-Encryption-Key”
OutputSize = 32 bytes

ControllerToAccessoryBulkTransferEncryptionKey = HKDF-SHA-512 of
InputKey = <Current HAP session shared secret>
Salt = <ControllerKeySalt><AccessoryKeySalt>
Info = ”HDS-Write-Encryption-Key”
OutputSize = 32 bytes
```

Each message shall be secured with the AEAD algorithm

```
 AEAD_CHACHA20_POLY1305
```

as specified in Section 2.8 of RFC 7539.

6.10.2 Frame Format

Data sent over any HDS supports encryption. The header contains a length to indicate the number of payload bytes and a type to indicate the structure of the payload. The type and length are authenticated, but unencrypted. The payload is encrypted and authenticated. The auth tag is unencrypted. Frames are encrypted and authenticated using the AEAD algorithm.

```
0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+-----------------------------------------------+
|   Type (1)    |                  Length (3)                   | 4 bytes
+---------------+-----------------------------------------------+
\                                                               /
|                           Payload                             | <Length> bytes
/                                                               \
+---------------------------------------------------------------+
|                                                               |
|                         authTag (16)                         | 16 bytes
|                                                               |
+---------------------------------------------------------------+
```

Type = 0x01

Length = Length of payload in bytes (big endian). The length only counts for the length of the `payload` field. It does not count the fixed-size `type/length/authTag` fields. In order to improve packet latency, the length field must be between 0 and 1,048,575 bytes.

Payload contains the header and the message described below. AuthTag is used to authenticate the message. Type, Length, and Payload fields are authenticated.

6.10.3 Payload Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+
|   HeaderLen   | 1 byte
+---------------+-----------------------------------------------+
\                                                               /
|                         Header                                | <HeaderLen> bytes
/                                                               \
+---------------------------------------------------------------+
\                                                               /
|                         Message                               | <MessageLen> bytes
/                                                               \
+---------------------------------------------------------------+
```

```
MessageLen = ( Length - HeaderLen ) - 1
```

The payload has two parts: the header and the message. The header is a dictionary of the fields needed for describing the message. The message is a dictionary that contains the fields particular to that event, request, or response. Data format from "6.10.4 Data Format" (page 97) is used to encode the header and the message individually.

6.10.3.1 Messages

There are four types of messages possible in the Home Data Stream:

1. Event - a one-way message that does not require any response
2. Request - a message that requires a response or acknowledgement
3. Response - the reply to a request
4. Control - Hello message as first message for the stream

6.10.3.1.1 Event Messages

An event message is a one-way message that does not require any response. It can be sent by either the accessory or the controller.

An event message contains two required keys in the header:

Table 6-15: Properties of Event Messages

Key	**Type**	**Description**
`protocol`	string	The name of the protocol that should handle this message
`event`	string	The topic of this message

6.10.3.2 Request Messages

A request message is a two-way message that requires a response. It can be sent by either the accessory or the controller.

A request message contains three required keys in the header:

Table 6-16: Properties of Request Messages

Key	**Type**	**Description**
`protocol`	string	The name of the protocol that should handle this message
`request`	string	The topic of this message
`id`	int64	A unique identifier to use to match to the response

The sender must choose an identifier that is unambiguous and helps match up the corresponding response when it is received. Once the sender receives a response for that identifier, it may use that identifier in a later request, but it must make sure that there are never two requests in flight that use the same identifier. It is possible for the Accessory and Controller to independently create requests that happen to use the same identifier. This is permitted as the identifier only has a unique meaning to its sender.

Each request has a 10-second response timeout. If the sender does not receive a response to the request within this timeout, it shall close the Data Stream connection.

If the receiver is unable to handle the request due to resource constraints (eg, low memory), the receiver must decode the header of the request and send a response with an `out of resource` error. The receiver is not required to inspect the message dictionary (but should still perform authentication of the entire payload).

6.10.3.2.1 Response Messages

After receiving a request, the receiver must reply with a response message.

A response message contains four required keys in the header:

Table 6-17: Properties of Response Messages

Key	Type	Description
`protocol`	string	The name of the protocol that should handle this message
`response`	string	The topic of this message
`id`	int64	A unique identifier to use to match to the response
`status`	int64	A status code for success or error reason

If the response is a success, then the message can contain relevant information for that topic.

6.10.3.2.2 Hello

The controller sends a hello request as the first message on the Data Stream. The header should use `control` as the protocol and `hello` as the topic. There are no fields in the message of the request or the response.

6.10.4 Data Format

Data is encoded using a tag byte to indicate the type of object and additional info. Objects that don't fit entirely in the tag byte use subequent bytes. The following table below shows the encodings for objects. Integers and floating point values use little endian encoding. Float32 is an IEEE single precision value. Float64 is an IEEE double precision value.

Table 6-18: Properties of Data Format Table 1

Value	Format	Description
0x00	Invalid	Reserved to avoid common errors with zero'd memory.
0x01	Boolean	True value.
0x02	Boolean	False value.
0x03	n/a	Terminator for variable-item objects.
0x04	Null	Null object.
0x05	UUID	<16:big endian UUID>.
0x06	Date	<8:little endian Float64 seconds since 2001-01-01 00:00:00>.
0x07	Number	Integer -1.
0x08-0x2E	Number	Integers 0-38 (value = tag - 0x08).
0x02F	Number	Integer 39.
0x30	Number	8-bit signed integer. Range: -128 to 127.
0x31	Number	16-bit, little endian, signed integer. Range: -32768 to 32767.
0x32	Number	32-bit, little endian, signed integer, Range: -2147483648 to 2147483647.
0x33	Number	64-bit, little endian, signed integer, Range: -9223372036854775807 to 9223372036854775807.
0x35	Number	Little endian Float32.
0x36	Number	Little endian Float64.
0x40-0x60	String	0-32 bytes of UTF-8 (length = tag - 0x40). Not NUL terminated.
0x61	String	<1:length> <n:UTF-8 bytes>. Not NUL terminated.
0x62	String	<2:little endian length> <n:UTF-8 bytes>. Not NUL terminated.
0x63	String	<4:little endian length> <n:UTF-8 bytes>. Not NUL terminated.
0x64	String	<8:little endian length> <n:UTF-8 bytes>. Not NUL terminated.
0x6F	String	<n:UTF-8 bytes>. <1:NUL>.
0x70-0x90	Data	0-32 bytes of data (length = tag - 0x70).
0x91	Data	<1:length><n:bytes>.
0x92	Data	<2:little endian length><n:bytes>.
0x93	Data	<4:little endian length><n:bytes>.
0x94	Data	<8:little endian length><n:bytes>.
0x9F	Data	[data object 1][data object 2]...[data object n]<1:terminator tag>.

Table 6-19: Properties of Data Format Table 2

Value	Format	Description
0xA0-0xCF	Data	Reserved
0xD0-0xDE	Array	0-12 items (count = tag - 0D0). <object 1><object 2>...<object n>.
0xDF	Array	[object 1][object 2]...[object n]<1:terminator tag>.
0xE0-0xEE	Dictionary	0-14 entries (count = tag - 0E0). <key 1><value 1><key 2><value 2>...<key n><value n>.
0xEF	Array	[key 1][value 1][key 2][value 2]...[key n][value n]<1:terminator tag>.

6.11 Testing IP Accessories

This section outlines conformance tests for accessories.

1. A service must not contain characteristics with duplicate instance IDs.
2. A service must contain at least one characteristic.
3. An accessory object must not contain services with duplicate instance IDs.
4. An accessory object must contain at least one service.
5. A bridge must not contain accessory objects with duplicate instance IDs.
6. An accessory object must contain only one accessory information service.
7. An accessory information service must have instance ID = 1.
8. A service type must be a UUID.
9. A characteristic type must be a UUID.
10. An Apple-defined characteristic type must contain all or a subset of the properties defined in "10 Apple-defined Characteristics" (page 177). It also must not have extra permissions.
11. An Apple-defined service type must contain all of its mandatory characteristics.
12. Instance IDs must be >= 1.
13. Instance IDs must be an integer.
14. A service must not have characteristics with duplicate characteristic types.
15. A service must not have more than 100 characteristics.
16. An accessory object must not have more than 100 services.
17. A bridge must not have more than 100 accessory objects.

18. Value of c# in Bonjour TXT record must be within range of [1-4294967295].

19. If characteristic properties includes a value for maxLen, the value must be not be > 256.

20. The value of the characteristic (if it supports Paired Read) must be valid according to the specified format, and metadata, as applicable. For example, if the minValue/maxValue metadata are 10, and 50, then the value should not be 60.

7 HomeKit Accessory Protocol for Bluetooth LE Accessories

7.1 Overview

This chapter describes the HomeKit Accessory Protocol (HAP) for Bluetooth LE accessories. Bluetooth LE Accessories confirming to this version are referred to as HAP-BLE 2.0 accessories. Services and characteristics supported by accessories are represented using standard GATT.

7.2 Accessory Requirements

1. Must be compatible with only one (current) protocol version of the HAP-BLE specification.
2. Must conform to at a minimum to Core Bluetooth specification 4.2 (Accessories should confirm to Bluetooth 5.0 or later for future compatibility)
3. Must not allow the firmware to be downgraded.
4. Must support a timer with a minimum resolution of at least 100 milliseconds.
5. Must implement a security session idle timeout and terminate the security session (and the Bluetooth link) after 30 seconds of inactivity (i.e without any HAP Transactions).
6. Must support refresh of the security session when a controller re-pair verifies with the accessory.

7.3 HAP for Bluetooth LE

7.3.1 HAP Transactions and Procedures

A HAP transaction is set of related message exchanges between the controller and the accessory that perform a single HAP function. In HAP a request-response pair is considered a single transaction. The HAP PDUs that form a single transaction have the same transaction ID. A transaction allows the controller to validate the receipt of all HAP requests to the accessory by receiving an authenticated response from the accessory. The accessory can validate the sender and targeted characteristic or service for all requests.

A connected set of transactions that achieve a particular purpose form a HAP procedure. A procedure shall be determined to be completed successfully only after all the transactions as specified in the respective procedure has completed successfully. The iOS controller will complete all procedures in a time bound fashion in order to provide a deterministic response to HAP clients. Any procedure that times out or fails validation of the response shall result in the current HAP secure session being invalidated and a new session may be established by the controller. A failure to re-establish the HAP secure session within 10 seconds must result in both the accessory and the iOS controller dropping the Bluetooth link.

There shall be only one outstanding procedure on a characteristic at a time. Accessories should be able to support procedures on different characteristics in parallel.

An accessory must cancel any pending procedures when a new HAP secure session starts getting established.

7.3.2 Attribute Database Authentication

To validate the authenticity of the GATT database (i.e., types, instance IDs of services and characteristics and all characteristic metadata descriptors) each HAP characteristic must support a signature read procedure. The signature read response must contains the following tuple:

```
< characteristic type, service type, service instance id,
[characteristic metadata descriptors] >
```

The signature read procedure must be supported with and without a secure session. When a secure session is established the procedure is encrypted and authenticated. Without a secure session the procedure is in the clear.

After an initial pair-verify and every configuration change, an iOS controller will perform a secured read of the characteristic signature and validate the service / characteristic types, and instance IDs confirming the unsecured GATT database. The authenticated characteristic metadata read from the signature read response is used for all HAP procedures on the characteristic.

Accessories must return characteristic metadata only via the signature read procedure and must not publish the metadata descriptors in the Bluetooth LE GATT database.

7.3.3 HAP PDU Format

The HAP PDUs have a PDU header follow by an optional PDU body. All parameters in the HAP-BLE PDU have a little-endian format, (i.e. the least significant byte is transferred first) unless specified otherwise. HAP PDU format is specified in Table 7-1 (page 102).

Table 7-1: HAP PDU Format

PDU Header		PDU Body(Optional)	
Control Field (1 Byte)	PDU Fixed Params	Body Length (2 Bytes)	Additional Params and Values in TLV8s (1-n Bytes)

7.3.3.1 HAP PDU Header - Control Field

The first byte in the HAP PDU header is a control field that defines how the PDU and the rest of the bytes in the PDU are interpreted, see Table 7-2 (page 102).

Table 7-2: HAP PDU Header - Control Field

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
X	R	R	X	X	X	X	0

The control field bit 7 indicates the fragmentation status of the HAP-BLE PDU.

Table 7-3: Control Field Bit 7 Values

Bit 7	Description
0	First Fragment (or no fragmentation)
1	Continuation of fragmented PDU

The control field bit 4 indicates instance id size in the HAP-BLE PDU.

Table 7-4: Control Field Bit 4 Values

Bit 4	Description
0	16 bit IIDs (or IID = 0)
1	64-bit IIDs

The control field Bit 1-3 indicates PDU types (i.e. Request or Response). All other values in the PDU type are reserved.

Table 7-5: Control Field Bit 1-3 Values

Bit 3	Bit 2	Bit 1	PDU Type
0	0	0	Request
0	0	1	Response

Control field bit 0 is reserved for extending the control byte. It shall be set to 0 in this version of HAP-BLE.

Table 7-6: Control Field Bit 0 Values

Bit 0	Length Extension
0	1 Byte Control Field
1	Reserved

Bits 4, 5 and 6 in the control field are reserved and shall be set to 0 in this version of the HAP-BLE specification.

7.3.3.2 HAP Request Format

The HAP Request format is shown in Table 7-7 (page 103).

Table 7-7: HAP-BLE Request Format

PDU Header				PDU Body (Optional)	
Control Field (1 Byte)	HAP Opcode (1 Byte)	TID (1 Byte)	CharID / SvcID (2 Bytes)	Body Length (2 Bytes)	Additional Params and Value in TLV8s (1-n Bytes)

- HAP Opcode: The second byte in the HAP-BLE PDU Request header is the HAP Opcode field. It indicates the opcode for the HAP Request PDU. If an accessory receives a HAP PDU with an opcode that it does not support it shall reject the PDU and respond with a status code `Unsupported PDU` as defined in Table 7-37 (page 123) in its HAP response. The supported HAP opcodes are listed in Table 7-8 (page 104).
- TID : Transaction Identifier is an 8 bit number identifying the transaction number of this PDU. The TID is randomly generated by the originator of the request and is used to match a request/response pair.
- CharID / SvcID: Characteristic / Service Instance Identifier is the instance id of the characteristic / service for a particular request.

Table 7-8: HAP Opcode Description

HAP Opcode	Description
0x01	HAP-Characteristic-Signature-Read
0x02	HAP-Characteristic-Write
0x03	HAP-Characteristic-Read
0x04	HAP-Characteristic-Timed-Write
0x05	HAP-Characteristic-Execute-Write
0x06	HAP-Service-Signature-Read
0x07	HAP-Characteristic-Configuration
0x08	HAP-Protocol-Configuration

7.3.3.3 HAP Response Format

The HAP Response format is shown in Table 7-9 (page 104).

Table 7-9: HAP Response Format

PDU Header			PDU Body (Optional)	
Control Field (1 Byte)	TID (1 Byte)	Status (1 Byte)	Body Length (2 Bytes)	Additional Params and Value in TLV8s (1-n Bytes)

- TID: The second byte in the HAP PDU Response is the transaction identifier (TID). The transaction identifier must be set to the same value as the TID of the request.
- Status: The status code indicating success or an appropriate reason for failure (see Table 7-37 (page 123)).

7.3.3.4 HAP PDU Body

Some HAP PDUs may include a body. The first 2 bytes of the PDU body is the length field that indicates the total length of the PDU body. Following the length field is a list of TLV parameters that includes the value and additional parameters associated with the respective request or response.

All additional parameters in the request and response are encoded as individual TLV8 entries. An accessory must parse the TLV8 entries and ignore parameters that it does not support.

The additional parameters allows for future extensibility of commands by adding new meaning on how the request or response should be processed by the receiver.

Table 7-10: Additional Parameter Types Description

Additional Parameter Types	Description
0x01	HAP-Param-Value
0x02	HAP-Param-Additional-Authorization-Data
0x03	HAP-Param-Origin (local vs remote)
0x04	HAP-Param-Characteristic-Type
0x05	HAP-Param-Characteristic-Instance-ID
0x06	HAP-Param-Service-Type
0x07	HAP-Param-Service-Instance-ID
0x08	HAP-Param-TTL
0x09	HAP-Param-Return-Response
0x0A	HAP-Param-HAP-Characteristic-Properties-Descriptor
0x0B	HAP-Param-GATT-User-Description-Descriptor
0x0C	HAP-Param-GATT-Presentation-Format-Descriptor
0x0D	HAP-Param-GATT-Valid-Range
0x0E	HAP-Param-HAP-Step-Value-Descriptor
0x0F	HAP-Param-HAP-Service-Properties
0x10	HAP-Param-HAP-Linked-Services
0x11	HAP-Param-HAP-Valid-Values-Descriptor
0x12	HAP-Param-HAP-Valid-Values-Range-Descriptor

7.3.3.5 HAP PDU Fragmentation Scheme

HAP PDU fragments (excluding the first fragment) shall use the format as specified in Table 7-11 (page 106).

Table 7-11: HAP PDU Fragmentation Scheme

PDU Header		Continuation of PDU Body
Control Field (1 Byte)	TID (1 Byte)	
0b 1000 XXX0	0xXX	

Only the control byte and the TID bytes shall be included in the continuation fragment's PDU header. The bit 7 in the control byte shall be set to 1 indicating a continuation of fragment. All fragments shall have bits 1-3 (Request / Response type) set to the same value as it was in the first fragment. The TID must also be set to the same value as it was in the first fragment. Each fragment is encrypted independently when a secure session is established and when the characteristic requires a secure read or write.

The HAP PDU must be fragmented to fit the underlying protocols MTU size. In the case of HAP-BLE over GATT the maximum size of a fragment shall be the GATT MTU of 512 bytes. When secure reads or writes are performed and the HAP PDU requires fragmentation, the first HAP PDU shall have a total length of up to 496 bytes (512 - 16 byte auth tag). Subsequent fragments can be up to 496 bytes.

Individual fragments are allowed to be less than the max fragment size and for optimal performance the fragment size will be chosen to fit within an ATT_MTU when appropriate.

The first fragment must contain the complete PDU Header and the Body Length when the optional PDU body is present.

7.3.4 HAP-BLE PDU Payloads

7.3.4.1 HAP-Characteristic-Signature-Read-Request

Table 7-12: HAP-Characteristic-Signature-Read-Request

PDU Header			
Control Field (1 Byte)	HAP PDU Type (1 Byte)	TID (1 Byte)	Characteristics Instance ID CharID (2 Bytes)
0b 0000 0000	0x01	0xXX	0xXXXX

7.3.4.2 HAP-Characteristic-Signature-Read-Response

Table 7-13: HAP-Characteristic-Signature-Read-Response 1

PDU Header			PDU Body			
Control Field (1 Byte)	TID (1 Byte)	Status (1 Byte)	Body Length (2 Bytes)	TLV (Chr Type)	TLV (Svc ID)	TLV (SvcType)
0b 0000 0010	0xXX	0xXX	0xXXXX	<0x04,0x10, 128-bit UUID>	<0x07, 0x02, 16-bit SvcID>	<0x06,0x10, 128-bit UUID>

Table 7-14: HAP-Characteristic-Signature-Read-Response 2

PDU Body (continued)				
TLV (HAP-Chr-Properties)	TLV (Optional) (GATT-User-Description)	TLV (GATT-Format)	TLV (Optional) (GATT-Valid-Range)	TLV (Optional) (HAP-Step-Value)
<0x0A, 0x02, HAP-Chr-Properties>	<0x0B, 0xXX, UTF-8 user description string>	<0x0C, 0x07, GATT-Format>	<0x0D, 0xXX, GATT-Valid-Range>	<0x0E, 0xXX, HAP-Step-Value>

7.3.4.3 HAP-Characteristic-Signature-Read-Response (with Valid Values)

Table 7-15: HAP-Characteristic-Signature-Read-Response (with Valid Values) 1

PDU Header			PDU Body			
Control Field (1 Byte)	TID (1 Byte)	Status (1 Byte)	Body Length (2 Bytes)	TLV (Chr Type)	TLV (Svc ID)	TLV (SvcType)
0b 0000 0010	0xXX	0xXX	0xXXXX	<0x08,0x10, 128-bit UUID>	<0x0A, 0x02, 16-bit SvcID>	<0x09,0x10, 128-bit UUID>

Table 7-16: HAP-Characteristic-Signature-Read-Response (with Valid Values) 2

PDU Body (continued)			
TLV (HAP-Chr-Properties)	TLV (Optional Descriptors)	TLV (Optional) (HAP-Valid-Values)	TLV (Optional) (HAP-Valid-Values-Range)
<0x0A, 0x02, HAP-Chr-Properties>	<...>	<0x11, 0xXX, 0xAA, 0xBB, 0xCC,...>	<0x12, 0xXX, 0xS1, 0xE1,...0xSN, 0xEN>

7.3.4.4 HAP-Characteristic-Write-Request

Table 7-17: HAP-Characteristic-Write-Request

PDU Header				PDU Body	
Control Field (1 Byte)	HAP PDU Type (1 Byte)	TID (1 Byte)	CharID (2 Bytes)	Body Length (2 Bytes)	TLV (Char Value)
0b 0000 0000	0x02	0xXX	0xXXXX	0xXXXX	<0x01,0xXX,value>

7.3.4.5 HAP-Characteristic-Write-Response

Table 7-18: HAP-Characteristic-Write-Response

PDU Header		
Control Field (1 Byte)	TID (1 Byte)	Status (1 Byte)
0b 0000 0010	0xXX	0xXX

7.3.4.6 HAP-Characteristic-Read-Request

Table 7-19: HAP-Characteristic-Read-Request

PDU Header			
Control Field (1 Byte)	HAP PDU Type (1 Byte)	TID (1 Byte)	Characteristics Instance ID CharID (2 Bytes)
0b 0000 0000	0x03	0xXX	0xXXXX

7.3.4.7 HAP-Characteristic-Read-Response

Table 7-20: HAP-Characteristic-Read-Response

PDU Header			PDU Body	
Control Field (1 Byte)	TID (1 Byte)	Status (1 Byte)	Body Length (2 Bytes)	TLV (Char Value)
0b 0000 0010	0xXX	0xXX	0xXXXX	<0x01,0xXX,value>

7.3.4.8 HAP-Characteristic-Timed-Write-Request

Table 7-21: HAP-Characteristic-Timed-Write-Request

PDU Header				PDU Body		
Control Field (1 Byte)	HAP PDU Type (1 Byte)	TID (1 Byte)	CharID (2 Bytes)	Body Length (2 Bytes)	TLV (TTL*100ms) (1 Byte)	TLV (Char Value)
0b 0000 0000	0x04	0xXX	0xXXXX	0xXXXX	<0x08,0x01, 0xXX>	<0x01, 0xXX, value>

7.3.4.9 HAP-Characteristic-Timed-Write-Response

Table 7-22: HAP-Characteristic-Timed-Write-Response

PDU Header		
Control Field (1 Byte)	TID (1 Byte)	Status (1 Byte)
0b 0000 0010	0xXX	0xXX

7.3.4.10 HAP-Characteristic-Execute-Write-Request

Table 7-23: HAP-Characteristic-Execute-Write-Request

PDU Header			
Control Field (1 Byte)	HAP PDU Type (1 Byte)	TID (1 Byte)	Characteristics Instance ID CharID (2 Bytes)
0b 0000 0000	0x05	0xXX	0xXXXX

7.3.4.11 HAP-Characteristic-Execute-Write-Response

Table 7-24: HAP-Characteristic-Execute-Write-Response

PDU Header		
Control Field (1 Byte)	TID (1 Byte)	Status (1 Byte)
0b 0000 0010	0xXX	0xXX

7.3.4.12 HAP-Service-Signature-Read-Request

Table 7-25: HAP-Service-Signature-Read-Request

PDU Header			
Control Field (1 Byte)	HAP PDU Type (1 Byte)	TID (1 Byte)	Service Instance ID (SvcID) (2 Bytes)
0b 0000 0000	0x06	0xXX	0xXXXX

7.3.4.13 HAP-Service-Signature-Read-Response

Table 7-26: HAP-Service-Signature-Read-Response

PDU Header			PDU Body		
Control Field (1 Byte)	TID (1 Byte)	Status (1 Byte)	Body Length (2 Bytes)	TLV (Svc Properties)	TLV (Linked Svc)
0b 0000 0010	0xXX	0xXX	0xXXXX	<0x0F,0x02, 0xXXXX>	<0x10,0xXX, SvcID1, SvcID2, ..., SvcIDn>

If the accessory receives an invalid (eg., 0) Service Instance ID in the HAP-Service-Signature-Read-Request, it must respond with a valid HAP-Service-Signature-Read-Response with Svc Properties set to 0 and Linked Svc (if applicable) set to 0 length.

7.3.4.14 HAP-Characteristic-Configuration-Request

Table 7-27: HAP-Characteristic-Configuration-Request

PDU Header				PDU Body	
Control Field (1 Byte)	HAP Opcode (1 Byte)	TID (1 Byte)	CharID (2 Bytes)	Body length (2 Bytes)	Configuration Parameters Optional TLVs
0b 0000 0000	0x07	0xXX	0xXXXX	0xXXXX	<0x01,0x02,0x01,0x00> <0x02,0x01,0x01>

Table 7-28: Characteristic configuration parameter types

Characteristic Configuration Parameter Type	Description
0x01	HAP-Characteristic-Configuration-Param-Properties
0x02	HAP-Characteristic-Configuration-Param-Broadcast-Interval

Table 7-29: Characteristic configuration properties

Bitmask	Description
0x0001	Enable/Disable Broadcast Notification
0x0002 - 0xFFFF	Reserved

Table 7-30: Broadcast Interval

Value	Description
0x01	20 ms (Default)
0x02	1280 ms
0x03	2560 ms
0x04-0xFF	Reserved

7.3.4.15 HAP-Characteristic-Configuration-Response

Table 7-31: HAP-Characteristic-Configuration-Response

PDU Header			PDU Body	
Control Field (1 Byte)	TID (1 Byte)	Status (1 Byte)	Body length (2 Bytes)	Configuration Parameters Optional TLVs
0b 0000 0010	0xXX	0xXX	0xXXXX	<0x01,0x02,0x01,0x00> <0x02,0x01,0x01>

The accessory must include all parameters in the response even if the default Broadcast Interval is used.

7.3.4.16 HAP-Protocol-Configuration-Request

Table 7-32: Protocol Configuration Request Types

Protocol Configuration Request Type	Description
0x01	Generate-Broadcast-Encryption-Key
0x02	Get-All-Params
0x03	Set-Accessory-Advertising-Identifier

Table 7-33: HAP-Protocol-Configuration-Request

PDU Header				PDU Body	
Control Field (1 Byte)	HAP Opcode (1 Byte)	TID (1 Byte)	Service Instance ID SvcID (2 Bytes)	Body length (2 Bytes)	Configuration Parameters Optional TLVs
0b 0000 0000	0x08	0xXX	0xXXXX	0xXXXX	<0x01,0x00> <0x02,0x00>

7.3.4.17 HAP-Protocol-Configuration-Response

Table 7-34: Protocol Configuration Parameter Types

Protocol Configuration Parameter Type	Description
0x01	HAP-Param-Current-State-Number
0x02	HAP-Param-Current-Config-Number
0x03	HAP-Param-Accessory-Advertising-Identifier
0x04	HAP-Param-Broadcast-Encryption-Key

Table 7-35: HAP-Protocol-Configuration-Response 1

PDU Header			PDU Body		
Control Field (1 Byte)	TID (1 Byte)	Status (1 Byte)	Body Length (2 Bytes)	TLV (Optional State Number)	TLV (Optional Configuration Number)
0b 0000 0010	0xXX	0xXX	0xXXXX	<0x01,0x02, 0xXX XX>	<0x02, 0x01, 0xXX>

Table 7-36: HAP-Protocol-Configuration-Response 2

PDU Body (continued)	
Optional TLV (Advertising Identifier)	Optional TLV (Broadcast Encryption Key)
<0x03, 0x06, 0xXX XX XX XX XX XX>	<0x04, 0x20, 32-byte broadcast encryption key>

7.3.5 HAP Procedures

7.3.5.1 HAP Characteristic Signature Read Procedure

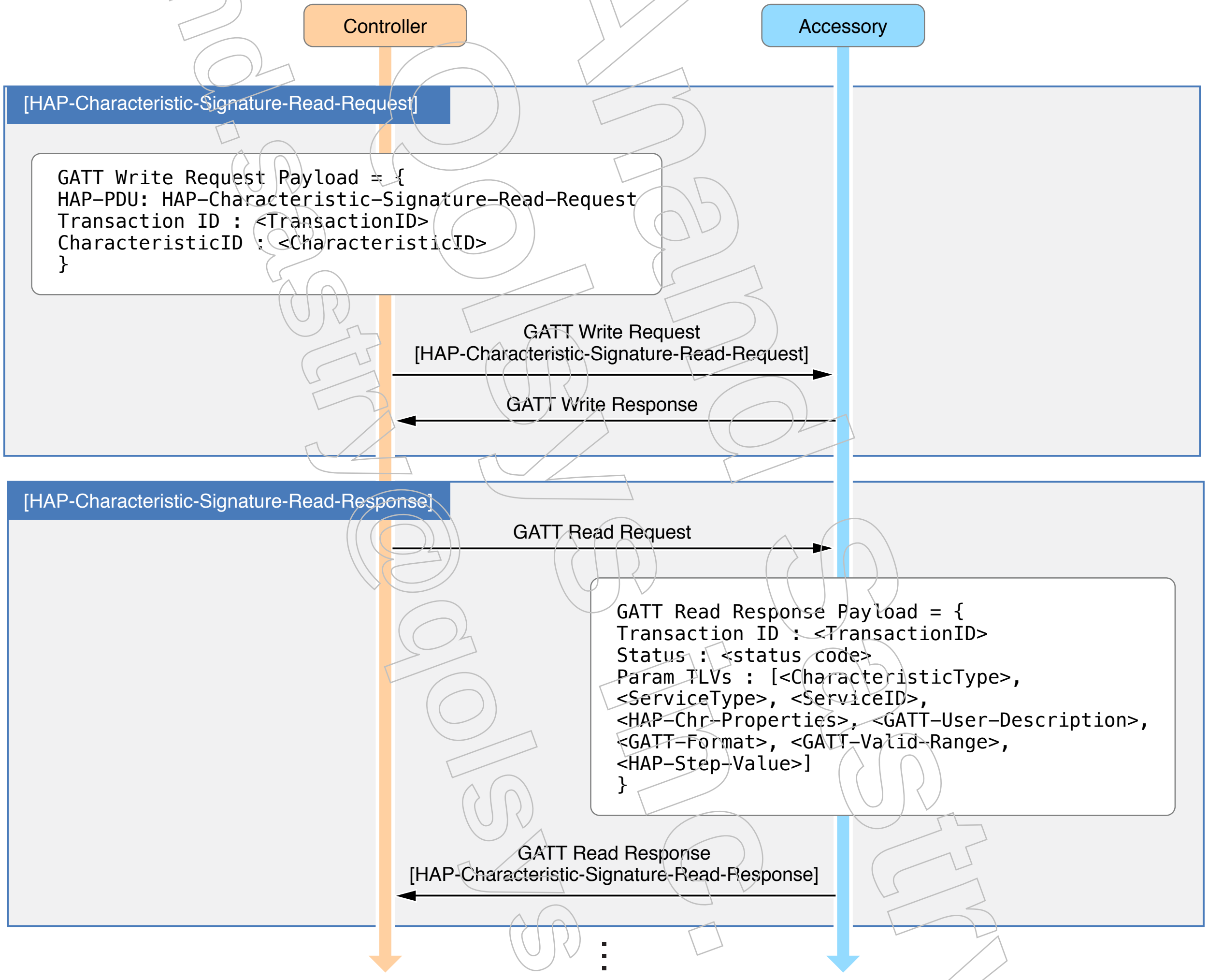
This procedure is used to read the signature of each HAP characteristic. Each HAP characteristic must support this procedure and return the following information:

- Characteristic Instance ID
- Characteristic Type
- Service Instance ID of the service that the characteristic belongs
- Service Type of the service that the characteristic belongs
- All the metadata associated with the characteristic.

The signature read procedure must be supported with and without a secure session. When a secure session is established the procedure is encrypted and authenticated. Without a secure session the procedure is in the clear.

The "7.4.4.3 Service Instance ID" (page 135) characteristic is not considered a HAP characteristic and does not support this procedure.

The characteristics "5.13.1.1 Pair Setup" (page 56), "5.13.1.2 Pair Verify" (page 56) and "5.13.1.3 Pairing Features" (page 57) of "5.13.1 Pairing Service" (page 56) do not support `Paired Read` and `Paired Write` and only support the "7.3.5.1 HAP Characteristic Signature Read Procedure" (page 112) without a secure session.

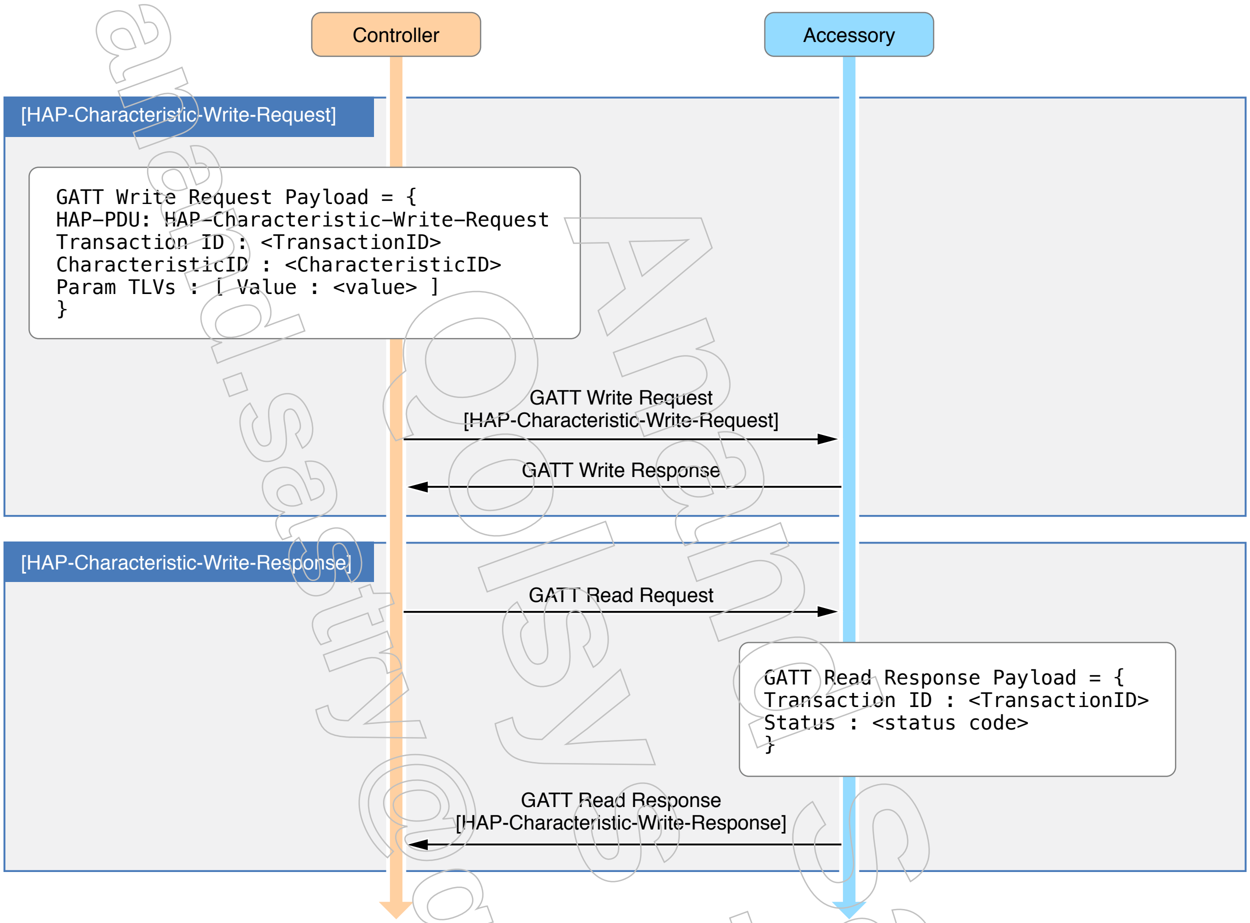

Controller
Accessory
[HAP-Characteristic-Signature-Read-Request]
GATT Write Request
[HAP-Characteristic-Signature-Read-Request]
GATT Write Response
[HAP-Characteristic-Signature-Read-Response]
GATT Read Request
GATT Read Response
[HAP-Characteristic-Signature-Read-Response]
⋮

```
GATT Write Request Payload = {
HAP-PDU: HAP-Characteristic-Signature-Read-Request
Transaction ID : <TransactionID>
CharacteristicID : <CharacteristicID>
}
```

```
GATT Read Response Payload = {
Transaction ID : <TransactionID>
Status : <status code>
Param TLVs : [<CharacteristicType>,
<ServiceType>, <ServiceID>,
<HAP-Chr-Properties>, <GATT-User-Description>,
<GATT-Format>, <GATT-Valid-Range>,
<HAP-Step-Value>]
}
```

Figure 7-1: HAP Characteristic Signature Read Procedure

7.3.5.2 HAP Characteristic Write Procedure


Controller
Accessory
[HAP-Characteristic-Write-Request]
GATT Write Request
[HAP-Characteristic-Write-Request]
GATT Write Response
[HAP-Characteristic-Write-Response]
GATT Read Request
GATT Read Response
[HAP-Characteristic-Write-Response]

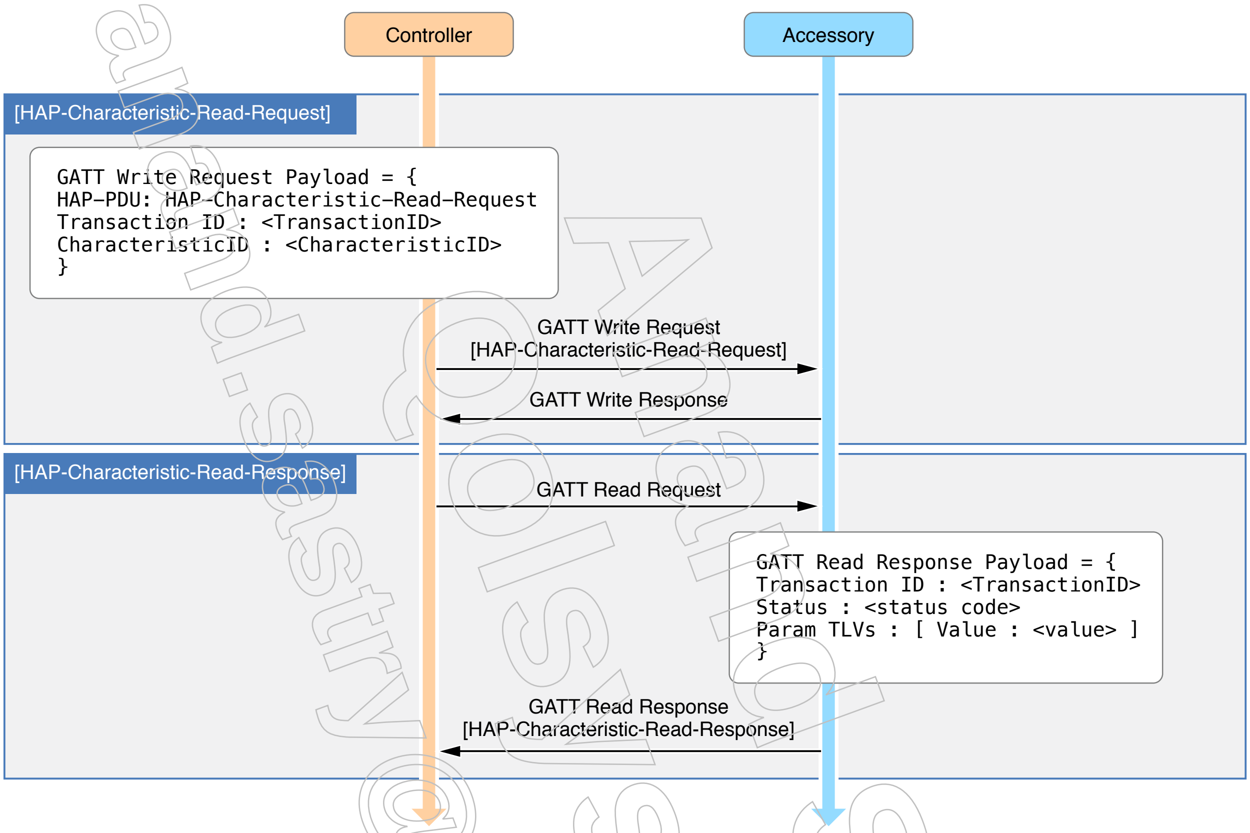
```
GATT Write Request Payload = {
HAP-PDU: HAP-Characteristic-Write-Request
Transaction ID : <TransactionID>
CharacteristicID : <CharacteristicID>
Param TLVs : [ Value : <value> ]
}
```

```
GATT Read Response Payload = {
Transaction ID : <TransactionID>
Status : <status code>
}
```

Figure 7-2: HAP Characteristic Write Procedure

This procedure is used to write a value to the HAP characteristic.

7.3.5.3 HAP Characteristic Read Procedure


Controller
Accessory
[HAP-Characteristic-Read-Request]
GATT Write Request
[HAP-Characteristic-Read-Request]
GATT Write Response
[HAP-Characteristic-Read-Response]
GATT Read Request
GATT Read Response
[HAP-Characteristic-Read-Response]

```
GATT Write Request Payload = {
HAP-PDU: HAP-Characteristic-Read-Request
Transaction ID : <TransactionID>
CharacteristicID : <CharacteristicID>
}
```

```
GATT Read Response Payload = {
Transaction ID : <TransactionID>
Status : <status code>
Param TLVs : [ Value : <value> ]
}
```

Figure 7-3: HAP Characteristic Read Procedure

This procedure is used to read a value of the HAP characteristic.

7.3.5.4 HAP Characteristic Timed Write Procedure

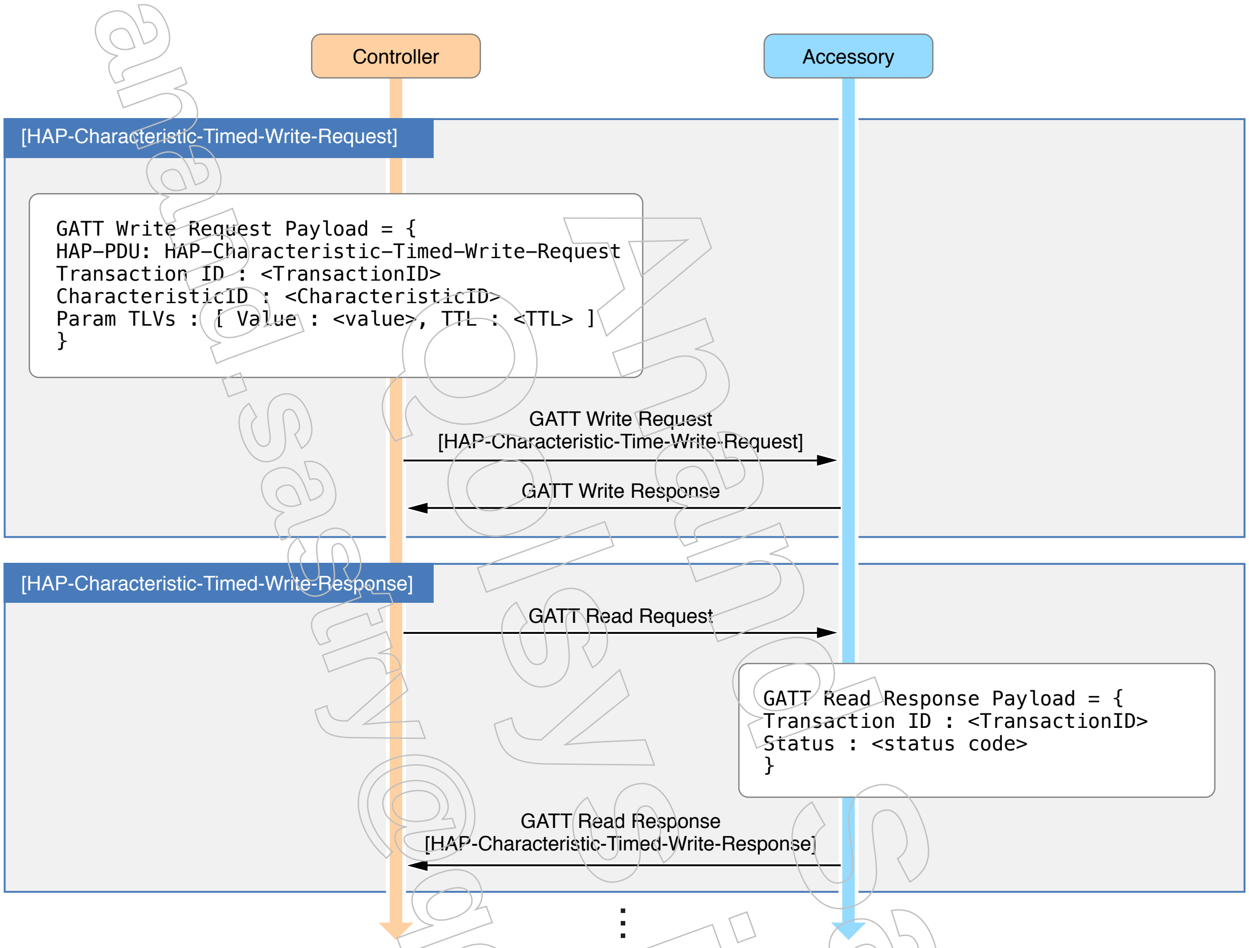
The HAP characteristic timed write procedure will write to characteristic that require time sensitive actions. An example is security class characteristics (Lock Target State, Target Door State etc). The accessory must indicate the characteristic requires the timed write in the HAP Characteristics Properties descriptor. An accessory must support timed writes to all characteristics even if the characteristic property does not require it.

The HAP characteristic timed write procedure shall be used to securely execute a write command within a specified TTL. The accessory must start the TTL timer after sending the HAP-Characteristic-Timed-Write-Response. The scheduled request shall be executed only if the accessory receives the HAP-Characteristic-ExecuteWrite-Request before its TTL timer expires.

If the accessory receives a HAP-Characteristic-Execute-Write-Request after the TTL timer has expired it shall ignore the request and respond with an error in the HAP-Characteristic-Execute-Write-Response.

The controller will send a HAP-Characteristic-Execute-Write-Request only after it has received a HAP-Characteristic-Timed-Write-Response within the TTL milliseconds of having issued the GATT-Read-Request for reading the response.

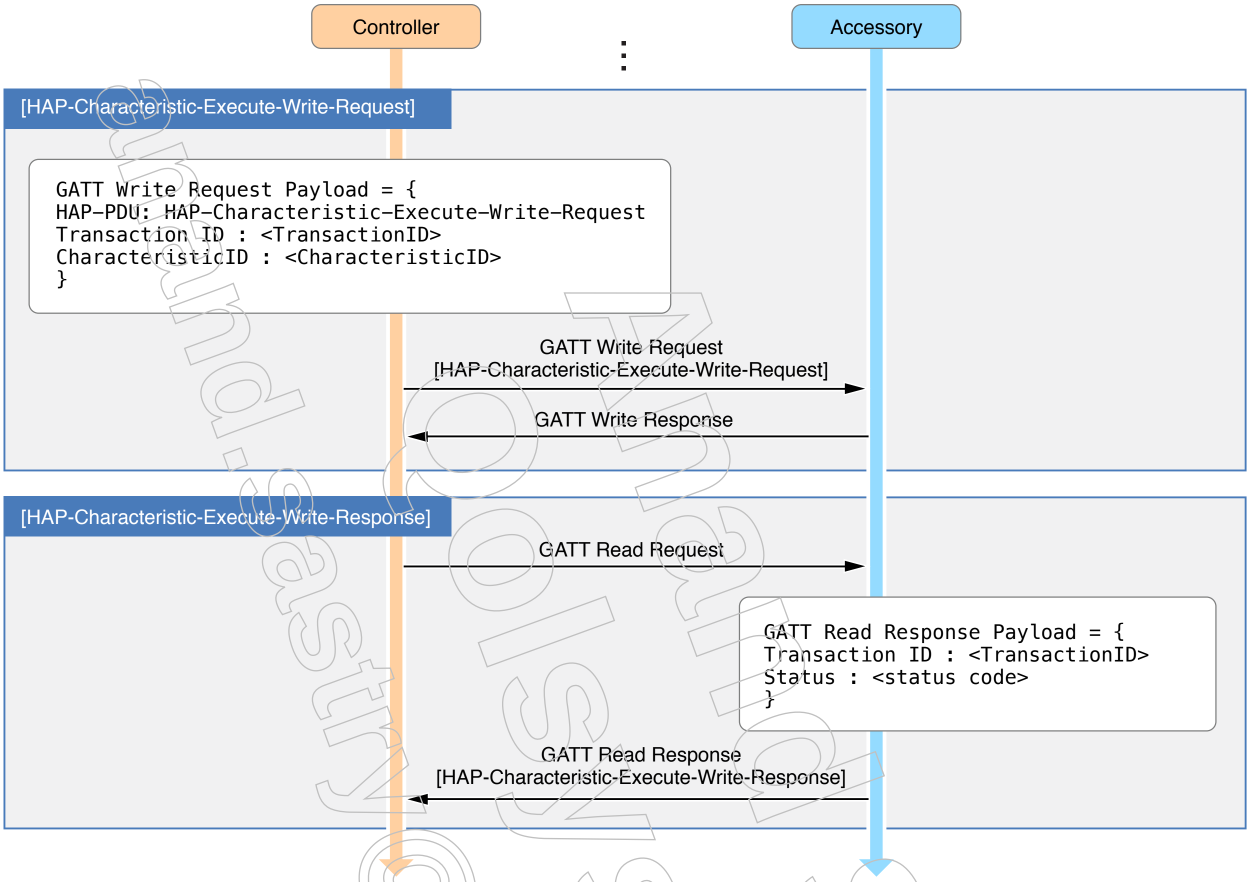
If the accessory receives another HAP procedure to the same characteristics in the middle of the timed write procedure, it shall drop the security session and disconnect the Bluetooth link.


Controller
Accessory
[HAP-Characteristic-Timed-Write-Request]
GATT Write Request
[HAP-Characteristic-Time-Write-Request]
GATT Write Response
[HAP-Characteristic-Timed-Write-Response]
GATT Read Request
GATT Read Response
[HAP-Characteristic-Timed-Write-Response]
⋮

```
GATT Write Request Payload = {
HAP-PDU: HAP-Characteristic-Timed-Write-Request
Transaction ID : <TransactionID>
CharacteristicID : <CharacteristicID>
Param TLVs : [ Value : <value>, TTL : <TTL> ]
}
```

```
GATT Read Response Payload = {
Transaction ID : <TransactionID>
Status : <status code>
}
```

Figure 7-4: HAP Characteristic Timed Write Procedure (1)


Controller
Accessory
[HAP-Characteristic-Execute-Write-Request]
GATT Write Request
[HAP-Characteristic-Execute-Write-Request]
GATT Write Response
[HAP-Characteristic-Execute-Write-Response]
GATT Read Request
GATT Read Response
[HAP-Characteristic-Execute-Write-Response]

```
GATT Write Request Payload = {
HAP-PDU: HAP-Characteristic-Execute-Write-Request
Transaction ID : <TransactionID>
CharacteristicID : <CharacteristicID>
}
```

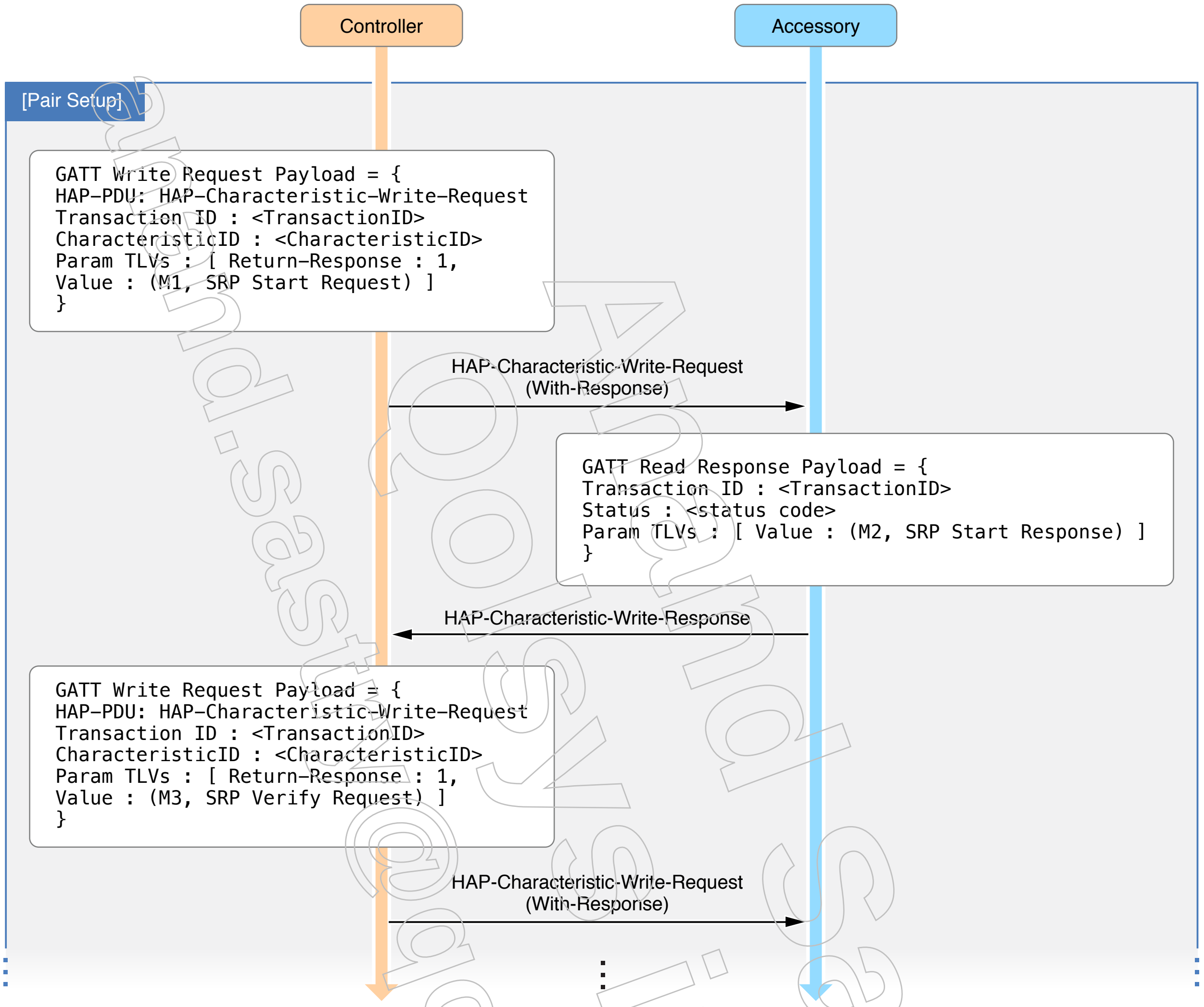
```
GATT Read Response Payload = {
Transaction ID : <TransactionID>
Status : <status code>
}
```

Figure 7-5: HAP Characteristic Timed Write Procedure (2)

7.3.5.5 HAP Characteristic Write-with-Response Procedure

This procedure extends the HAP Characteristic Write Request Procedure by adding an additional TLV (HAP-Param-Return-Response) to the request body's TLVs in addition to the characteristic value (HAP-Param-Value). This procedure is used when a HAP write request generates a response value. The generated response is returned in the HAP-Characteristic-Write-Response's PDU's body as an additional TLV (HAP-Param-Value).

Pair setup and pair verify use this procedure as illustrated in Figure 7-6 (page 118), Figure 7-7 (page 119) and Figure 7-8 (page 120) (Note: The underlying GATT write and read requests to deliver the HAP requests and responses are not shown)

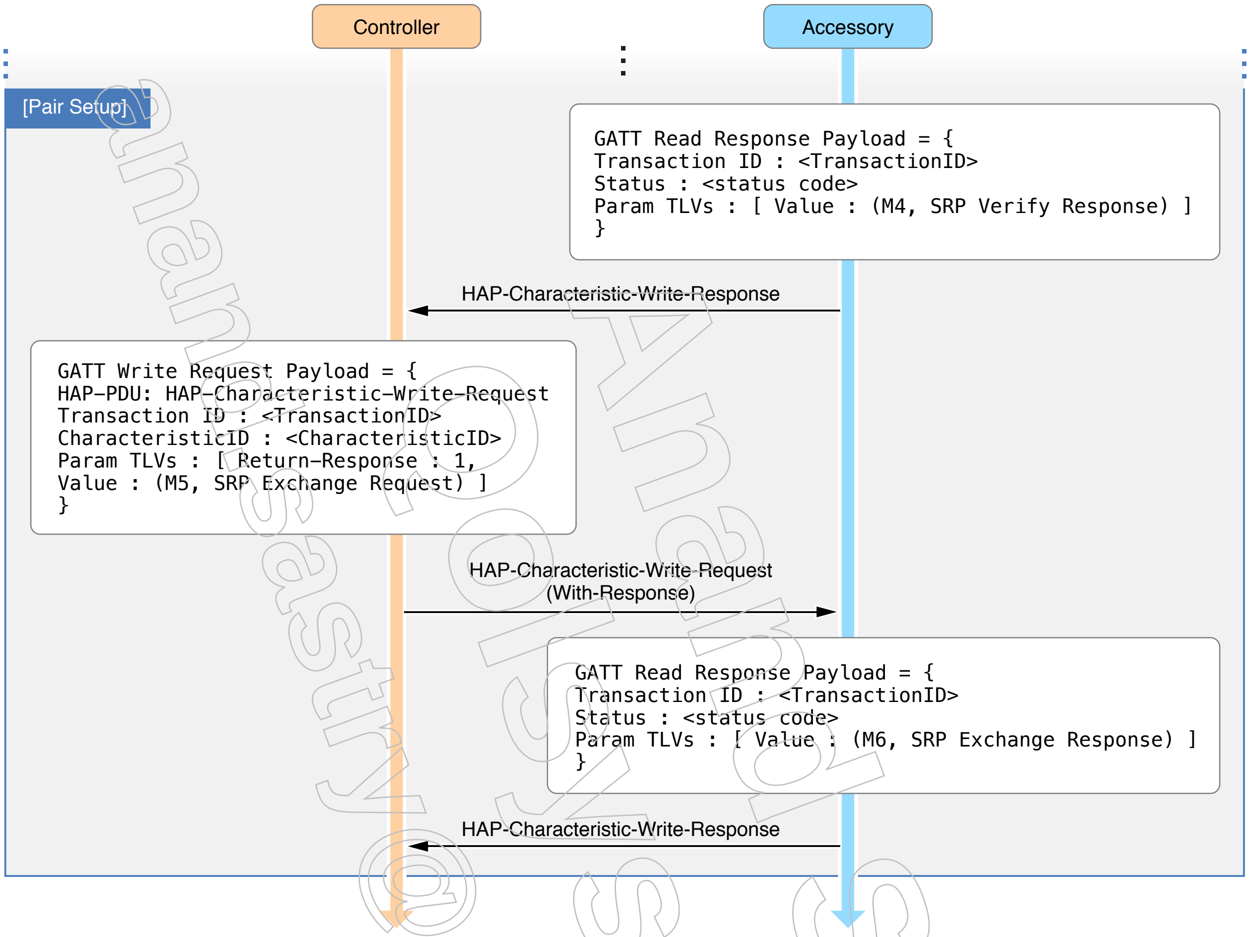

Controller
Accessory
[Pair Setup]
HAP-Characteristic-Write-Request
(With-Response)
HAP-Characteristic-Write-Response
HAP-Characteristic-Write-Request
(With-Response)

```
GATT Write Request Payload = {
HAP–PDU: HAP–Characteristic–Write–Request
Transaction ID : <TransactionID>
CharacteristicID : <CharacteristicID>
Param TLVs : [ Return–Response : 1,
Value : (M1, SRP Start Request) ]
}
```

```
GATT Read Response Payload = {
Transaction ID : <TransactionID>
Status : <status code>
Param TLVs : [ Value : (M2, SRP Start Response) ]
}
```

```
GATT Write Request Payload = {
HAP–PDU: HAP–Characteristic–Write–Request
Transaction ID : <TransactionID>
CharacteristicID : <CharacteristicID>
Param TLVs : [ Return–Response : 1,
Value : (M3, SRP Verify Request) ]
}
```

Figure 7-6: HAP Characteristic Write-with-Response Pair Setup Procedure (1)

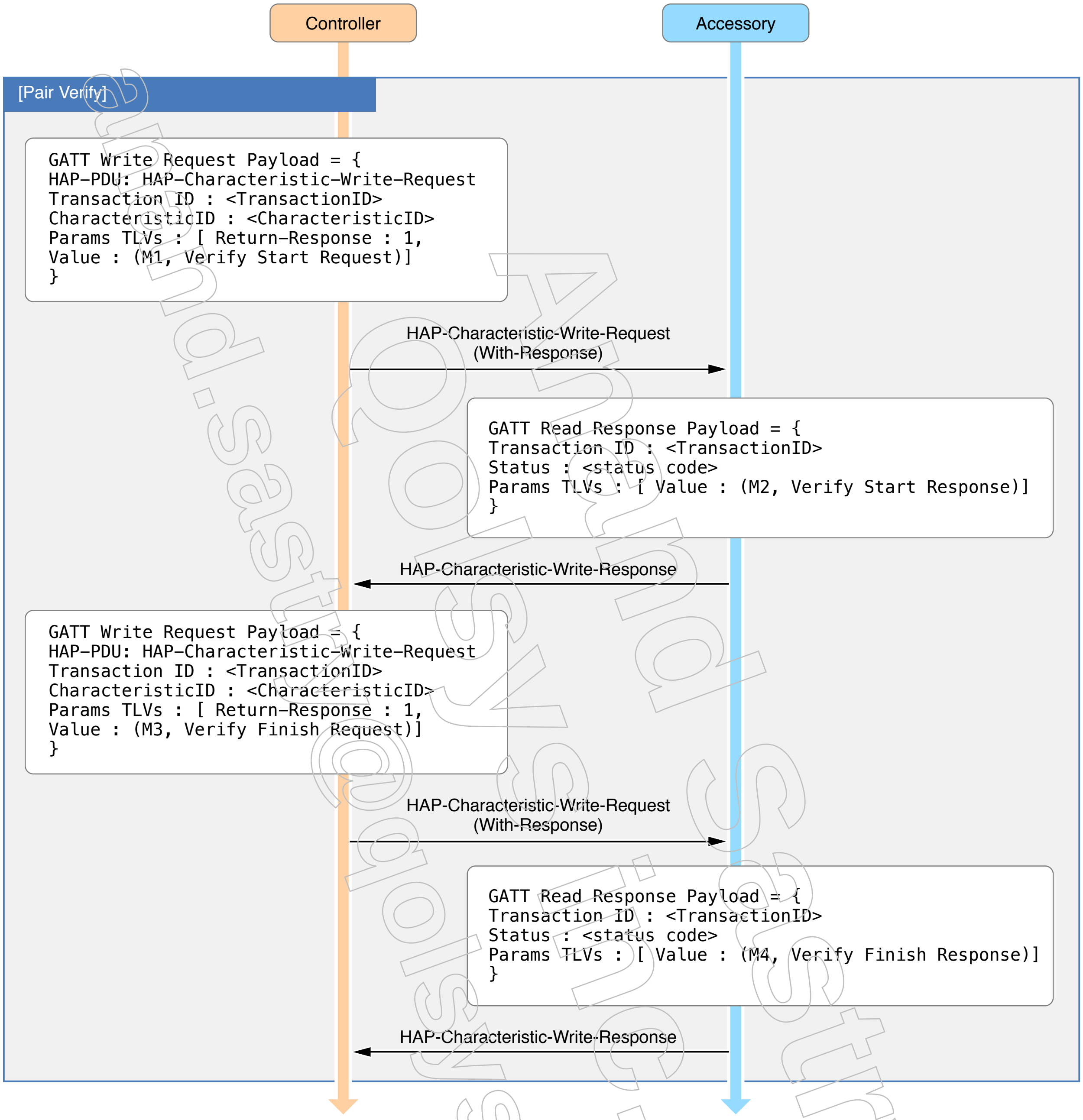

Controller
Accessory
[Pair Setup]
HAP-Characteristic-Write-Response
HAP-Characteristic-Write-Request
(With-Response)
HAP-Characteristic-Write-Response

```
GATT Read Response Payload = {
Transaction ID : <TransactionID>
Status : <status code>
Param TLVs : [ Value : (M4, SRP Verify Response) ]
}
```

```
GATT Write Request Payload = {
HAP–PDU: HAP–Characteristic–Write–Request
Transaction ID : <TransactionID>
CharacteristicID : <CharacteristicID>
Param TLVs : [ Return–Response : 1,
Value : (M5, SRP Exchange Request) ]
}
```

```
GATT Read Response Payload = {
Transaction ID : <TransactionID>
Status : <status code>
Param TLVs : [ Value : (M6, SRP Exchange Response) ]
}
```

Figure 7-7: HAP Characteristic Write-with-Response Pair Setup Procedure (2)


Controller
Accessory
[Pair Verify]
HAP-Characteristic-Write-Request
(With-Response)
HAP-Characteristic-Write-Response
HAP-Characteristic-Write-Request
(With-Response)
HAP-Characteristic-Write-Response

```
GATT Write Request Payload = {
HAP-PDU: HAP-Characteristic-Write-Request
Transaction ID : <TransactionID>
CharacteristicID : <CharacteristicID>
Params TLVs : [ Return-Response : 1,
Value : (M1, Verify Start Request)]
}
```

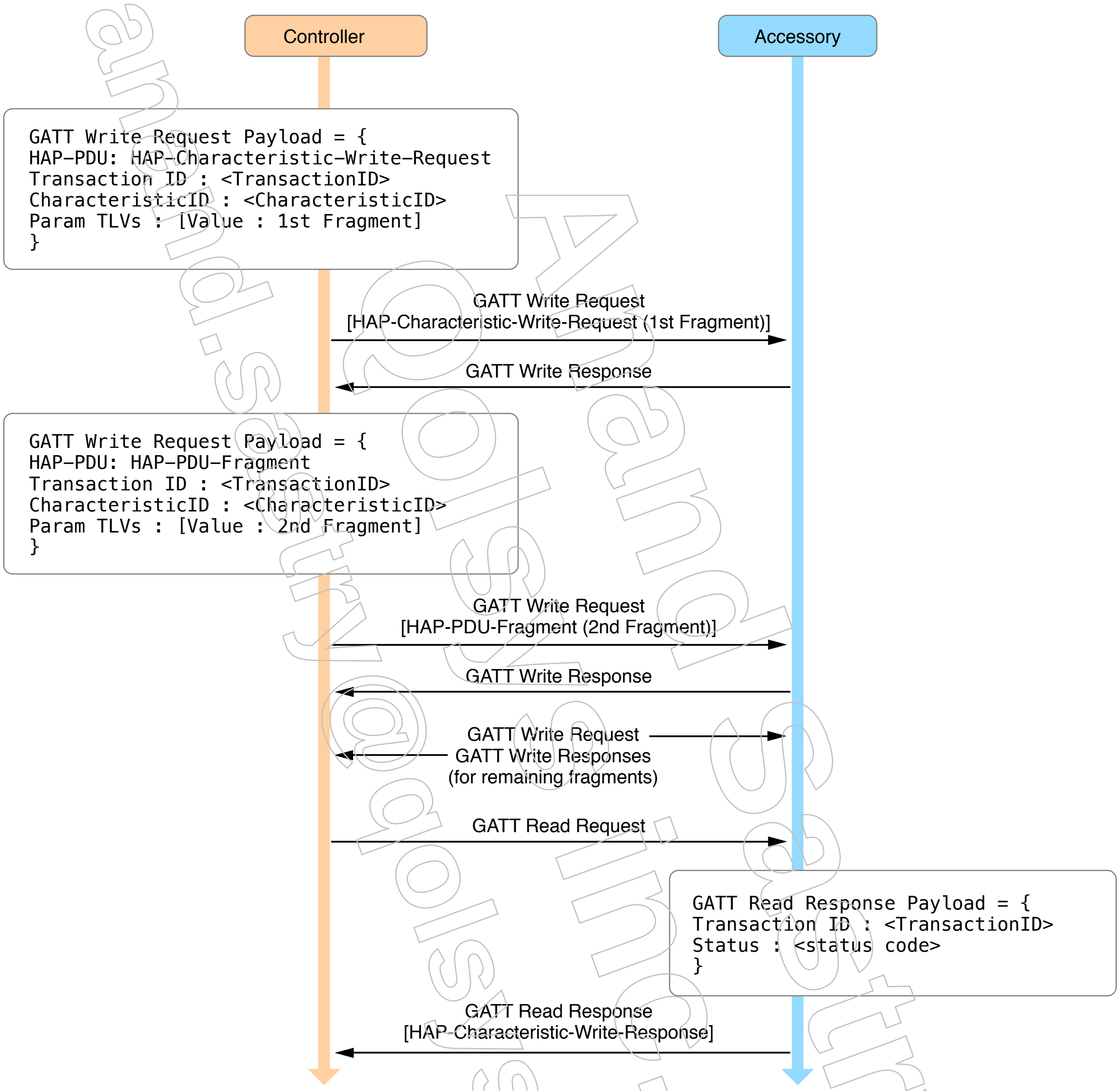
```
GATT Read Response Payload = {
Transaction ID : <TransactionID>
Status : <status code>
Params TLVs : [ Value : (M2, Verify Start Response)]
}
```

```
GATT Write Request Payload = {
HAP-PDU: HAP-Characteristic-Write-Request
Transaction ID : <TransactionID>
CharacteristicID : <CharacteristicID>
Params TLVs : [ Return-Response : 1,
Value : (M3, Verify Finish Request)]
}
```

```
GATT Read Response Payload = {
Transaction ID : <TransactionID>
Status : <status code>
Params TLVs : [ Value : (M4, Verify Finish Response)]
}
```

Figure 7-8: HAP Characteristic Write-with-Response Pair Verify Procedure

7.3.5.6 HAP Fragmented Writes



```
GATT Write Request Payload = {
HAP-PDU: HAP-Characteristic-Write-Request
Transaction ID : <TransactionID>
CharacteristicID : <CharacteristicID>
Param TLVs : [Value : 1st Fragment]
}
```

```
GATT Write Request Payload = {
HAP-PDU: HAP-PDU-Fragment
Transaction ID : <TransactionID>
CharacteristicID : <CharacteristicID>
Param TLVs : [Value : 2nd Fragment]
}
```

```
GATT Read Response Payload = {
Transaction ID : <TransactionID>
Status : <status code>
}
```

Figure 7-9: HAP Fragmented Writes

7.3.5.7 HAP Fragmented Read

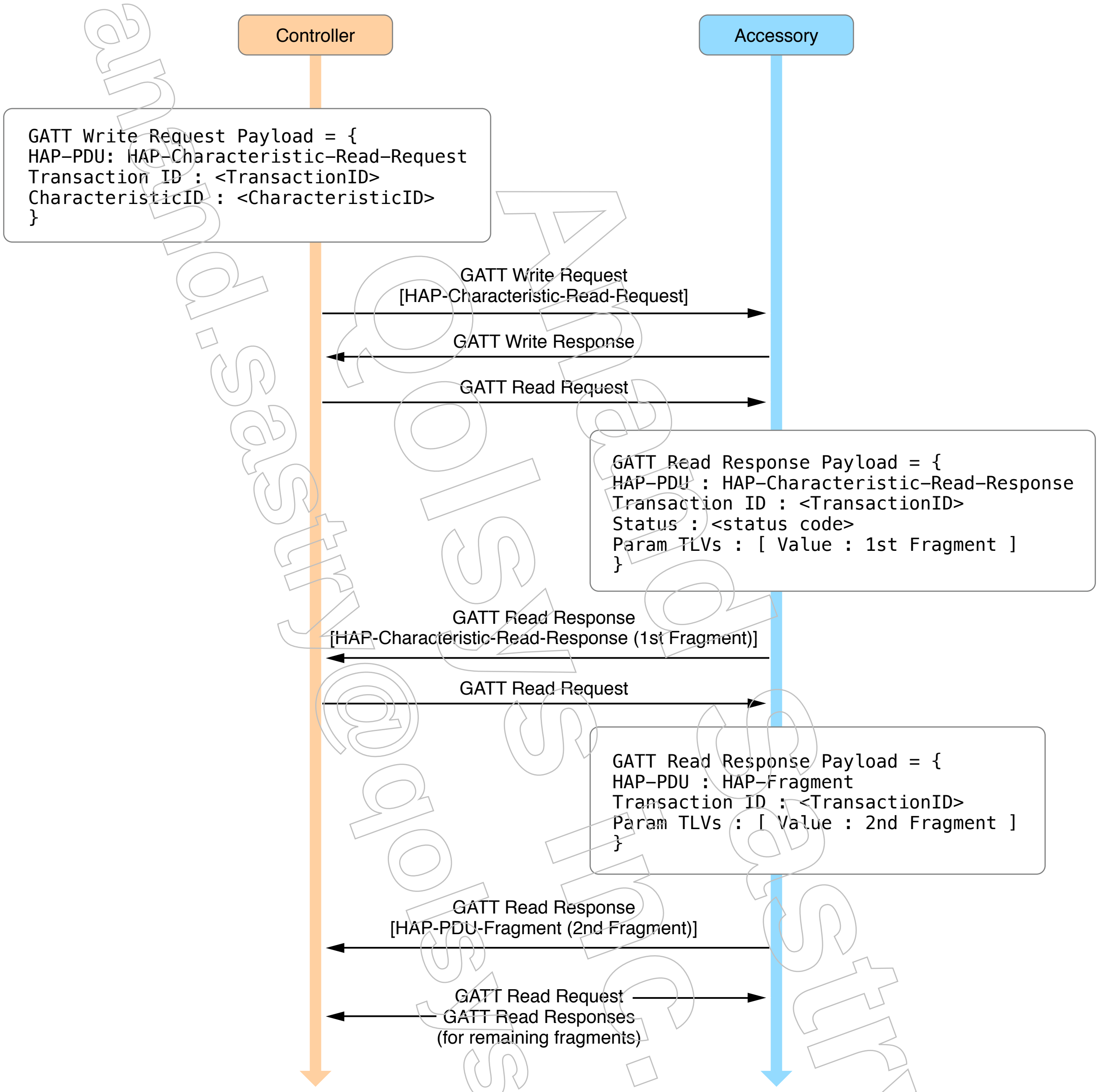

Controller
Accessory
GATT Write Request Payload = {
HAP–PDU: HAP–Characteristic–Read–Request
Transaction ID : <TransactionID>
CharacteristicID : <CharacteristicID>
}
GATT Write Request
[HAP-Characteristic-Read-Request]
GATT Write Response
GATT Read Request
GATT Read Response Payload = {
HAP–PDU : HAP–Characteristic–Read–Response
Transaction ID : <TransactionID>
Status : <status code>
Param TLVs : [Value : 1st Fragment]
}
GATT Read Response
[HAP-Characteristic-Read-Response (1st Fragment)]
GATT Read Request
GATT Read Response Payload = {
HAP–PDU : HAP–Fragment
Transaction ID : <TransactionID>
Param TLVs : [Value : 2nd Fragment]
}
GATT Read Response
[HAP-PDU-Fragment (2nd Fragment)]
GATT Read Request
GATT Read Responses
(for remaining fragments)

Figure 7-10: HAP Fragmented Read

7.3.5.8 HAP Characteristic Configuration Procedure

This procedure is used to enable or disable broadcast notification on each characteristic. When broadcast notification is enabled the "7-30 Broadcast Interval" (page 111) specifies the advertisement interval to be used when the configured characteristic change is indicated via "7.4.6.2 Broadcasted Events" (page 141).

"7-27 HAP-Characteristic-Configuration-Request" (page 110) and "7-31 HAP-Characteristic-Configuration-Response" (page 111) shall be used to configure characteristics.

7.3.5.9 HAP Protocol Configuration Procedure

This procedure is used to generate the Broadcast Encryption Key "7.4.7.3 Broadcast Encryption Key Generation" (page 143) and to get or set protocol specific parameters on the accessory.

"7-33 HAP-Protocol-Configuration-Request" (page 111) and "7-35 HAP-Protocol-Configuration-Response 1" (page 112) shall be used for protocol configuration.

For broadcast encryption key generation and usage see "7.4.7.3 Broadcast Encryption Key Generation" (page 143) and "7.4.7.4 Broadcast Encryption Key expiration and refresh" (page 143)

7.3.6 HAP Status Codes

Table 7-37: HAP Status Codes Description

Status Codes	Definition	Description
0x00	Success	The request was successful.
0x01	Unsupported-PDU	The request failed as the HAP PDU was not recognized or supported.
0x02	Max-Procedures	The request failed as the accessory has reached the the limit on the simultaneous procedures it can handle.
0x03	Insufficient Authorization	Characteristic requires additional authorization data.
0x04	Invalid Instance ID	The HAP Request's characteristic or service instance id did not match the addressed characteristic's instance id.
0x05	Insufficient Authentication	Characteristic access required a secure session to be established.
0x06	Invalid Request	Accessory was not able to perform the requested operation.

7.3.7 Pair-Resume Procedure

The security session restore feature (Pair-Resume) allows accessories that can save the 256 bit shared secret and a 64 bit session identifier from previous secure session to restore a session without any new ECC computations, significantly speeding up the session restoration and overall re-connection time.

7.3.7.1 Accessory Requirements

1. Must support restoring sessions for at-least 8 sessions
2. Must use a Least Frequently Used or a Least Recently Used scheme to forget sessions that exceeds the maximum number of sessions that the accessory can resume.

7.3.7.2 Defines

Table 7-38: Defines Description

Define	Value	Description
kTLVMethod_Resume	0x06	Pair-Resume
kTLVType_SessionID	0x0E	Identifier to resume a session.

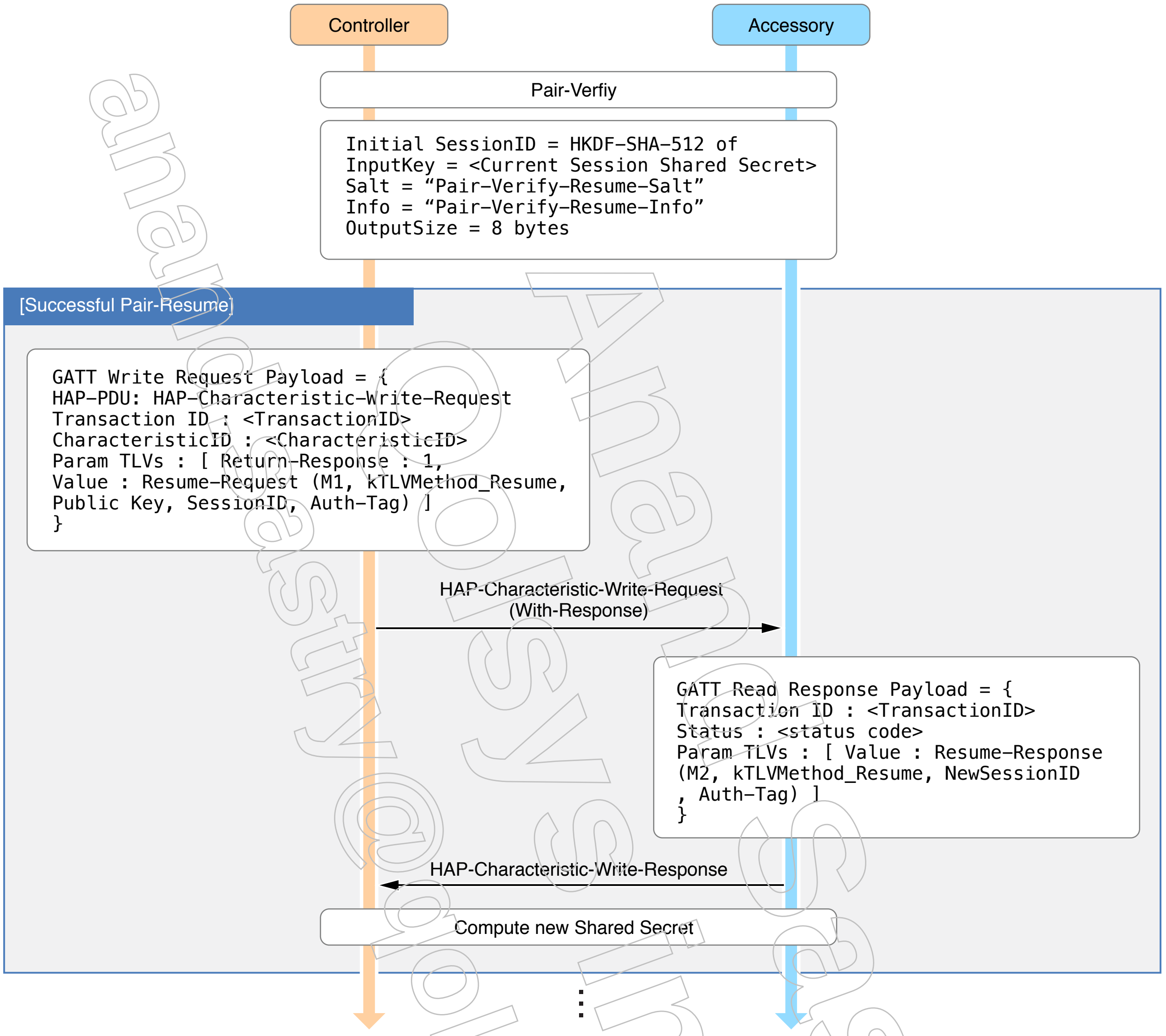
7.3.7.3 Initial SessionID

After an initial pair-verify the controller and the accessory will generate an initial session ID that can be used to restore the security session in the next Pair-Resume attempt. The `SessionID` is computed by using the `HKDF−SHA−512` of the following:

```
InputKey    = <Current Session Shared Secret>
Salt        = ”Pair−Verify−ResumeSessionID−Salt”
Info        = ”Pair−Verify−ResumeSessionID−Info”
OutputSize  = 8 bytes
```

7.3.7.4 Pair Resume

The initial `SessionID` generated during the pair-verify can be used by the controller to resume the subsequent session. Pair-Resume is compatible with the pair-verify procedure and if the accessory does not have the previous `SessionID` stored it can fall back to normal pair-verify procedure.

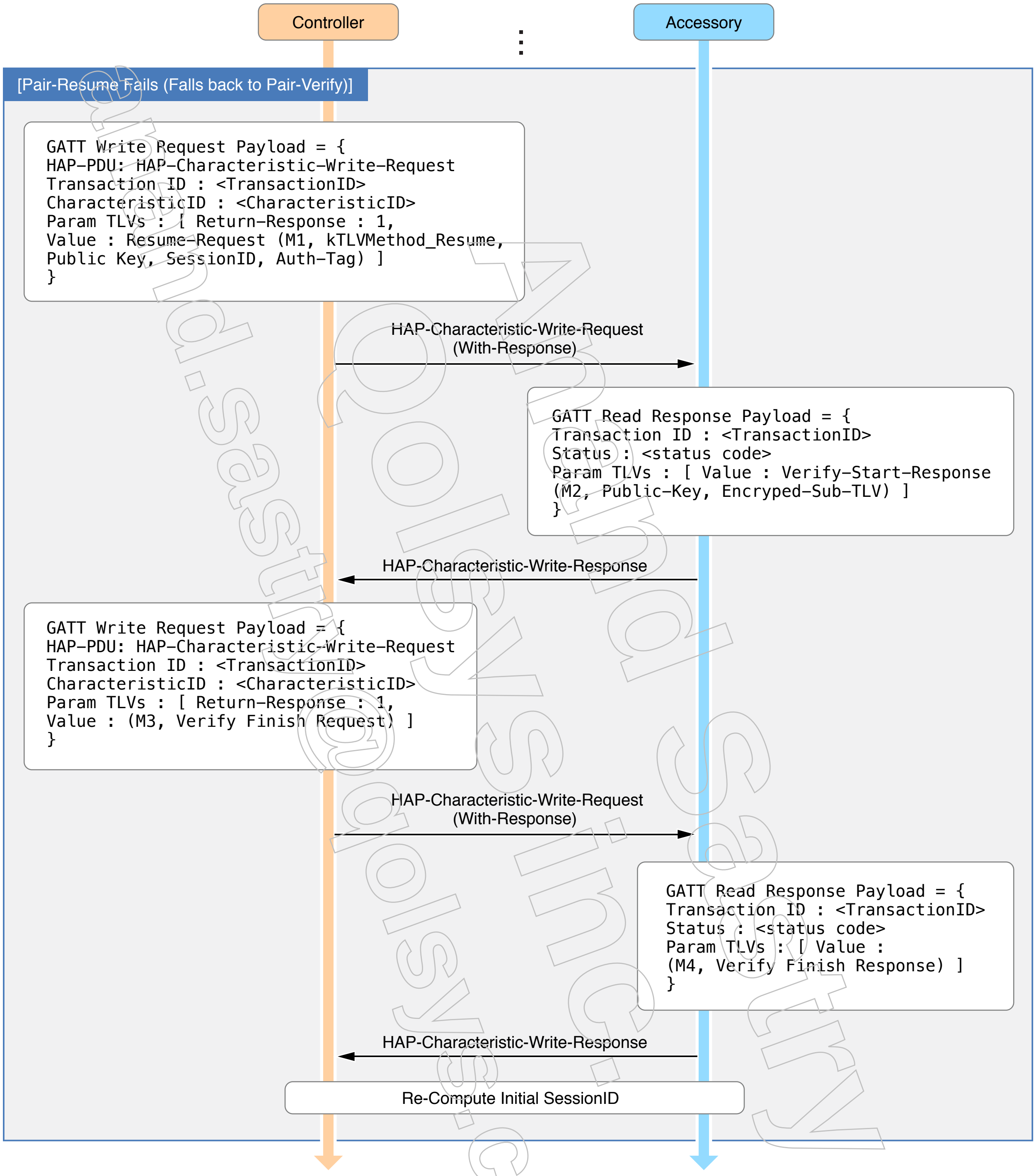

Controller
Accessory
Pair-Verfiy
[Successful Pair-Resume]
HAP-Characteristic-Write-Request
(With-Response)
HAP-Characteristic-Write-Response
Compute new Shared Secret

```
Initial SessionID = HKDF-SHA-512 of
InputKey = <Current Session Shared Secret>
Salt = “Pair-Verify-Resume-Salt”
Info = “Pair-Verify-Resume-Info”
OutputSize = 8 bytes
```

```
GATT Write Request Payload = {
HAP-PDU: HAP-Characteristic-Write-Request
Transaction ID : <TransactionID>
CharacteristicID : <CharacteristicID>
Param TLVs : [ Return-Response : 1,
Value : Resume-Request (M1, kTLVMethod_Resume,
Public Key, SessionID, Auth-Tag) ]
}
```

```
GATT Read Response Payload = {
Transaction ID : <TransactionID>
Status : <status code>
Param TLVs : [ Value : Resume-Response
(M2, kTLVMethod_Resume, NewSessionID
, Auth-Tag) ]
}
```

Figure 7-11: Pair Resume Procedure(1)


Controller
Accessory
⋮
[Pair-Resume Fails (Falls back to Pair-Verify)]
HAP-Characteristic-Write-Request
(With-Response)
HAP-Characteristic-Write-Response
HAP-Characteristic-Write-Request
(With-Response)
HAP-Characteristic-Write-Response
Re-Compute Initial SessionID

```
GATT Write Request Payload = {
HAP-PDU: HAP-Characteristic-Write-Request
Transaction ID : <TransactionID>
CharacteristicID : <CharacteristicID>
Param TLVs : [ Return-Response : 1,
Value : Resume-Request (M1, kTLVMethod_Resume,
Public Key, SessionID, Auth-Tag) ]
}
```

```
GATT Read Response Payload = {
Transaction ID : <TransactionID>
Status : <status code>
Param TLVs : [ Value : Verify-Start-Response
(M2, Public-Key, Encryped-Sub-TLV) ]
}
```

```
GATT Write Request Payload = {
HAP-PDU: HAP-Characteristic-Write-Request
Transaction ID : <TransactionID>
CharacteristicID : <CharacteristicID>
Param TLVs : [ Return-Response : 1,
Value : (M3, Verify Finish Request) ]
}
```

```
GATT Read Response Payload = {
Transaction ID : <TransactionID>
Status : <status code>
Param TLVs : [ Value :
(M4, Verify Finish Response) ]
}
```

Figure 7-12: Pair Resume Procedure(2)

7.3.7.4.1 M1: Controller -> Accessory - Resume Request

The controller performs the following steps:

1. Generate a new and random Curve25519 key pair. This enables <M1> to be usable as the first message of pair verify if pair resume fails.

2. Derive a request encryption key, RequestKey using the HKDF−SHA−512 of the following:

```
InputKey     = <Shared Secret from previous session>
Salt         = <Controller's new Curve25519 public key><SessionID>
Info         = "Pair-Resume-Request-Info"
OutputSize   = 32 bytes
```

3. Encrypt and generate auth-tag with an empty RequestData. This uses the ChaCha20-Poly1305 AEAD algorithm with the following parameters:

```
encryptedData, authTag = ChaCha20-Poly1305(RequestKey, Nonce="PR-Msg01",
AAD=<none>, Msg=<0 byte RequestData>)
```

4. Send request to accessory with the following TLV items:

kTLVType_State	<M1>
kTLVType_Method	<kTLVMethod_Resume>
kTLVType_PublicKey	<Controller's Curve25519 public key>
kTLVType_SessionID	<Session ID to resume>
kTLVType_EncryptedData	<16 bytes of auth tag>

*Note: The Session ID is encoded as a 8 byte little-endian integer

7.3.7.4.2 M2: Accessory -> Controller - Resume Response

When the accessory receives <M1>, it must perform the following steps:

1. Look up the session ID in the list of saved sessions to get shared secret. If the session is not found or has expired, the accessory must treat this as <M1> of pair verify. This avoids needing another round trip to fail and start pair verify. If the session is found and valid then continue with pair resume.

2. Invalidate session ID so it cannot be used again.

3. Derive the encryption key (same as the controller).

4. Verify authTag of encrypted data. If this fails, respond with <M2> and kTLVError_Authentication.

5. Generate new, random session ID*. This can be used for a subsequent pair resume.

   *Note: The <New Session ID> generated by the accessory in the pair resume must be generated randomly and must not be in anyway derived from current or previous session information.

6. Derive a response encryption key, ResponseKey using the HKDF−SHA−512 of the following:

```
InputKey    = <Shared Secret from previous session>
Salt        = <Controller's Curve25519 public key><New SessionID>
Info        = "Pair-Resume-Response-Info"
OutputSize  = 32 bytes
```

*Note: The New Session ID is encoded as a 8 byte little-endian integer

7. Encrypt and generate auth-tag with an empty ResponseData. This uses the ChaCha20-Poly1305 AEAD algorithm with the following parameters:
   `encryptedData, authTag = ChaCha20-Poly1305(ResponseKey, Nonce="PR-Msg02", AAD=<none>, Msg=<0 byte ResponseData>)`

8. Send response to controller with the following TLV items:

```
kTLVType_State          <M2>
kTLVType_Method         <kTLVMethod_Resume>
kTLVType_SessionID      <New Session ID>
kTLVType_EncryptedData  <16 bytes of auth tag>
```

*Note: The New Session ID is encoded as an 8 byte little-endian integer

7.3.7.5 Compute Shared Secret

After a successful pair-resume the accessory and controller derive a new shared secret for subsequent communication. The `Shared Secret` is computed by using the `HKDF-SHA-512` of the following:

```
InputKey    = <Shared Secret from previous session>
Salt        = <Controller's Curve25519 public key><New SessionID>
Info        = "Pair-Resume-Shared-Secret-Info"
OutputSize  = 32 bytes
```

*Note: The New Session ID is encoded as a 8 byte little-endian integer

The new shared secret is used to derive new read/write keys as specified in AEAD Algorithm with the nonce reset to 0.

7.4 Bluetooth LE

7.4.1 General requirements

7.4.1.1 Accessory Role

The accessory must implement the Bluetooth LE peripheral role as specified in the Bluetooth specification.

7.4.1.2 Bluetooth Device Address

The accessory must use a static random Bluetooth Device Address for the LE physical channel.

7.4.1.3 Advertising Channels

The accessory must advertise on all three advertising channels (37, 38, and 39) at each advertising event.

7.4.1.4 Advertising Interval

- The accessory must advertise at a periodic interval.
- The accessory is allowed to change the advertising interval dynamically.
- The accessory shall not advertise while it is connected to a HomeKit controller.
- The accessory must be able to advertise with a minimum interval of 20 ms.
- The accessory must choose a regular advertising interval in the range of 160 ms to 2500 ms, depending on the accessory category and its power consumption characteristics.
- Mains powered accessories or accessories with larger battery capacity should use a shorter interval between 160 ms to 800 ms for its regular advertisements. Battery powered accessories that do not have any controllable Apple defined characteristics (such as temperature sensors, door sensors etc) are allowed to use a larger regular advertising interval between 1250 ms and 2500 ms for enhanced battery life.

7.4.1.5 Transmission Power

The accessory should be able to transmit at the maximum output power level as allowed by the Bluetooth Core Specification. For good wireless range accessories should transmit at least at +4 dBm.

7.4.1.6 MTU Size

The accessory must support an MTU size, ATT_MTU larger than 100 bytes during the ATT Exchange MTU Request handshake. Accessories should use an MTU size of 150 bytes or larger.

7.4.1.7 Maximum Payload Size

The maximum length of an HAP characteristic value shall be 64000 bytes, the maximum length of the Bluetooth LE Attribute value shall be 512 bytes and accessory must support ATT prepared writes of up-to 512 bytes. HAP characteristics that exceed 512 bytes must be fragmented as per the HAP PDU fragmentation scheme.

7.4.1.8 Global State Number (GSN)

The Global State Number represents the state at which a required change on the accessory was last notified to the HomeKit controller. Accessories shall maintain a 16 bit monotonically increasing GSN value. This value must have a range of 1-65535 and wrap to 1 when it overflows. This value must persist across reboots, power cycles, etc. This value must be reset back to 1 when factory reset or a firmware update occurs on the accessory. For more details see "7.4.6 HAP Notifications" (page 140)

7.4.1.9 Unpaired Identify

Unpaired Identify must be allowed only if the accessory is unpaired, i.e. it has no paired controllers. During unpaired identify, unpaired controllers must be allowed to write to "10.14 Identify" (page 183) characteristic.

7.4.1.10 BLE SIG Certification Requirements

- A HAP-BLE accessory must be certified and listed as a qualified end product by the Bluetooth SIG.

7.4.2 HAP BLE Advertisement Formats

All the bytes in the Advertisement Data are reserved and must not be used for any other purpose.

7.4.2.1 HAP BLE Regular Advertisement Format

The regular advertisement is used for broadcasting accessory presence and current state of the accessory. This advertisement data must include the following fields:

7.4.2.1.1 Flags

- Total length of three bytes.
- Length byte with value 0x02: one for the Flags AD type and one for the Flags value.
- Flags AD type, 0x01, as defined by the Bluetooth Core Specification
- The Flags value must have the LE General Discoverable Mode, bit 1, set. Since the LE General Discoverable Mode flag is set the LE Limited Discoverable Mode must not be set, as defined by the Bluetooth Core Specification

Table 7-39: Flags Description

Length	AD Type	Value
0x02	0x01	0bxxxxxx1x

7.4.2.1.2 Manufacturer Data

- Total length of 23 bytes.
- LEN - Length byte with a value 0x16.
- ADT - AD Type of 0xFF.
- CoID - Company Identifier code, 0x004C (Apple, Inc.), in little endian format.
- TY - 8 bits for Type, which shall be set to 0x06.
- STL - 8 bits for SubType and Length, the 3 significant bits specify the HomeKit advertising format SubType and shall be set to 1, and the remaining 5 bits is the length of the remaining bytes in the manufacturer specific data which shall be set to the value 17.
- SF- 8 bits for Status Flags, Bits 1-7 are reserved and shall be set to 0, Bit 0 shall reflect the value of the HAP Pairing Status Flag.
- 48-bit `Device ID` ("5.4 Device ID" (page 38)) of the accessory.
- ACID - 16-bit little endian unsigned Accessory Category Identifier,which indicates the category that best describes the primary function of the accessory. This must have a range of 1-65535. This must take one of the values defined in the "14-1 Accessory Categories" (page 271). The Category Identifier must not change except during a firmware update.
- GSN - 16-bit little endian unsigned Global State Number.
- CN - 8 bits for Configuration Number, with a default starting value of 1. Accessories must increment the config number after a firmware update. This value must have a range of 1-255 and wrap to 1 when it overflows. This value must persist across reboots, power cycles and firmware updates.
- CV - 8 bit little endian Compatible Version, this value shall be set to 0x02 for this version of the HAP BLE.
- SH - 4 byte little endian Setup Hash to support enhanced setup payload information (see "4.2.3 Setup Hash" (page 33))

Table 7-40: Manufacturer Specific Data

Length	AD Type	Company ID	Type	STL	SF	48-bit Device ID
0x16	0xFF	0x4C 0x00	0x06	0xXX	0x00/0x01	0xXX 0xXX 0xXX 0xXX 0xXX 0xXX

Table 7-41: Manufacturer Specific Data - continued

ACID	GSN	CN	CV	SH
0xXX 0xXX	0xXX 0xXX	0xXX	0x02	0xXX 0xXX 0xXX 0xXX

Table 7-42: SubType / Length

Value	SubType
0x31	HomeKit regular advertisement

Table 7-43: HAP Pairing Status Flag

Value	Description
0b0	The accessory has been paired with a controllers.
0b1	The accessory has not been paired with any controllers.

7.4.2.1.3 Local Name

- Length byte that corresponds to the total length of the local name plus the Local Name AD type.
- Local Name AD type, 0x09 or 0x08, as defined by the Bluetooth Core Specification. The Advertisement data shall contain either the 'Shortened Local Name' or the 'Complete Local Name' If the complete local name can be included in the Advertisement data, the AD type of 0x09 (Complete Local Name) must be used, else the AD type of of 0x08 (Shortened Local Name) must be used with the name truncated to fit in the advertisement data. When the advertisement includes the shortened local name the accessory should include the complete local name in the Scan Response.

Table 7-44: Local Name

Length	AD Type	Value
0xXX	0x08/0x09	0xXX 0xXX ... 0xXX

7.4.2.2 HAP BLE Encrypted Notification Advertisement Format

This advertisement must only be used to notify changes in characteristic values that are configured for "7.4.6.2 Broadcasted Events" (page 141).

The advertisement data must include the following fields:

7.4.2.2.1 Flags

- Total length of three bytes.
- Length byte with value 0x02: one for the Flags AD type and one for the Flags value.

- Flags AD type, 0x01, as defined by the Bluetooth Core Specification
- The Flags value must have the LE General Discoverable Mode, bit 1, set. Since the LE General Discoverable Mode flag is set the LE Limited Discoverable Mode must not be set, as defined by the Bluetooth Core Specification

Table 7-45: Flags Description

Length	AD Type	Value
0x02	0x01	0bxxxxxx1x

7.4.2.2.2 Manufacturer Data

- Total length of 28 bytes.
- LEN - Length byte with a value 0x1B.
- ADT - AD Type of 0xFF.
- CoID - Company Identifier code, 0x004C (Apple, Inc.), in little endian format.
- TY - 8 bits for Type, which shall be set to 0x11.
- STL - 8 bits for HomeKit SubType and Length, the 3 significant bits specify the HomeKit SubType, and the remaining 5 bits is the length of the remaining bytes in the manufacturer specific data which shall be set to the value 22 (i.e the lower nibble must be set to 0x16).
- 48-bit `Accessory Advertising Identifier` The 6 byte unique identifier assigned to the accessory. By default this identifier is the same as the accessory's unique pairing identifier i.e. 48-bit `Device ID` ("5.4 Device ID" (page 38)) of the accessory.
- Encrypted and Authenticated Payload
  - GSN - 16-bit little endian unsigned Global State Number.
  - IID - 16-bit Characteristic instance id for the characteristic value included in this notification.
  - Value - 8 to 64 bits little endian characteristic value. Only characteristics with fixed length formats can be included in the broadcast notification. See "7-51 HAP Format to BT SIG Format mapping" (page 138) (Characteristics with format of string or data/tlv8 cannot be used in broadcast notifications) When the characteristic value is less than 8 bytes the remaining bytes shall be set to 0.
  - Auth Tag - 32 bit truncated AuthTag.

Table 7-46: Manufacturer Specific Data

Length	AD Type	Company ID	Type	STL	48-bit Accessory Advertising Identifier
0x1B	0xFF	0x4C 0x00	0x11	0xXX	0xXX 0xXX 0xXX 0xXX 0xXX 0xXX

Table 7-47: Manufacturer Specific Data - continued (Encrypted and Authenticated Payload)

GSN	IID	Value	Auth Tag
0xXX 0xXX	0xXX 0xXX	0xXX 0xXX 0xXX 0xXX 0xXX 0xXX 0xXX 0xXX	0xXX 0xXX 0xXX 0xXX

Table 7-48: SubType / Length

Value	SubType / Length
0x36	HomeKit encrypted notification

7.4.3 HAP-BLE Protocol Information Service

"9.2 HAP Protocol Information" (page 153)

7.4.3.1 BLE Protocol Version Characteristic

This is a version string with the following format: "X.Y.Z". This must be set to "2.2.0" for this version of HAP BLE. Please see "10.37 Version" (page 193) for detailed definition of the characteristic.

7.4.3.2 Service Signature Characteristic

The accessory must support "7.4.4.5 Characteristic Descriptors included in the GATT Database" (page 136) on this service to allow configuration of protocol features and parameters for the accessory. "7.3.5.9 HAP Protocol Configuration Procedure" (page 123). In addtion the "7.4.3 HAP-BLE Protocol Information Service" (page 134) must set "Service Supports Configuration" in its "7-49 HAP Service Properties" (page 135) to indicate support for protocol configuration.

7.4.4 Attributes of HAP Services and Characteristics

HAP-BLE accessories must include the "9.1 Accessory Information" (page 153) and the "9.2 HAP Protocol Information" (page 153) in its GATT database.

7.4.4.1 HAP Services and HAP Characteristics

Services containing the "7.4.4.3 Service Instance ID" (page 135) and characteristics that contain the "7.4.4.5 Characteristic Descriptors included in the GATT Database" (page 136) are defined as HAP service and HAP characteristic respectively.

7.4.4.2 Instance IDs

- Every HAP service and characteristic must include an instance ID that is unique and assigned from the same number pool.
- The instance IDs must be maintained for the lifetime of the HAP pairing.
- The instance IDs must have values > 0 and a maximum value of (2^16)-1, 65535.
- "9.1 Accessory Information" (page 153) must have an instance ID of 1.

7.4.4.3 Service Instance ID

Every HAP service must have a read-only Service Instance ID characteristic (`SvcID`) with UUID `E604E95D-A759-4817-87D3-AA005083A0D1`.

The `SvcID` is read using standard GATT read request (this implies that the HAP Characteristic Read Request procedure is not used to read the service instance ID characteristic). The value of this characteristic is a 16-bit unsigned integer encoded as a 2 byte little endian integer. This characteristic must always be read in the clear (with and without an established HAP secure session). (Note: `SvcID` characteristic must not have its own characteristic instance ID)

7.4.4.4 HAP Service Properties

The service properties is a multi byte value where only 2 bytes shall be used in this version. The value is encoded as a 2 byte little endian integer.

Accessories must include the "HAP Service Properties" characteristic only if it supports non-default properties or has linked services. Other services must not include this characteristic.

Accessories must not change its service signature once it is paired with a controller.

Table 7-49: HAP Service Properties

BitMask	Description
0x0001	Primary Service
0x0002	Hidden Service
0x0004	Service Supports Configuration

7.4.4.4.1 HAP Linked Services

Accessories must include a HAP-Param-Linked-Services in its service signature read response as an array of linked services instance ids. When the service does not link to other services it must return an empty list with length = 0.

7.4.4.5 Characteristic Descriptors included in the GATT Database

7.4.4.5.1 Bluetooth LE Characteristic Properties

- The Bluetooth LE characteristic properties as defined in GATT Bluetooth Core Specification must only be a combination of the following: 'Read', 'Write' and 'Indicate'. All other possible properties are not permitted.
- All Bluetooth LE characteristics that map to a HAP characteristics must have the 'Read' and 'Write' property set to allow for read and write to the LE characteristic. Note that the Bluetooth LE Characteristic only indicates the GATT Read / Write and Indicate operations possible on the LE characteristic, the properties associated with the HAP characteristic is indicated by the HAP Characteristic Properties Descriptor.
- Characteristics that support HAP Events must indicate by setting the 'Indicate' properly on the LE Characteristic.

7.4.4.5.2 Characteristic Instance ID

The Characteristic Instance ID is a custom characteristic descriptor with UUID `DC46F0FE-81D2-4616-B5D9-6ABDD796939A`. The accessory must expose the instance ID of each characteristic in the Characteristic Instance ID Descriptor. The value of the descriptor is a 16-bit unsigned integer encoded as a 2 byte little endian integer. This descriptor value must support always being read in the clear (with and without a security session).

7.4.4.5.3 Client Characteristic Configuration

Characteristics supporting "7.4.6 HAP Notifications" (page 140) must support the Client Characteristics Configuration descriptor with the 'Indication' bit, All other configurations are reserved and shall not be used. This descriptor value must support always being read in the clear, i.e. with or without a security session.

7.4.4.5.4 Service Signature Characteristic

The HAP service should include an optional "Service Signature" HAP characteristic with UUID 000000A5-0000-1000-8000-0026BB765291 that support HAP procedures for services. This Service Signature characteristic is a HAP Characteristic with a valid characteristic instance id descriptor. The service signature procedures is used to read additional information pertaining to the service such as the supported properties for the service and the list of services that the service links to. The Service Signature Characteristic on the "7.4.3 HAP-BLE Protocol Information Service" (page 134) is used to configure the accessory for protocol specific configurations and set protocol parameters.

7.4.4.6 Characteristic Metadata Descriptors

HAP Characteristics can include additional descriptors that describe HAP properties, description, format and valid values for the characteristic. These descriptors are read by the controller using the HAP Characteristic Signature Read Procedure. These descriptors must not be published in the Bluetooth LE GATT database.

7.4.4.6.1 HAP Characteristic Properties

This descriptor is used to specify HAP specific properties associated with the characteristic. This descriptor is a multi byte value where only 2 bytes shall be used in this version. The value of this descriptor is encoded as a 2 byte little endian integer.

Table 7-50: HAP Characteristic Description

BitMask	Description
0x0001	Characteristic Supports Read
0x0002	Characteristic Supports Write
0x0004	Characteristic Supports Additional Authorization Data
0x0008	Characteristic Requires HAP Characteristic Timed Write Procedure
0x0010	Characteristics Supports Secure Reads
0x0020	Characteristics Supports Secure Writes
0x0040	Characteristic Hidden from User
0x0080	Characteristic Notifies Events in Connected State
0x0100	Characteristic Notifies Events in Disconnected State
0x0200	Characteristic Supports Broadcast Notify

7.4.4.6.2 Characteristic User Description

This descriptor contains a user description for the characteristic which is an UTF-8 string that is a user textual description of the characteristic.

7.4.4.6.3 Characteristic Presentation Format

This descriptor is defined as per the Bluetooth SIG-defined Characteristic Presentation Format descriptor. Only the 'Format' and 'Unit' fields shall be used, the other fields shall be set according to the following rules:

- Exponent shall be set to 0
- Namespace shall be set to 1
- Description shall be set to 0

BT SIG-defined Format shall be mapped to HAP Formats according to Table 7-51 (page 138) Format of characteristics that are not defined as part of this specification shall default to the format type "data".

Table 7-51: HAP Format to BT SIG Format mapping

HAP Format	BT SIG Format Value	BT SIG Description
bool	0x01	unsigned 8-bit; 0 = false, 1 = true
uint8	0x04	unsigned 8-bit integer
uint16	0x06	unsigned 16-bit integer
uint32	0x08	unsigned 32-bit integer
uint64	0x0A	unsigned 64-bit integer
int	0x10	signed 32-bit integer
float	0x14	IEEE-754 32-bit floating point
string	0x19	UTF-8 string
data/tlv8	0x1B	Opaque structure

BT SIG-defined Unit shall be mapped to HAP Units according to Table 7-52 (page 138)

Table 7-52: HAP Unit to BT SIG Unit mapping

HAP Units	BT SIG Assigned Number	Type
celsius	0x272F	org.bluetooth.unit.thermodynamic_temperature.degree_celsius
arcdegrees	0x2763	org.bluetooth.unit.plane_angle.degree
percentage	0x27AD	org.bluetooth.unit.percentage
unitless	0x2700	org.bluetooth.unit.unitless
lux	0x2731	org.bluetooth.unit.illuminance.lux
seconds	0x2703	org.bluetooth.unit.time.second

7.4.4.6.4 Minimum and Maximum Length Descriptor

The Bluetooth SIG-defined 'Valid Range' descriptor (see `https://developer.bluetooth.org/gatt/descriptors/`) is used to specify the Minimum Value and Maximum Value for the characteristic. The format for this descriptors value is the same as the characteristics format, this descriptor must only be used for formats of integer types or float.

7.4.4.6.5 Step Value Descriptor

This descriptor is used to specify the minimum step value of the characteristic. The format for this descriptors value is the same as the characteristics format, this descriptor must only be used for formats of integer types or float.

7.4.5 Additional HAP Characteristic Requirements

Characteristics which are part of a HAP Service must contain the descriptors defined in Table 7-53 (page 139) only, all other descriptors are reserved. The descriptor values must not change while the accessory is paired with a controller.

Table 7-53: Bluetooth LE Characteristic Descriptors

Type	UUID	Required
Characteristic Instance ID (Custom Descriptor)	DC46F0FE-81D2-4616-B5D9-6ABDD796939A	Yes
HAP Characteristic Properties (Custom Descriptor)	NA	Yes
org.bluetooth.descriptor.gatt. characteristic_presentation_format	NA	Yes
org.bluetooth.descriptor.gatt. characteristic_user_description	NA	No
org.bluetooth.descriptor.gatt. client _characteristic_configuration	NA	No
org.bluetooth.descriptor.gatt. valid_range	NA	No
Step Value (Custom Descriptor)	NA	No
Valid Values (Custom Descriptor)	NA	No
Valid Values Range (Custom Descriptor)	NA	No

7.4.5.1 Characteristic Format Types

The format types for characteristics follow the format types defined by GATT and "10 Apple-defined Characteristics" (page 177).

Apple defined characteristics shall follow the format types defined for the characteristics and shall not include the Characteristic Presentation Format Descriptor descriptors to redefine the format type. Section "10 Apple-defined Characteristics" (page 177) defines some characteristics as having a format type of TLV8. In these cases, the data payload may represent one or more TLV8 values as defined by the characteristic.

7.4.5.2 Characteristic with Additional Authorization Data

Some of the characteristics defined in "10 Apple-defined Characteristics" (page 177) supports additional authorization data by default when included as part of particular services. The characteristic and service combination that support additional authorization data by default are "10.32 Target Door State" (page 191) in the "9.4 Garage Door Opener" (page 154) and the "10.19 Lock Target State" (page 186) in the "9.4 Garage Door Opener" (page 154) and

"9.7 Lock Mechanism" (page 155). Other characteristics can also support additional authorization data. Characteristics that support additional authorization data indicates this support by having the "7.4.4.6 Characteristic Metadata Descriptors" (page 137) descriptor with the bit set to indicate Characteristic Supports Additional Authorization Data as defined in Table 7-50 (page 137). An accessory with a characteristic supporting additional authorization data may not always require additional authorization data to be present. When additional authorization data is present it is included as an additional type to the TLV8 format along with the Value and Remote TLV types.

If the accessory receives a write request without the authorization data on characteristics that currently requires it, the accessory must return the status code Insufficient Authorization from Table 7-37 (page 123). If the accessory received a write request with additional authorization data when it does not require it, the accessory shall ignore the authorization data and accept the write request.

7.4.5.3 Valid Values Descriptor

This descriptor is used to specify the valid values supported by the characteristic. This descriptor shall only be used for Apple defined characteristics of format unit8, that support a list of defined enumeration values.

The format for this descriptor is an n-byte sequence of UINT8 in ascending order where each byte contains one of the valid value supported by the accessory.

This descriptor shall only be used when the characteristic supports only a subset of the defined values.

7.4.5.4 Valid Values Range Descriptor

This descriptor is used to specify a range of valid values supported by the characteristic. This descriptor shall only be used for Apple defined characteristics of format uint8 that supports a list of list of defined enumeration values.

This descriptor shall be used when it is more efficient to specify a list of valid values as a range rather than individual values.

This descriptor can be used in conjunction with the "Valid Values Descriptor".

The format for this descriptor is a list of 2-byte sequences of UINT8's in ascending order where the first byte is the starting value and the second byte is the ending value of the valid values range.

This descriptor shall only be used when the characteristic supports only a subset of the defined values.

Omission of both the "Valid Values Descriptor" and "Valid Values Range Descriptor" implies that the characteristic supports all the defined enumeration values for this characteristic.

7.4.6 HAP Notifications

- Characteristics that support HAP notifications must have the 'Indicate' property set on their Bluetooth LE characteristic.
- The characteristic must support the Bluetooth SIG's Characteristic Client Configuration Descriptor (see Bluetooth Core Specification) to toggle Bluetooth LE indications after a secure session is established.
- The accessory must also support the "7.4.1.8 Global State Number (GSN)" (page 130) increment rules as described below.

7.4.6.1 Connected Events

When a controller is connected to an accessory the controller may choose to register for Bluetooth LE indications for some characteristics. When an update occurs on those characteristics, the accessory must send a zero-length indication to the connected controller for that characteristic. The zero-length indication must only be sent to a controller with an established security session. The controller will then perform a HAP-secured read of the characteristic.

Indications shall not be sent to the controller that changed the value of a characteristic if the change was due to a write from the controller.

After the first characteristic change on characteristics that are registered for Bluetooth LE indications in the current connected state, the GSN shall also be incremented by 1 and reflected in the subsequent advertisements after the current connection is disconnected. The GSN must increment only once for multiple characteristic changes while in the current connected state.

7.4.6.2 Broadcasted Events

When the value associated with a characteristic that is configured for broadcast notification changes while in a disconnected state the Global State Number (GSN) must be incremented for every such change and reflected in the Encrypted advertisement payload. The accessory must use the configured advertising interval for the characteristic to broadcast the changed value for at least 3 seconds.

If a controller connects to the accessory before the completion of the 3 second advertising period the accessory should abort the encrypted advertisement and continue with its regular advertisement at the regular advertising period after the controller disconnects.

If no controller connects to the accessory within the 3 second broadcast period then the accessory must fall back to the Disconnected Events advertisement rule with its current GSN as specified in "7.4.6.3 Disconnected Events" (page 141).

For additional characteristic changes before the completion of the 3 second period and before a controller connection, the GSN should be updated again and the accessory must reflect the latest changed characteristic value in its encrypted advertisement and continue to broadcast for an additional 3 seconds from the last change.

7.4.6.3 Disconnected Events

When the value associated with a characteristic that notifies a change while in a disconnected state changes, the Global State Number (GSN) may be incremented and reflected in the Bluetooth LE advertisement data of the accessory as specified in "7.4.2.1 HAP BLE Regular Advertisement Format" (page 131).

The GSN should increment only once for multiple characteristic value changes while in in disconnected state until the accessory state changes from disconnected to connected.

Disconnected events are always enabled when the characteristic supports it. Characteristics supporting disconnected events must support "7.4.6.1 Connected Events" (page 141) and "7.4.6.2 Broadcasted Events" (page 141).

The accessory should use disconnected events only to reflect important state changes in the accessory that might need to be notified to the user. For example a contact sensor state change or a current door state change indications are good use cases for disconnected events, whereas a temperature sensor must not use disconnected events to reflect every change in the temperature reading. After updating the GSN as specified in Section "7.4.2.1 HAP BLE

Regular Advertisement Format" (page 131) in the disconnected state the accessory must use a 20 ms advertising interval for at least 3 seconds.

When the broadcast events are enabled for characteristic that support disconnected events the accessory shall first follow the broadcasted events procedure and then fall back to disconnected events as specified in "7.4.6.2 Broadcasted Events" (page 141).

7.4.7 Security

Authenticated encryption for HAP over Bluetooth LE is provided by two mechanisms: "7.4.7.1 Pairing" (page 142) and "7.4.7.2 Session Security" (page 142).

7.4.7.1 Pairing

Pairing establishes a bi-directional, cryptographic relationship between a controller and an accessory. "5 Pairing" (page 38) describes the process for pairing a controller and an accessory over GATT and features two main components:

- Pair Setup: The process that creates a valid pairing between a controller and an accessory. This process exchanges long-term cryptographic public keys.
- Pair Verify: The process that verifies a valid pairing and establishes a shared secret used to secure the HAP session.

7.4.7.2 Session Security

Once the controller and accessory have an ephemeral shared secret, both the controller and the accessory use the shared secret to derive the read and write keys for session security:

```
AccessoryToControllerKey = HKDF-SHA-512 of
    InputKey   = <Pair Verify shared secret>
    Salt       = "Control-Salt"
    Info       = "Control-Read-Encryption-Key"
    OutputSize = 32 bytes

ControllerToAccessoryKey = HKDF-SHA-512 of
    InputKey   = <Pair Verify shared secret>
    Salt       = "Control-Salt"
    Info       = "Control-Write-Encryption-Key"
    OutputSize = 32 bytes
```

The controller and accessory use the derived keys in the following manner:

Table 7-54: Derived Key Usage

	Encryption Key	Decryption Key
Accessory	AccessoryToControllerKey	ControllerToAccessoryKey
Controller	ControllerToAccessoryKey	AccessoryToControllerKey

Each message is secured with the AEAD algorithm AEAD_CHACHA20_POLY1305 as specified in Section 2.8 of RFC 7539. The 32-bit fixed-common part of the 96-bit nonce is all zeros: `00 00 00 00`.

Accessories must tear down the security session after a Bluetooth LE disconnection occurs. Accessories must support multiple iterations of Pair Verify on a single Bluetooth LE connection, e.g. a controller connects, performs Pair Verify, and a later time in the same connection, performs Pair Verify again. When a new Pair Verify request is received, the accessory must tear down its security session and respond to any pending transactions with an error.

7.4.7.3 Broadcast Encryption Key Generation

When requested by paired controller with an established secure session the accessory shall derive the broadcast encryption key using the following parameters.

```
Broadcast-Encryption-Key = HKDF-SHA-512 of
InputKey   = <Current Session shared secret>
Salt       = <Controller's Ed25519 long term public key>
Info       = ”Broadcast-Encryption-Key”
OutputSize = 32 bytes
```

The broadcasted advertisement's encrypted and authenticated payload are secured with the AEAD algorithm AEAD_CHACHA20_POLY13 as specified in Section 2.8 of RFC 7539 with the following parameters.

(Encrypted-Advertisement-Payload, AuthTag) = ChaCha20-Poly1305(Broadcast-Encryption-Key, Nonce = GSN (padded with zeros), AAD = <48 bit accessory advertising identifier>, Msg = <12 bytes of advertisement payload to encrypt (2 byte GSN, 2 byte Instance Id, 8 bytes for value with optional padding)>)

4 byte AuthTag appended to the encrypted advertisement payload must be the first 4 bytes of the 16 byte AuthTag in the original order.

7.4.7.4 Broadcast Encryption Key expiration and refresh

There shall be only one broadcast encryption key in use on the accessory at any point in time. When the controller re-generates a new broadcast encryption using the "7.3.5.9 HAP Protocol Configuration Procedure" (page 123) the previous key shall be discarded and only the newly generated key shall be used for further broadcast notifications.

The broadcast encryption key shall expire automatically and must be discarded by the accessory after 32,767 (2^15 - 1) increments in GSN after the current broadcast key was generated. The controller will normally refresh the broadcast key to ensure that the key does not expire automatically on the accessory. The broadcast encryption key shall also expire on firmware updates and factory reset.

Characteristic changes while in a broadcast encryption key expired state shall not use broadcasted events and must fall back to disconnected/connected events until the controller has re-generated a new broadcast encryption key and re-registered charactersitics for broadcasted notification.

7.4.7.5 Securing HAP PDUs

When a secure session is established all HAP PDU’s are secured using the HAP session security. For example, if HAP Request PDU of size ’n’ is to be secured using HAP session security, the actual data received by the GATT server must be the following:

```
<n: encrypted HAP Request PDU’s>
<16:authTag>
```

The authTag must be appended to the encrypted value and must be sent as part of the same GATT message. The HAP server must verify and decrypt the received payload and process the HAP PDU.

7.4.8 Firmware Update Requirements

In addition to ”3.4 Firmware Updates” (page 30) the following additional requirements are applicable for Bluetooth accessories

- Bluetooth LE interface must support firmware updates over Bluetooth LE.
- Accessories supporting random static Bluetooth LE device address must use a new Bluetooth LE device address after a firmware update.

7.5 Testing Bluetooth LE Accessories

1. Accessory must support only the HAP-BLE 2.x advertisement format.
2. Accessory must include Accessory Information Service and Protocol Information Service with the required mandatory characteristics.
3. The Protocol Version characteristic value in the Protocol Information Service must match the value specified by this specification.
4. The Compatible Version (CV) in the advertisement packet must match the value specified in this specification.
5. BLE 2.0 accessories must not support HAP-BLE 1.0 procedures (read/write etc)
6. Accessory must handle malformed PDUs and fail the request.
7. Accessory must handle fragmented PDUs.
8. Accessory should be able to generate fragmented PDUs.
9. Accessory must accept new HAP procedure even if the previous procedure was not completed.

10. Accessory must support only one HAP procedure on a characteristic at any point in time.

11. Accessory should support parallel HAP procedures on different characteristics.

12. Accessory must reject GATT Read Requests on a HAP characteristic if it was not preceded by an GATT Write Request with the same transaction ID at most 10 seconds prior.

13. Accessory must indicate that Security Class characteristics require HAP-Characteristic-Timed-Write using the additional HAP characteristic property.

14. Accessory must return the same transaction ID in the HAP response that was part of the HAP request.

15. Accessory must reject HAP-Characteristic-Write-Request on characteristics that require timed writes.

16. Accessory must reject HAP-Characteristic-Write/Read/Timed-Write/Execute-Write Request to characteristics with instance ID that does not match the characteristic's instance ID.

17. Accessory must reject HAP-Characteristic-Execute-Write request after TTL and discard the queued HAP-Timed-Write request.

18. Accessory must not allow the accessory firmware to be downgraded.

19. Accessory must support a controller refreshing the security session.

20. Accessory must maintain the idle security session for a maximum of 30 seconds. After 30 seconds of inactivity without any HAP Transaction, the bluetooth link must be disconnected from the accessory.

21. Accessory must drop any HAP procedure continuation across security sessions (new session or refreshes).

22. Accessory must support HAP-Characteristic-Signature-Read-Request/Response for each HAP characteristic published.

23. Accessory must support the HAP-Characteristic-Signature-Read procedure at any time after a a secure session is established.

24. App for accessory firmware update should check and warn user of iOS version requirement on all controllers for the selected firmware.

25. Packaging for HAP-BLE 2.x accessories should indicate the minimum required iOS version.

26. Packaging/Documentation for HAP-BLE 2.x accessories should indicate minimum Apple TV version and tvOS version required for remote access.

27. Accessory must increment the config number after a firmware update.

28. Accessory must maintain the service and characteristic instance IDs for unchanged characteristics and services after a firmware update.

29. Accessory must not re-use an instance id from a service or characteristic that was removed from the current firmware update.

30. Accessory must tear down the security session when the Bluetooth LE link disconnects.

31. Accessory must support multiple iterations of pair verify on the same Bluetooth LE connection.

32. Accessories must only indicate disconnected event support on characteristic whose change may require user attention.

33. Changes on characteristics that do not support disconnected events must not change the global state number.

34. Multiple changes to characteristic supporting disconnected events white in disconnected state or in connected state must change state number only once.

35. Accessories must generate a new random Unique Identifier after every factory reset.

36. Accessories must not include the characteristic metadata in the GATT database.

37. Accessories must support characteristic signature read procedure with and without a secure session.

38. Accessories that include characteristic metadata must return the characteristic metadata descriptors in the characteristic signature read response.

39. Accessories must implement a 10 second HAP procedure timeout, all HAP procedures including Pair-Verify and Pair-Resume (with the exception of Pair-Setup) must complete within 10 seconds, if a procedure fails to complete within the procedure timeout the accessory must drop the security session and also drop the Bluetooth link.

40. After a Bluetooth link is established the first HAP procedure must begin within 10 seconds. Accessories must drop the Bluetooth Link if the controller fails to start a HAP procedure within 10 seconds of establishing the Bluetooth link.

8 HomeKit Accessory Protocol over iCloud

8.1 Overview

HomeKit Accessory Protocol (HAP) over iCloud enables a controller to communicate with an accessory remotely.

8.1.1 Setup

Before accessories may communicate over iCloud they must be associated with the HomeKit iCloud Service and paired with an admin controller via a local Pair Setup, see "5.6 Pair Setup" (page 41). All setup communication must occur over a local HAP Secure Session, see "5.7 Pair Verify" (page 49).

8.1.1.1 Activation

To communicate with the HomeKit iCloud Service, accessories must receive a HomeKit client certificate from an admin controller before being able to communicate with the HomeKit iCloud Service. During activation accessories must perform an MFi Challenge and generate a key pair. This operation is performed on behalf of the accessory by an admin controller communicating to the HomeKit iCloud Service.

1. The admin controller requests the accessory to sign a challenge with its Apple Authentication Coprocessor.
2. The accessory signs the challenge using its Apple Authentication Coprocessor.
3. The accessory requests the MFi Certificate from its Apple Authentication Coprocessor.
4. The accessory generates a prime256v1 Elliptic Curve private and public key.
5. The accessory replies to the admin controller with the MFi Challenge Response, MFi Certificate, and prime256v1 public key.
6. The admin controller requests the MFI Challenge Response be verified against the MFi Challenge and MFi Certificate by the HomeKit iCloud Service.
7. The admin controller requests a HomeKit client certificate based on the accessory's prime256v1 public key.
8. The admin controller sends the HomeKit client certificate to the accessory.
9. The accessory stores the prime256v1 private key and HomeKit client certificate in long-term secure storage.

8.1.1.2 Pairing

An admin controller will pair an accessory with the HomeKit iCloud Service before communication between the controller and accessory via HAP over iCloud may commence. To pair an accessory with the HomeKit iCloud Service the admin controller performs the following operations:

1. The admin controller sends the accessory the location of the `HomeKit bag` which contains information required by the accessory to connect to the HomeKit iCloud Service.
2. The accessory stores the `HomeKit bag` location to long-term secure storage.
3. The accessory fetches the `HomeKit bag` and connects to the HomeKit iCloud server at the hostname specified in the bag using the HomeKit client certificate.
4. The accessory requests an iCloud accessory identifier from the HomeKit iCloud server.
5. The accessory sends its iCloud accessory identifier to the admin controller.
6. The controller sends its iCloud controller identifier to the accessory.
7. The accessory requests a pairing token from the HomeKit iCloud server.
8. The accessory sends the pairing token to the admin controller.
9. The admin controller requests an access token from the HomeKit iCloud server in exchange for the pairing token provided by the accessory.
10. The admin controller sends the access token to the accessory.
11. The accessory stores the access token and controller identifier to long-term secure storage.
12. The accessory begins accepting requests from the newly paired controller.
13. The newly paired controller stores its controller identifier and access token in long-term secure storage.

8.1.2 Messaging

Once an accessory has been paired via HAP over iCloud, messages may be relayed between controllers and accessories. There are two types of messages for HAP over iCloud, commands and notifications.

8.1.2.1 Commands

Commands are controller-initiated request-response messages used for controlling the accessory. To initiate a command, the controller performs the following operations:

1. If required, initiate a HAP Security Session via the HomeKit iCloud server.
2. If required, encrypt the request payload using the HAP Security Session.
3. Serialize the request.
4. Fragment the request based on the MTU of the HomeKit iCloud server.
5. Send the fragmented packets to the accessory via the HomeKit iCloud server.

The accessory performs the following operations on receipt of the request:

1. Reassemble the fragmented request.
2. Deserialize the request.
3. If required, decrypt the request payload using the HAP Security Session.
4. Process the request and generate response payload.
5. If required, encrypt the response payload using the HAP Security Session.
6. Serialize the response.
7. Fragment the response based on the MTU of the HomeKit iCloud server.
8. Send the fragmented packets to the controller via the HomeKit iCloud server.

The controller performs the following operations on receipt of the response:

1. Reassemble the fragmented response.
2. Deserialize the response.
3. If required, decrypt the response payload using the HAP Security Session.
4. Process the response.

8.1.2.2 Notifications

Notifications are accessory-initiated, unsolicited responses used for notifying the controller of changes to the accessory. Notifications via HAP over iCloud do not support encryption or fragmentation. To send a notification, the accessory performs the following operations:

1. Serialize the notification request.
2. Add the access token and controller identifier for each controller to notify.
3. Send the notification request to the controllers via the HomeKit iCloud server.

Upon receipt of the notification request, a controller may determine to query some information about the notification via a command, see "8.1.2.1 Commands" (page 148).

8.1.3 Managing Additional Controllers

Controllers will have an access token provided by the HomeKit iCloud Service to communicate with an accessory.

8.1.3.1 Enabling Access for a Controller

Controllers which are paired with an accessory must first be enabled before they communicate with the accessory via iCloud. An admin controller enables the additional controller with the following operations:

1. The admin controller requests a consent token from the additional controller.
2. The additional controller requests a consent token from the HomeKit iCloud Service.
3. The additional controller replies to the admin with the consent token and its iCloud controller identifier.
4. The admin controller requests an access token for the additional controller.
5. The admin controller sends the access token and iCloud controller identifier to the accessory.
6. The accessory stores the access token and controller identifier to long-term secure storage.
7. The accessory begins accepting requests from the newly enabled controller.
8. The admin controller sends the access token to the newly enabled controller.
9. The newly enabled controller stores its controller identifier and access token to long-term secure storage.

8.1.3.2 Disabling Access for a Controller

Before controllers may be unpaired from an accessory their access with the accessory via iCloud will be disabled. An admin controller disables a controller from an accessory with the following operations:

1. The admin controller requests the accessory disable the controller.
2. The accessory removes the requested controller's access token and controller identifier from long-term secure storage.
3. The accessory stops accepting requests from the newly disabled controller.
4. The admin controller requests that the disabled controller removes its access token to the accessory.
5. The disabled controller removes its access token for the accessory from long-term secure storage.

8.2 Requirements

Accessories supporting HomeKit Accessory Protocol over iCloud have the following requirements:

8.2.1 IP Requirements

- The accessory must support IPv4.
- The accessory must act as a DHCPv4 client.
- The accessory may support static IPv4.

8.2.2 TCP Requirements

- The accessory must support at least two simultaneous TCP connections for communication with HomeKit iCloud services.

8.2.3 TLS Requirements

- The accessory must support TLS 1.2 as defined in RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2.
- The accessory must support the TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 cipher suite as defined in RFC 5289: TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM).
- The accessory must support the ALPN extension as defined in RFC 7301: Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension.

8.2.4 DNS Requirements

- The accessory must use the DNS server address provided to it during DHCP negotiation and must not use DNS server address for any other source.

8.2.5 HTTP Requirements

- The accessory must be HTTP/2 compliant as defined in RFC 7540: Hypertext Transfer Protocol Version 2 (HTTP/2).
- The accessory must support HTTP Caching.

8.2.6 Application Requirements

- The accessory must broadcast its firmware version in the Accessory Information service, see "9.1 Accessory Information" (page 153).

8.2.7 Factory Reset Operation Requirements

If a factory reset operation is performed on the accessory, it must:

- Delete the `Relay Accessory Certificate` and `Relay Accessory LTSK` from long-term secure storage.
- Delete the `HomeKit Accessory Bag URL`.
- Delete the `Access Token` from long-term secure storage.
- Delete the `Controller Identifier` from long-term secure storage.
- Delete the `Accessory Identifier`.
- Transition the HomeKit Accessory Protocol over iCloud Enabled characteristics to `False`.
- Transition the HomeKit Accessory Protocol over iCloud State characteristics to `Unactivated`.

8.3 More Details

Please contact your Wi-Fi or Ethernet chipset provider to find out when support for this feature will be available.

9 Apple-defined Services

9.1 Accessory Information

Every accessory must expose a single instance of the Accessory Information service with the following definition. The values of Manufacturer, Model, Name and Serial Number must be persistent through the lifetime of the accessory.

Any other Apple-defined characteristics added to this service must only contain one or more of the following permissions: Paired Read or Notify. Custom characteristics added to this service must only contain one or more of the following permissions: Paired Read, Notify, or Broadcast. All other permissions are not permitted.

Property	Value
UUID	0000003E-0000-1000-8000-0026BB765291
Type	public.hap.service.accessory-information
Required Characteristics	"10.14 Identify" (page 183) "10.21 Manufacturer" (page 186) "10.22 Model" (page 187) "10.24 Name" (page 187) "10.31 Serial Number" (page 190) "10.10 Firmware Revision" (page 181)
Optional Characteristics	"10.11 Hardware Revision" (page 182) "10.84 Accessory Flags" (page 227)

9.2 HAP Protocol Information

Every accessory must expose a single instance of the HAP protocol information. For a bridge accessory, only the primary HAP accessory object must contain this service. The "10.37 Version" (page 193) value is transport dependent. Refer to "7.4.3.1 BLE Protocol Version Characteristic" (page 134) for BLE protocol version. Refer to "6.6.3 IP Protocol Version" (page 75) for IP protocol version.

Property	Value
UUID	000000A2-0000-1000-8000-0026BB765291
Type	public.hap.service.protocol.information.service
Required Characteristics	"10.37 Version" (page 193)

9.3 Fan

This service describes a fan.

Property	Value
UUID	00000040-0000-1000-8000-0026BB765291
Type	public.hap.service.fan
Required Characteristics	"10.26 On" (page 188)
Optional Characteristics	"10.28 Rotation Direction" (page 189) "10.29 Rotation Speed" (page 189) "10.24 Name" (page 187)

9.4 Garage Door Opener

This service describes a garage door opener that controls a single door. If a garage has more than one door, then each door should have its own Garage Door Opener Service.

Property	Value
UUID	00000041-0000-1000-8000-0026BB765291
Type	public.hap.service.garage-door-opener
Required Characteristics	"10.6 Current Door State" (page 179) "10.32 Target Door State" (page 191) "10.25 Obstruction Detected" (page 188)
Optional Characteristics	"10.16 Lock Current State" (page 184) "10.19 Lock Target State" (page 186) "10.24 Name" (page 187)

9.5 Lightbulb

This service describes a lightbulb.

Property	Value
UUID	00000043-0000-1000-8000-0026BB765291
Type	public.hap.service.lightbulb
Required Characteristics	"10.26 On" (page 188)
Optional Characteristics	"10.4 Brightness" (page 178) "10.13 Hue" (page 183) "10.24 Name" (page 187) "10.30 Saturation" (page 190) "10.107 Color Temperature" (page 238)

9.6 Lock Management

The HomeKit Lock Management Service is designed to expose deeper interaction with a Lock device.

Property	Value
UUID	00000044-0000-1000-8000-0026BB765291
Type	public.hap.service.lock-management
Required Characteristics	"10.15 Lock Control Point" (page 184) "10.37 Version" (page 193)
Optional Characteristics	"10.20 Logs" (page 186) "10.3 Audio Feedback" (page 178) "10.18 Lock Management Auto Security Timeout" (page 185) "10.2 Administrator Only Access" (page 177) "10.17 Lock Last Known Action" (page 185) "10.6 Current Door State" (page 179) "10.23 Motion Detected" (page 187)

9.7 Lock Mechanism

The HomeKit Lock Mechanism Service is designed to expose and control the physical lock mechanism on a device.

Property	Value
UUID	00000045-0000-1000-8000-0026BB765291
Type	public.hap.service.lock-mechanism
Required Characteristics	”10.16 Lock Current State” (page 184) ”10.19 Lock Target State” (page 186)
Optional Characteristics	”10.24 Name” (page 187)

9.8 Outlet

This service describes a power outlet.

Property	Value
UUID	00000047-0000-1000-8000-0026BB765291
Type	public.hap.service.outlet
Required Characteristics	”10.26 On” (page 188) ”10.27 Outlet In Use” (page 188)
Optional Characteristics	”10.24 Name” (page 187)

9.9 Switch

This service describes a binary switch.

Property	Value
UUID	00000049-0000-1000-8000-0026BB765291
Type	public.hap.service.switch
Required Characteristics	”10.26 On” (page 188)
Optional Characteristics	”10.24 Name” (page 187)

9.10 Thermostat

This service describes a thermostat.

Property	Value
UUID	0000004A-0000-1000-8000-0026BB765291
Type	public.hap.service.thermostat
Required Characteristics	”10.7 Current Heating Cooling State” (page 180) ”10.33 Target Heating Cooling State” (page 191) ”10.9 Current Temperature” (page 181) ”10.35 Target Temperature” (page 193) ”10.36 Temperature Display Units” (page 193)
Optional Characteristics	”10.5 Cooling Threshold Temperature” (page 178) ”10.8 Current Relative Humidity” (page 180) ”10.12 Heating Threshold Temperature” (page 182) ”10.24 Name” (page 187) ”10.34 Target Relative Humidity” (page 192)

9.11 Air Quality Sensor

This service describes an air quality sensor. ”10.70 Air Quality” (page 210) refers to the cumulative air quality recorded by the accessory which may be based on multiple sensors present.

This service requires iOS 9 or later and is updated in iOS 10.

Property	Value
UUID	0000008D-0000-1000-8000-0026BB765291
Type	public.hap.service.sensor.air-quality
Required Characteristics	”10.70 Air Quality” (page 210)
Optional Characteristics	”10.24 Name” (page 187) ”10.99 Ozone Density” (page 234) ”10.100 Nitrogen Dioxide Density” (page 235) ”10.101 Sulphur Dioxide Density” (page 235) ”10.102 PM2.5 Density” (page 236) ”10.103 PM10 Density” (page 236) ”10.104 VOC Density” (page 236) ”10.54 Status Active” (page 202) ”10.56 Status Fault” (page 203) ”10.59 Status Tampered” (page 204) ”10.58 Status Low Battery” (page 204)

9.12 Security System

This service describes a security system service.

This service requires iOS 9 or later.

Property	Value
UUID	0000007E-0000-1000-8000-0026BB765291
Type	public.hap.service.security-system
Required Characteristics	"10.40 Security System Current State" (page 195) "10.41 Security System Target State" (page 196)
Optional Characteristics	"10.24 Name" (page 187) "10.63 Security System Alarm Type" (page 206) "10.56 Status Fault" (page 203) "10.59 Status Tampered" (page 204)

9.13 Carbon Monoxide Sensor

This service describes a Carbon Monoxide Sensor.

This service requires iOS 9 or later.

Property	Value
UUID	0000007F-0000-1000-8000-0026BB765291
Type	public.hap.service.sensor.carbon-monoxide
Required Characteristics	"10.43 Carbon Monoxide Detected" (page 197)
Optional Characteristics	"10.24 Name" (page 187) "10.54 Status Active" (page 202) "10.56 Status Fault" (page 203) "10.59 Status Tampered" (page 204) "10.58 Status Low Battery" (page 204) "10.65 Carbon Monoxide Level" (page 207) "10.66 Carbon Monoxide Peak Level" (page 208)

9.14 Contact Sensor

This service describes a Contact Sensor.

This service requires iOS 9 or later.

Property	Value
UUID	00000080-0000-1000-8000-0026BB765291
Type	public.hap.service.sensor.contact
Required Characteristics	”10.44 Contact Sensor State” (page 197)
Optional Characteristics	”10.24 Name” (page 187) ”10.54 Status Active” (page 202) ”10.56 Status Fault” (page 203) ”10.59 Status Tampered” (page 204) ”10.58 Status Low Battery” (page 204)

9.15 Door

This service describes a motorized door.

This service requires iOS 9 or later.

Property	Value
UUID	00000081-0000-1000-8000-0026BB765291
Type	public.hap.service.door
Required Characteristics	”10.47 Current Position” (page 198) ”10.61 Target Position” (page 205) ”10.52 Position State” (page 201)
Optional Characteristics	”10.24 Name” (page 187) ”10.49 Hold Position” (page 199) ”10.25 Obstruction Detected” (page 188)

9.16 Humidity Sensor

This service describes a Humidity Sensor.

This service requires iOS 9 or later.

Property	Value
UUID	00000082-0000-1000-8000-0026BB765291
Type	public.hap.service.sensor.humidity
Required Characteristics	"10.8 Current Relative Humidity" (page 180)
Optional Characteristics	"10.24 Name" (page 187) "10.54 Status Active" (page 202) "10.56 Status Fault" (page 203) "10.59 Status Tampered" (page 204) "10.58 Status Low Battery" (page 204)

9.17 Leak Sensor

This service describes a Leak Sensor.

This service requires iOS 9 or later.

Property	Value
UUID	00000083-0000-1000-8000-0026BB765291
Type	public.hap.service.sensor.leak
Required Characteristics	"10.50 Leak Detected" (page 200)
Optional Characteristics	"10.24 Name" (page 187) "10.54 Status Active" (page 202) "10.56 Status Fault" (page 203) "10.59 Status Tampered" (page 204) "10.58 Status Low Battery" (page 204)

9.18 Light Sensor

This service describes a Light Sensor.

This service requires iOS 9 or later.

Property	Value
UUID	00000084-0000-1000-8000-0026BB765291
Type	public.hap.service.sensor.light
Required Characteristics	"10.45 Current Ambient Light Level" (page 198)
Optional Characteristics	"10.24 Name" (page 187) "10.54 Status Active" (page 202) "10.56 Status Fault" (page 203) "10.59 Status Tampered" (page 204) "10.58 Status Low Battery" (page 204)

9.19 Motion Sensor

This service describes a Motion Sensor.

This service requires iOS 9 or later.

Property	Value
UUID	00000085-0000-1000-8000-0026BB765291
Type	public.hap.service.sensor.motion
Required Characteristics	"10.23 Motion Detected" (page 187)
Optional Characteristics	"10.24 Name" (page 187) "10.54 Status Active" (page 202) "10.56 Status Fault" (page 203) "10.59 Status Tampered" (page 204) "10.58 Status Low Battery" (page 204)

9.20 Occupancy Sensor

This service describes an Occupancy Sensor.

This service requires iOS 9 or later.

Property	Value
UUID	00000086-0000-1000-8000-0026BB765291
Type	public.hap.service.sensor.occupancy
Required Characteristics	"10.51 Occupancy Detected" (page 200)
Optional Characteristics	"10.24 Name" (page 187) "10.54 Status Active" (page 202) "10.56 Status Fault" (page 203) "10.59 Status Tampered" (page 204) "10.58 Status Low Battery" (page 204)

9.21 Smoke Sensor

This service describes a Smoke detector Sensor.

This service requires iOS 9 or later.

Property	Value
UUID	00000087-0000-1000-8000-0026BB765291
Type	public.hap.service.sensor.smoke
Required Characteristics	"10.55 Smoke Detected" (page 202)
Optional Characteristics	"10.24 Name" (page 187) "10.54 Status Active" (page 202) "10.56 Status Fault" (page 203) "10.59 Status Tampered" (page 204) "10.58 Status Low Battery" (page 204)

9.22 Stateless Programmable Switch

This service describes a stateless programmable switch.

The following rules apply to a stateless programmable switch accessory:

- Each physical switch on the accessory must be represented by a unique instance of this service.
- If there are multiple instances of this service on the accessory, they must be linked to a "9.38 Service Label" (page 172).
- If there are multiple instances of this service on the accessory, "10.105 Service Label Index" (page 237) is a required characteristic.

- "10.105 Service Label Index" (page 237) value for each instance of this service linked to the same "9.38 Service Label" (page 172) must be unique.
- The User visible label on the physical accessory should match the "10.106 Service Label Namespace" (page 237) described by the accessory.
- If there is only one instance of this service on the accessory, "9.38 Service Label" (page 172) is not required and consequently "10.105 Service Label Index" (page 237) must not be present.

This service requires iOS 10.3 or later.

Property	Value
UUID	00000089-0000-1000-8000-0026BB765291
Type	public.hap.service.stateless-programmable-switch
Required Characteristics	"10.53 Programmable Switch Event" (page 201)
Optional Characteristics	"10.24 Name" (page 187) "10.105 Service Label Index" (page 237)

9.23 Temperature Sensor

This service describes a Temperature Sensor.

This service requires iOS 9 or later.

Property	Value
UUID	0000008A-0000-1000-8000-0026BB765291
Type	public.hap.service.sensor.temperature
Required Characteristics	"10.9 Current Temperature" (page 181)
Optional Characteristics	"10.24 Name" (page 187) "10.54 Status Active" (page 202) "10.56 Status Fault" (page 203) "10.58 Status Low Battery" (page 204) "10.59 Status Tampered" (page 204)

9.24 Window

This service describes a motorized window.

This service requires iOS 9 or later.

Property	Value
UUID	0000008B-0000-1000-8000-0026BB765291
Type	public.hap.service.window
Required Characteristics	"10.47 Current Position" (page 198) "10.61 Target Position" (page 205) "10.52 Position State" (page 201)
Optional Characteristics	"10.24 Name" (page 187) "10.49 Hold Position" (page 199) "10.25 Obstruction Detected" (page 188)

9.25 Window Covering

This service describes motorized window coverings or shades - examples include shutters, blinds, awnings etc.

This service requires iOS 9 or later.

Property	Value
UUID	0000008C-0000-1000-8000-0026BB765291
Type	public.hap.service.window-covering
Required Characteristics	"10.61 Target Position" (page 205) "10.47 Current Position" (page 198) "10.52 Position State" (page 201)
Optional Characteristics	"10.24 Name" (page 187) "10.49 Hold Position" (page 199) "10.46 Current Horizontal Tilt Angle" (page 198) "10.60 Target Horizontal Tilt Angle" (page 205) "10.48 Current Vertical Tilt Angle" (page 199) "10.62 Target Vertical Tilt Angle" (page 206) "10.25 Obstruction Detected" (page 188)

9.26 Battery Service

This service describes a battery service.

This service requires iOS 9 or later.

Property	Value
UUID	00000096-0000-1000-8000-0026BB765291
Type	public.hap.service.battery
Required Characteristics	"10.42 Battery Level" (page 196) "10.64 Charging State" (page 207) "10.58 Status Low Battery" (page 204)
Optional Characteristics	"10.24 Name" (page 187)

9.27 Carbon Dioxide Sensor

This service describes a Carbon Dioxide Sensor.

This service requires iOS 9 or later.

Property	Value
UUID	00000097-0000-1000-8000-0026BB765291
Type	public.hap.service.sensor.carbon-dioxide
Required Characteristics	"10.67 Carbon Dioxide Detected" (page 208)
Optional Characteristics	"10.24 Name" (page 187) "10.54 Status Active" (page 202) "10.56 Status Fault" (page 203) "10.59 Status Tampered" (page 204) "10.58 Status Low Battery" (page 204) "10.68 Carbon Dioxide Level" (page 209) "10.69 Carbon Dioxide Peak Level" (page 209)

9.28 Camera RTP Stream Management

A Camera RTP Stream Management service allows description of the supported audio and video codecs and parameters supported by the accessory as well as configuration and control of the RTP session to stream the audio/video stream to a controller. The service has the following definition:

This service requires iOS 10 or later.

Property	Value
UUID	00000110-0000-1000-8000-0026BB765291
Type	public.hap.service.camera-rtp-stream-management
Required Characteristics	"10.71 Streaming Status" (page 210) "10.72 Supported Video Stream Configuration" (page 211) "10.73 Supported Audio Stream Configuration" (page 214) "10.74 Supported RTP Configuration" (page 215) "10.75 Setup Endpoints" (page 216) "10.76 Selected RTP Stream Configuration" (page 220)

The supported Video/Audio Input/Output Configuration characteristics allow an IP camera accessory to describe the supported audio/video codec and source parameters.

The Selected Stream Configuration characteristic is a control point characteristic that a controller will use to set up an RTP session for streaming audio/video. The value written to this characteristic selects the audio/video/RTP configuration to be used for the streaming session.

9.29 Microphone

A Microphone service is used to control the sourcing of the input audio – primarily through a microphone.

This service requires iOS 10 or later.

Property	Value
UUID	00000112-0000-1000-8000-0026BB765291
Type	public.hap.service.microphone
Required Characteristics	"10.78 Mute" (page 224)
Optional Characteristics	"10.24 Name" (page 187) "10.77 Volume" (page 224)

9.30 Speaker

A Speaker service is to use to control the audio output settings on a speaker device.

This service requires iOS 10 or later.

Property	Value
UUID	00000113-0000-1000-8000-0026BB765291
Type	public.hap.service.speaker
Required Characteristics	"10.78 Mute" (page 224)
Optional Characteristics	"10.24 Name" (page 187) "10.77 Volume" (page 224)

9.31 Doorbell

The Doorbell Service describes a doorbell and is the primary service of the Video Doorbell Profile.

This service requires iOS 10 or later.

Property	Value
UUID	00000121-0000-1000-8000-0026BB765291
Type	public.hap.service.doorbell
Required Characteristics	"10.53 Programmable Switch Event" (page 201)
Optional Characteristics	"10.24 Name" (page 187) "10.77 Volume" (page 224) "10.4 Brightness" (page 178)

9.32 Fan v2

This service describes a fan.

If the fan service is included in air purifiers accessories, "10.94 Current Fan State" (page 232) and "10.93 Target Fan State" (page 231) are required characteristics.

This service requires iOS 10.3 or later.

Property	Value
UUID	000000B7-0000-1000-8000-0026BB765291
Type	public.hap.service.fanv2
Required Characteristics	"10.95 Active" (page 232)
Optional Characteristics	"10.24 Name" (page 187) "10.94 Current Fan State" (page 232) "10.93 Target Fan State" (page 231) "10.28 Rotation Direction" (page 189) "10.29 Rotation Speed" (page 189) "10.96 Swing Mode" (page 233) "10.85 Lock Physical Controls" (page 227)

9.33 Slat

This service describes a slat which tilts on a vertical or a horizontal axis. "10.97 Current Tilt Angle" (page 233) and "10.98 Target Tilt Angle" (page 234) may be included in this service if the user can set the slats to a particular tilt angle.

"10.96 Swing Mode" (page 233) implies that the slats can swing automatically (e.g. vents on a fan).

This service requires iOS 10.3 or later.

Property	Value
UUID	000000B9-0000-1000-8000-0026BB765291
Type	public.hap.service.vertical-slat
Required Characteristics	"10.87 Current Slat State" (page 228) "10.88 Slat Type" (page 229)
Optional Characteristics	"10.24 Name" (page 187) "10.96 Swing Mode" (page 233) "10.97 Current Tilt Angle" (page 233) "10.98 Target Tilt Angle" (page 234)

9.34 Filter Maintenance

This service can be used to describe maintenance operations for a filter.

This service requires iOS 10.3 or later.

Property	Value
UUID	000000BA-0000-1000-8000-0026BB765291
Type	public.hap.service.filter-maintenance
Required Characteristics	"10.90 Filter Change Indication" (page 230)
Optional Characteristics	"10.24 Name" (page 187) "10.89 Filter Life Level" (page 229) "10.91 Reset Filter Indication" (page 230)

9.35 Air Purifier

This service describes an air purifier. An air purifier accessory can have additional linked services such as:

- "9.34 Filter Maintenance" (page 168) service(s) to describe one or more air filters.
- "9.11 Air Quality Sensor" (page 157) services to describe air quality sensors.
- "9.32 Fan v2" (page 167) service to describe a fan which can be independently controlled
- "9.33 Slat" (page 168) service to control vents

If "9.32 Fan v2" (page 167) is included as a linked service in an air purifier accessory:

- Changing "10.95 Active" (page 232) characteristic on the "9.35 Air Purifier" (page 169) must result in corresponding change to "10.95 Active" (page 232) characteristic on the "9.32 Fan v2" (page 167).
- Changing "10.95 Active" (page 232) characteristic on the "9.32 Fan v2" (page 167) from `Inactive` to `Active` does not require the "10.95 Active" (page 232) on the "9.35 Air Purifier" (page 169) to change. This enables `Fan Only` mode on air purifier.
- Changing "10.95 Active" (page 232) characteristic on the "9.32 Fan v2" (page 167) from `Active` to `Inactive` must result in the "10.95 Active" (page 232) on the "9.35 Air Purifier" (page 169) to change to `Inactive`.

An air purifier accessory service may include "10.29 Rotation Speed" (page 189) to control fan speed if the fan cannot be independently controlled.

This service requires iOS 10.3 or later.

Property	Value
UUID	000000BB-0000-1000-8000-0026BB765291
Type	public.hap.service.air-purifier
Required Characteristics	"10.95 Active" (page 232) "10.86 Current Air Purifier State" (page 228) "10.92 Target Air Purifier State" (page 231)
Optional Characteristics	"10.24 Name" (page 187) "10.29 Rotation Speed" (page 189) "10.96 Swing Mode" (page 233) "10.85 Lock Physical Controls" (page 227)

9.36 Heater Cooler

This service can be used to describe either of the following:

- a heater
- a cooler
- a heater and a cooler

A heater/cooler accessory may have additional:

- "9.32 Fan v2" (page 167) service to describe a fan which can be independently controlled
- "9.33 Slat" (page 168) service to control vents

A heater must include "10.12 Heating Threshold Temperature" (page 182). A cooler must include "10.5 Cooling Threshold Temperature" (page 178).

A heater/cooler accessory service may include "10.29 Rotation Speed" (page 189) to control fan speed if the fan cannot be independently controlled.

This service requires iOS 10.3 or later.

Property	Value
UUID	000000BC-0000-1000-8000-0026BB765291
Type	public.hap.service.heater-cooler
Required Characteristics	”10.95 Active” (page 232) ”10.9 Current Temperature” (page 181) ”10.108 Current Heater Cooler State” (page 238) ”10.109 Target Heater Cooler State” (page 239)
Optional Characteristics	”10.24 Name” (page 187) ”10.29 Rotation Speed” (page 189) ”10.36 Temperature Display Units” (page 193) ”10.96 Swing Mode” (page 233) ”10.5 Cooling Threshold Temperature” (page 178) ”10.12 Heating Threshold Temperature” (page 182) ”10.85 Lock Physical Controls” (page 227)

9.37 Humidifier Dehumidifier

This service can be used to describe either of the following:

- an air humidifier
- an air dehumidifier
- an air humidifier and an air dehumidifier

An air humidifier/dehumidifier accessory may have additional:

- ”9.32 Fan v2” (page 167) service to describe a fan which can be independently controlled
- ”9.33 Slat” (page 168) service to control vents

A dehumidifier must include ”10.113 Relative Humidity Dehumidifier Threshold” (page 242). A humidifier must include ”10.114 Relative Humidity Humidifier Threshold” (page 242).

A humidifier/dehumidifier accessory service may include ”10.29 Rotation Speed” (page 189) to control fan speed if the fan cannot be independently controlled.

This service requires iOS 10.3 or later.

Property	Value
UUID	000000BD-0000-1000-8000-0026BB765291
Type	public.hap.service.humidifier-dehumidifier
Required Characteristics	"10.95 Active" (page 232) "10.8 Current Relative Humidity" (page 180) "10.110 Current Humidifier Dehumidifier State" (page 240) "10.111 Target Humidifier Dehumidifier State" (page 240)
Optional Characteristics	"10.24 Name" (page 187) "10.113 Relative Humidity Dehumidifier Threshold" (page 242) "10.114 Relative Humidity Humidifier Threshold" (page 242) "10.29 Rotation Speed" (page 189) "10.96 Swing Mode" (page 233) "10.112 Water Level" (page 241) "10.85 Lock Physical Controls" (page 227)

9.38 Service Label

This service describes label scheme.

This service requires iOS 10.3 or later.

Property	Value
UUID	000000CC-0000-1000-8000-0026BB765291
Type	public.hap.service.service-label
Required Characteristics	"10.106 Service Label Namespace" (page 237)

9.39 Irrigation System

This service describes an irrigation system. This service must be present on an irrigation systems which supports on-device schedules or supports a top-level "10.95 Active" (page 232) control across multiple valves.

A sprinkler system accessory may be:

- a combination of "9.39 Irrigation System" (page 172) service on a bridge accessory with a collection of one or more "9.40 Valve" (page 173) services (with "10.119 Valve Type" (page 245) set to `Irrigation`) as bridged accessories (The bridge accessory is typically connected to each valve using wires). OR

- a combination of "9.39 Irrigation System" (page 172) service with a collection of one or more linked "9.40 Valve" (page 173) services (with "10.119 Valve Type" (page 245) set to `Irrigation`) (The valves are collocated e.g. hose based system). OR

- a combination of "9.40 Valve" (page 173) service(s) with "10.119 Valve Type" (page 245) set to `Irrigation` (e.g., a system with one or more valves which does not support scheduling)

An irrigation system is set to `Active` when the system is enabled. When one of the valves is set to `In Use`, the irrigation system must be set to in use.

An accessory which includes this service must include the "10.117 Set Duration" (page 244) in the "9.40 Valve" (page 173).

An irrigation system accessory which does not auto detect the number of valves it is connected to and requires user to provide this information must include the "10.120 Is Configured" (page 245) in the "9.40 Valve" (page 173).

"10.118 Remaining Duration" (page 244) on this service implies the total remaining duration to water across all the valves.

This service requires iOS 11.2 or later.

Property	Value
UUID	000000CF-0000-1000-8000-0026BB765291
Type	public.hap.service.irrigation-system
Required Characteristics	"10.95 Active" (page 232) "10.115 Program Mode" (page 242) "10.116 In Use" (page 243)
Optional Characteristics	"10.118 Remaining Duration" (page 244) "10.24 Name" (page 187) "10.56 Status Fault" (page 203)

9.40 Valve

This service describes accessories like irrigation valves or water outlets. A valve is set to `In Use` when there are fluid flowing through the valve.

If an accessory has this service with "10.119 Valve Type" (page 245) set to Irrigation it must include the "10.117 Set Duration" (page 244) and "10.118 Remaining Duration" (page 244) characteristic on the "9.40 Valve" (page 173) service.

"10.105 Service Label Index" (page 237) must be present on each instance of this service if the accessory consists of:

- a bridge accessory (the "9.38 Service Label" (page 172) service must be included here) which includes multiple bridged accessories each with "9.40 Valve" (page 173) service.

- an accessory (the "9.38 Service Label" (page 172) service must be included here) which includes multiple linked "9.40 Valve" (page 173) services

If an accessory has this service with "10.105 Service Label Index" (page 237) included, the default "10.24 Name" (page 187) must be empty string unless user has already assigned a name to this valve before accessory is HomeKit paired. In such a case, the default name should be the user configured name for this valve.

"10.120 Is Configured" (page 245) must be present on each instance of this service if the accessory is used in an irrigation system or shower system where all valves may not be configured to use (e.g., depending on physical wire connection).

This characteristic requires iOS 11.2 or later.

Setting the value of "10.95 Active" (page 232) to `Active` on this service must result in "9.39 Irrigation System" (page 172) bridge to be set to `Active` if this service is used in context of an Irrigation system.

This service requires iOS 11.2 or later.

Property	Value
UUID	000000D0-0000-1000-8000-0026BB765291
Type	public.hap.service.valve
Required Characteristics	"10.95 Active" (page 232) "10.116 In Use" (page 243) "10.119 Valve Type" (page 245)
Optional Characteristics	"10.117 Set Duration" (page 244) "10.118 Remaining Duration" (page 244) "10.120 Is Configured" (page 245) "10.105 Service Label Index" (page 237) "10.56 Status Fault" (page 203) "10.24 Name" (page 187)

9.41 Faucet

This service describes accessories like faucets or shower heads. This service must only be included when an accessory has either a linked "9.36 Heater Cooler" (page 170) with single linked "9.40 Valve" (page 173) service or multiple linked "9.40 Valve" (page 173) services (with/without "9.36 Heater Cooler" (page 170) service) to describe water outlets. This service serves as a top level service for such accessories.

A faucet which supports heating of water must include "9.36 Heater Cooler" (page 170) and "9.40 Valve" (page 173) service as linked services. An accessory which supports one or multiple water outlets and heating of water through a common temperature control, must include "9.36 Heater Cooler" (page 170) and "9.40 Valve" (page 173) service(s) as linked services to the faucet service.

Setting the value of "10.95 Active" (page 232) to `InActive` on this service must turn off the faucet accessory. The accessory must retain the state of "10.95 Active" (page 232) characteristics on any linked "9.40 Valve" (page 173) services when the "10.95 Active" (page 232) on this service is toggled. The accessory must set the value of "10.95 Active" (page 232) to `InActive` of any linked "9.36 Heater Cooler" (page 170) service when the "10.95 Active" (page 232) on this service is set to `InActive`.

This service requires iOS 11.2 or later.

Property	Value
UUID	000000D7-0000-1000-8000-0026BB765291
Type	public.hap.service.faucet
Required Characteristics	"10.95 Active" (page 232)
Optional Characteristics	"10.56 Status Fault" (page 203) "10.24 Name" (page 187)

9.42 Target Control Management

This service manages the configuration for a remote accessory and allows it to indicate the supported configuration. This service must be marked as primary service for an accessory whose primary functionality is to act as a target controller (e.g., remote for Apple TV).

This service requires iOS 12 or later.

Property	Value
UUID	00000122-0000-1000-8000-0026BB765291
Type	public.hap.service.target-control-management
Required Characteristics	"10.121 Target Control Supported Configuration" (page 246) "10.122 Target Control List" (page 247)

9.43 Target Control

This service handles the control of a selected target from the remote accessory. If an accessory can support control of multiple concurrent Apple TVs at the same time without requiring the user to select an Apple TV on the remote UI, it must expose multiple instances of this service.

This service requires iOS 12 or later.

Property	Value
UUID	00000125-0000-1000-8000-0026BB765291
Type	public.hap.service.target-control
Required Characteristics	"10.124 Active Identifier" (page 250) "10.95 Active" (page 232) "10.123 Button Event" (page 249)
Optional Characteristics	"10.24 Name" (page 187)

9.44 Audio Stream Management

This service manages the configuration for audio input from the accessory, if applicable.

This service requires iOS 12 or later.

Property	Value
UUID	00000127-0000-1000-8000-0026BB765291
Type	public.hap.service.audio-stream-management
Required Characteristics	"10.73 Supported Audio Stream Configuration" (page 214) "10.125 Selected Audio Stream Configuration" (page 250)

9.45 Data Stream Transport Management

This service allows configuration and management of the transport data stream. The version field must be set to "1.0".

This service requires iOS 12 or later.

Property	Value
UUID	00000129-0000-1000-8000-0026BB765291
Type	public.hap.service.data-stream-transport-management
Required Characteristics	"10.126 Supported Data Stream Transport Configuration" (page 251) "10.127 Setup Data Stream Transport" (page 252) "10.37 Version" (page 193)

9.46 Siri

This service allows configuration and management of Siri.

This service must be linked to the "9.44 Audio Stream Management" (page 176) and "9.45 Data Stream Transport Management" (page 176). This service requires iOS 12 or later.

Property	Value
UUID	00000133-0000-1000-8000-0026BB765291
Type	public.hap.service.siri
Required Characteristics	"10.128 Siri Input Type" (page 253)

10 Apple-defined Characteristics

10.1 Overview

This section specifies pre-defined characteristics and their properties. Each characteristic has a unique UUID.

Accessories must use Apple-defined characteristics to expose functionality of the accessory if they are available; e.g., a temperature sensor must use the Apple-defined "10.9 Current Temperature" (page 181) characteristic rather than defining its own custom characteristic to expose the same functionality.

Note that all the characteristics with the format "bool" must expose/accept a Boolean value expressed as one of the following: true, false, 0 (false), or 1 (true).

Additionally, if a set of valid values is specified, for example as in "10.36 Temperature Display Units" (page 193), the characteristic must only expose/accept these values.

10.1.1 Overriding Properties

Each of the characteristics in this chapter is defined by Apple. Some include default properties about the characteristic, such as a minimum value or maximum value. The following properties of an Apple-defined characteristic may be modified in order to better fit the specific application:

- Minimum Value
- Maximum Value
- Step Value
- Maximum Length
- Maximum Data Length

An example of modifying the default properties of an Apple-defined characteristic is using the Current Temperature characteristic, which has a default minimum value of 0°C, for an outdoor temperature sensor. Rather than creating a new characteristic, the accessory can simply set the "minimum value" property to -30°C.

The accessory must specify the new values used as part of the metadata information when modifying the default properties. Percentages are excluded from properties that can be overridden.

10.2 Administrator Only Access

This mode implies that when enabled, the device will only accept administrator access.

Property	Value
UUID	00000001-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.administrator-only-access
Permissions	Paired Read, Paired Write, Notify
Format	bool

10.3 Audio Feedback

This characteristic describes whether audio feedback (e.g. a beep, or other external sound mechanism) is enabled.

Property	Value
UUID	00000005-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.audio-feedback
Permissions	Paired Read, Paired Write, Notify
Format	bool

10.4 Brightness

This characteristic describes a perceived level of brightness, e.g. for lighting, and can be used for backlights or color. The value is expressed as a percentage (%) of the maximum level of supported brightness.

Property	Value
UUID	00000008-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.brightness
Permissions	Paired Read, Paired Write, Notify
Format	int
Minimum Value	0
Maximum Value	100
Step Value	1
Unit	percentage

10.5 Cooling Threshold Temperature

This characteristic describes the cooling threshold in Celsius for devices that support simultaneous heating and cooling. The value of this characteristic represents the 'maximum temperature' that must be reached before cooling is turned on.

For example, if the "10.33 Target Heating Cooling State" (page 191) is set to "Auto" and the current temperature goes above the 'maximum temperature', then the cooling mechanism should turn on to decrease the current temperature until the 'minimum temperature' is reached.

Property	Value
UUID	0000000D-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.temperature.cooling-threshold
Permissions	Paired Read, Paired Write, Notify
Format	float
Minimum Value	10
Maximum Value	35
Step Value	0.1
Unit	celsius

10.6 Current Door State

This characteristic describes the current state of a door.

Property	Value
UUID	0000000E-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.door-state.current
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	4
Step Value	1
Valid Values	0 "Open. The door is fully open." 1 "Closed. The door is fully closed." 2 "Opening. The door is actively opening." 3 "Closing. The door is actively closing." 4 "Stopped. The door is not moving, and it is not fully open nor fully closed." 5-255 "Reserved"

10.7 Current Heating Cooling State

This characteristic describes the current mode of a device that supports cooling or heating its environment, e.g. a thermostat is "heating" a room to 75 degrees Fahrenheit.

Property	Value
UUID	0000000F-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.heating-cooling.current
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	2
Step Value	1
Valid Values	0 "Off." 1 "Heat. The Heater is currently on." 2 "Cool. Cooler is currently on." 3-255 "Reserved"

10.8 Current Relative Humidity

This characteristic describes the current relative humidity of the environment that contains the device. The value is expressed in percentage (%).

Property	Value
UUID	00000010-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.relative-humidity.current
Permissions	Paired Read, Notify
Format	float
Minimum Value	0
Maximum Value	100
Step Value	1
Unit	percentage

10.9 Current Temperature

This characteristic describes the current temperature of the environment in Celsius irrespective of display units chosen in "10.36 Temperature Display Units" (page 193).

Property	Value
UUID	00000011-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.temperature.current
Permissions	Paired Read, Notify
Format	float
Minimum Value	0
Maximum Value	100
Step Value	0.1
Unit	celsius

10.10 Firmware Revision

This characteristic describes a firmware revision string x[.y[.z]] (e.g. "100.1.1"):

- <x> is the major version number, required.
- <y> is the minor version number, required if it is non-zero or if <z> is present.
- <z> is the revision version number, required if non-zero.

The firmware revision must follow the below rules:

- <x> is incremented when there is significant change. e.g.,1.0.0, 2.0.0, 3.0.0, etc.
- <y> is incremented when minor changes are introduced such as 1.1.0, 2.1.0, 3.1.0 etc.
- <z> is incremented when bug-fixes are introduced such as 1.0.1, 2.0.1, 3.0.1 etc.
- Subsequent firmware updates can have a lower <y> version only if <x> is incremented
- Subsequent firmware updates can have a lower <z> version only if <x> or <y> is incremented
- Each number (major, minor and revision version) must not be greater than (2^32 -1)

The characteristic value must change after every firmware update.

Property	Value
UUID	00000052-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.firmware.revision
Permissions	Paired Read
Format	string

10.11 Hardware Revision

This characteristic describes a hardware revision string x[.y[.z]] (e.g. "100.1.1") and tracked when the board or components of the same accessory is changed :

- <x> is the major version number, required.
- <y> is the minor version number, required if it is non-zero or if <z> is present.
- <z> is the revision version number, required if non-zero.

The hardware revision must follow the below rules:

- <x> is incremented when there is significant change. e.g.,1.0.0, 2.0.0, 3.0.0, etc.
- <y> is incremented when minor changes are introduced such as 1.1.0, 2.1.0, 3.1.0 etc.
- <z> is incremented when bug-fixes are introduced such as 1.0.1, 2.0.1, 3.0.1 etc.
- Subsequent hardware updates can have a lower <y> version only if <x> is incremented
- Subsequent hardware updates can have a lower <z> version only if <x> or <y> is incremented

The characteristic value must change after every hardware update.

Property	Value
UUID	00000053-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.hardware.revision
Permissions	Paired Read
Format	string

10.12 Heating Threshold Temperature

This characteristic describes the heating threshold in Celsius for devices that support simultaneous heating and cooling. The value of this characteristic represents the 'minimum temperature' that must be reached before heating is turned on.

For example, if the "10.33 Target Heating Cooling State" (page 191) is set to "Auto" and the current temperature goes below the 'minimum temperature', then the heating mechanism should turn on to increase the current temperature until the 'minimum temperature' is reached.

Property	Value
UUID	00000012-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.temperature.heating-threshold
Permissions	Paired Read, Paired Write, Notify
Format	float
Minimum Value	0
Maximum Value	25
Step Value	0.1
Unit	celsius

10.13 Hue

This characteristic describes hue or color.

Property	Value
UUID	00000013-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.hue
Permissions	Paired Read, Paired Write, Notify
Format	float
Minimum Value	0
Maximum Value	360
Step Value	1
Unit	arcdegrees

10.14 Identify

This characteristic is used to cause the accessory to run its identify routine.

Only the "9.1 Accessory Information" (page 153) is allowed to contain the "identify" characteristic.

Property	Value
UUID	00000014-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.identify
Permissions	Paired Write
Format	bool

10.15 Lock Control Point

The device accepts writes to this characteristic to perform vendor-specific actions as well as those defined by the "9.6 Lock Management" (page 155) of the "11.2 Lock" (page 255) . For example, user management related functions should be defined and performed using this characteristic.

Property	Value
UUID	00000019-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.lock-management.control-point
Permissions	Paired Write
Format	tlv8

10.16 Lock Current State

The current state of the physical security mechanism (e.g. deadbolt).

Property	Value
UUID	0000001D-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.lock-mechanism.current-state
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	3
Step Value	1
Valid Values	0 "Unsecured" 1 "Secured" 2 "Jammed" 3 "Unknown" 4-255 "Reserved"

10.17 Lock Last Known Action

The last known action of the lock mechanism (e.g. deadbolt).

Property	Value
UUID	0000001C-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.lock-mechanism.last-known-action
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	8
Step Value	1
Valid Values	0 ”Secured using physical movement, interior” 1 ”Unsecured using physical movement, interior” 2 ”Secured using physical movement, exterior” 3 ”Unsecured using physical movement, exterior” 4 ”Secured with keypad” 5 ”Unsecured with keypad” 6 ”Secured remotely” 7 ”Unsecured remotely” 8 ”Secured with Automatic Secure timeout” 9-255 ”Reserved”

10.18 Lock Management Auto Security Timeout

A value greater than 0 indicates if the lock mechanism enters the unsecured state, it will automatically attempt to enter the secured state after n seconds, where n is the value provided in the write. A value of 0 indicates this feature is disabled.

Property	Value
UUID	0000001A-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.lock-management.auto-secure-timeout
Permissions	Paired Read, Paired Write, Notify
Format	uint32
Unit	seconds

10.19 Lock Target State

The target state of the physical security mechanism (e.g. deadbolt).

Property	Value
UUID	0000001E-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.lock-mechanism.target-state
Permissions	Paired Read, Paired Write, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 ”Unsecured” 1 ”Secured” 2-255 ”Reserved”

10.20 Logs

Read from this characteristic to get timestamped logs from the device. The data is in TLV8 format as defined by the associated service profile. The ”9.6 Lock Management” (page 155), for example, defines its own specific structure for the log data.

Property	Value
UUID	0000001F-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.logs
Permissions	Paired Read, Notify
Format	tlv8

10.21 Manufacturer

This characteristic contains the name of the company whose brand will appear on the accessory, e.g., ”Acme”.

Property	Value
UUID	00000020-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.manufacturer
Permissions	Paired Read
Format	string
Maximum Length	64

10.22 Model

This characteristic contains the manufacturer-specific model of the accessory, e.g. "A1234". The minimum length of this characteristic must be 1.

Property	Value
UUID	00000021-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.model
Permissions	Paired Read
Format	string
Maximum Length	64

10.23 Motion Detected

This characteristic indicates if motion (e.g. a person moving) was detected.

Property	Value
UUID	00000022-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.motion-detected
Permissions	Paired Read, Notify
Format	bool

10.24 Name

This characteristic describes a name and must not be a null value.

Property	Value
UUID	00000023-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.name
Permissions	Paired Read
Format	string
Maximum Length	64

10.25 Obstruction Detected

This characteristic describes the current state of an obstruction sensor, such as one that is used in a garage door. If the state is true then there is an obstruction detected.

Property	Value
UUID	00000024-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.obstruction-detected
Permissions	Paired Read, Notify
Format	bool

10.26 On

This characteristic represents the states for "on" and "off."

Property	Value
UUID	00000025-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.on
Permissions	Paired Read, Paired Write, Notify
Format	bool

10.27 Outlet In Use

This characteristic describes if the power outlet has an appliance e.g., a floor lamp, physically plugged in. This characteristic is set to `True` even if the plugged-in appliance is off.

Property	Value
UUID	00000026-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.outlet-in-use
Permissions	Paired Read, Notify
Format	bool

10.28 Rotation Direction

This characteristic describes the direction of rotation of a fan.

Property	Value
UUID	00000028-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.rotation.direction
Permissions	Paired Read, Paired Write, Notify
Format	int
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 ”Clockwise” 1 ”Counter-clockwise” 2-255 ”Reserved”

10.29 Rotation Speed

This characteristic describes the rotation speed of a fan.

Property	Value
UUID	00000029-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.rotation.speed
Permissions	Paired Read, Paired Write, Notify
Format	float
Minimum Value	0
Maximum Value	100
Step Value	1
Unit	percentage

10.30 Saturation

This characteristic describes color saturation.

Property	Value
UUID	0000002F-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.saturation
Permissions	Paired Read, Paired Write, Notify
Format	float
Minimum Value	0
Maximum Value	100
Step Value	1
Unit	percentage

10.31 Serial Number

This characteristic contains the manufacturer-specific serial number of the accessory, e.g. "1A2B3C4D5E6F". The length must be greater than 1.

Property	Value
UUID	00000030-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.serial-number
Permissions	Paired Read
Format	string
Maximum Length	64

10.32 Target Door State

This characteristic describes the target state of a door.

Property	Value
UUID	00000032-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.door-state.target
Permissions	Paired Read, Paired Write, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 "Open" 1 "Closed" 2-255 "Reserved"

10.33 Target Heating Cooling State

This characteristic describes the target mode of a device that supports heating/cooling, e.g. a thermostat.

Property	Value
UUID	00000033-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.heating-cooling.target
Permissions	Paired Read, Paired Write, Notify
Format	uint8
Minimum Value	0
Maximum Value	3
Step Value	1
Valid Values	0 "Off" 1 "Heat. If the current temperature is below the target temperature then turn on heating." 2 "Cool. If the current temperature is above the target temperature then turn on cooling." 3 "Auto. Turn on heating or cooling to maintain temperature within the heating and cooling threshold of the target temperature." 4-255 "Reserved"

10.34 Target Relative Humidity

This characteristic describes the target relative humidity that the device is actively attempting to reach. The value is expressed in percentage (%).

Property	Value
UUID	00000034-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.relative-humidity.target
Permissions	Paired Read, Paired Write, Notify
Format	float
Minimum Value	0
Maximum Value	100
Step Value	1
Unit	percentage

10.35 Target Temperature

This characteristic describes the target temperature in Celsius that the device is actively attempting to reach. For example, a thermostat cooling a room to "75" degrees Fahrenheit would set the target temperature value to "23.9".

Property	Value
UUID	00000035-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.temperature.target
Permissions	Paired Read, Paired Write, Notify
Format	float
Minimum Value	10.0
Maximum Value	38.0
Step Value	0.1
Unit	celsius

10.36 Temperature Display Units

This characteristic describes units of temperature used for presentation purposes (e.g. the units of temperature displayed on the screen).

Property	Value
UUID	00000036-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.temperature.units
Permissions	Paired Read, Paired Write, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 "Celsius" 1 "Fahrenheit" 2-255 "Reserved"

10.37 Version

This characteristic contains a version string.

Property	Value
UUID	00000037-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.version
Permissions	Paired Read
Format	string
Maximum Length	64

10.38 Air Particulate Density

This characteristic indicates the current air particulate matter density in micrograms/m^3.

This characteristic requires iOS 9 or later.

Property	Value
UUID	00000064-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.air-particulate.density
Permissions	Paired Read, Notify
Format	float
Minimum Value	0
Maximum Value	1000

10.39 Air Particulate Size

This characteristic indicates the size of air particulate matter in micrometers.

This characteristic requires iOS 9 or later.

Property	Value
UUID	00000065-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.air-particulate.size
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 "2.5 Micrometers" 1 "10 Micrometers" 2-255 "Reserved"

10.40 Security System Current State

This characteristic describes the state of a security system

This characteristic requires iOS 9 or later.

Property	Value
UUID	00000066-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.security-system-state.current
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	4
Step Value	1
Valid Values	0 "Stay Arm. The home is occupied and the residents are active. e.g. morning or evenings" 1 "Away Arm. The home is unoccupied" 2 "Night Arm. The home is occupied and the residents are sleeping" 3 "Disarmed" 4 "Alarm Triggered" 5-255 "Reserved"

10.41 Security System Target State

This characteristic describes the target state of the security system.

This characteristic requires iOS 9 or later.

Property	Value
UUID	00000067-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.security-system-state.target
Permissions	Paired Read, Paired Write, Notify
Format	uint8
Minimum Value	0
Maximum Value	3
Step Value	1
Valid Values	0 ”Stay Arm. The home is occupied and the residents are active. e.g. morning or evenings” 1 ”Away Arm. The home is unoccupied” 2 ”Night Arm. The home is occupied and the residents are sleeping” 3 ”Disarm” 4-255 ”Reserved”

10.42 Battery Level

This characteristic describes the current level of the battery.

This characteristic requires iOS 9 or later.

Property	Value
UUID	00000068-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.battery-level
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	100
Step Value	1
Unit	percentage

10.43 Carbon Monoxide Detected

This characteristic indicates if a sensor detects abnormal levels of Carbon Monoxide. Value should revert to 0 after the Carbon Monoxide levels drop to normal levels

This characteristic requires iOS 9 or later.

Property	Value
UUID	00000069-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.carbon-monoxide.detected
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 "Carbon Monoxide levels are normal" 1 "Carbon Monoxide levels are abnormal"

10.44 Contact Sensor State

This characteristic describes the state of a door/window contact sensor. A value of 0 indicates that the contact is detected. A value of 1 indicates that the contact is not detected.

This characteristic requires iOS 9 or later.

Property	Value
UUID	0000006A-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.contact-state
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 "Contact is detected" 1 "Contact is not detected"

10.45 Current Ambient Light Level

This characteristic indicates the current light level. The value is expressed in Lux units (lumens/m^2)

This characteristic requires iOS 9 or later.

Property	Value
UUID	0000006B-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.light-level.current
Permissions	Paired Read, Notify
Format	float
Minimum Value	0.0001
Maximum Value	100000
Unit	lux

10.46 Current Horizontal Tilt Angle

This characteristic describes the current angle of horizontal slats for accessories such as windows, fans, portable heater/coolers etc. This characteristic takes values between -90 and 90. A value of 0 indicates that the slats are rotated to a fully open position. A value of -90 indicates that the slats are rotated all the way in a direction where the user-facing edge is higher than the window-facing edge.

This characteristic requires iOS 9 or later.

Property	Value
UUID	0000006C-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.horizontal-tilt.current
Permissions	Paired Read, Notify
Format	int
Minimum Value	-90
Maximum Value	90
Step Value	1
Unit	arcdegrees

10.47 Current Position

This characteristic describes the current position of accessories. This characteristic can be used with doors, windows, awnings or window coverings. For windows and doors, a value of 0 indicates that a window (or door) is fully closed while a value of 100 indicates a fully open position. For blinds/shades/awnings, a value of 0 indicates a position that permits the least light and a value of 100 indicates a position that allows most light.

This characteristic requires iOS 9 or later.

Property	Value
UUID	0000006D-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.position.current
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	100
Step Value	1
Unit	percentage

10.48 Current Vertical Tilt Angle

This characteristic describes the current angle of vertical slats for accessories such as windows, fans, portable heater/coolers etc. This characteristic takes values between -90 and 90. A value of 0 indicates that the slats are rotated to be fully open. A value of -90 indicates that the slats are rotated all the way in a direction where the user-facing edge is to the left of the window-facing edge.

This characteristic requires iOS 9 or later.

Property	Value
UUID	0000006E-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.vertical-tilt.current
Permissions	Paired Read, Notify
Format	int
Minimum Value	-90
Maximum Value	90
Step Value	1
Unit	arcdegrees

10.49 Hold Position

This characteristic causes the service such as door or window covering to stop at its current position. A value of 1 must hold the state of the accessory. For e.g, the window must stop moving when this characteristic is written a value of 1. A value of 0 should be ignored.

A write to "10.61 Target Position" (page 205) characteristic will release the hold.

This characteristic requires iOS 9 or later.

Property	Value
UUID	0000006F-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.position.hold
Permissions	Paired Write
Format	bool

10.50 Leak Detected

This characteristic indicates if a sensor detected a leak (e.g. water leak, gas leak). A value of 1 indicates that a leak is detected. Value should return to 0 when leak stops.

This characteristic requires iOS 9 or later.

Property	Value
UUID	00000070-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.leak-detected
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 "Leak is not detected" 1 "Leak is detected"

10.51 Occupancy Detected

This characteristic indicates if occupancy was detected (e.g. a person present). A value of 1 indicates occupancy is detected. Value should return to 0 when occupancy is not detected.

This characteristic requires iOS 9 or later.

Property	Value
UUID	00000071-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.occupancy-detected
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 "Occupancy is not detected" 1 "Occupancy is detected"

10.52 Position State

This characteristic describes the state of the position of accessories. This characteristic can be used with doors, windows, awnings or window coverings for presentation purposes.

This characteristic requires iOS 9 or later.

Property	Value
UUID	00000072-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.position.state
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	2
Step Value	1
Valid Values	0 "Going to the minimum value specified in metadata" 1 "Going to the maximum value specified in metadata" 2 "Stopped" 3-255 "Reserved"

10.53 Programmable Switch Event

This characteristic describes an event generated by a programmable switch. Reading this characteristic must return the last event triggered for BLE. For IP accessories, the accessory must set the value of Paired Read to null(i.e. "value"

: null) in the attribute database. A read of this characteristic must always return a null value for IP accessories. The value must only be reported in the events ("ev") property.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	00000073-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.input-event
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	2
Step Value	1
Valid Values	0 "Single Press" 1 "Double Press" 2 "Long Press" 3-255 "Reserved"

10.54 Status Active

This characteristic describes an accessory's current working status. A value of true indicates that the accessory is active and is functioning without any errors.

This characteristic requires iOS 9 or later.

Property	Value
UUID	00000075-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.status-active
Permissions	Paired Read, Notify
Format	bool

10.55 Smoke Detected

This characteristic indicates if a sensor detects abnormal levels of smoke. A value of 1 indicates that smoke levels are abnormal. Value should return to 0 when smoke levels are normal.

This characteristic requires iOS 9 or later.

Property	Value
UUID	00000076-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.smoke-detected
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 ”Smoke is not detected” 1 ”Smoke is detected”

10.56 Status Fault

This characteristic describes an accessory which has a fault. A non-zero value indicates that the accessory has experienced a fault that may be interfering with its intended functionality. A value of 0 indicates that there is no fault.

This characteristic requires iOS 9 or later.

Property	Value
UUID	00000077-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.status-fault
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 ”No Fault” 1 ”General Fault”

10.57 Status Jammed

This characteristic describes an accessory which is in a jammed state. A status of 1 indicates that an accessory's mechanisms are jammed prevents it from functionality normally. Value should return to 0 when conditions that jam the accessory are rectified.

This characteristic requires iOS 9 or later.

Property	Value
UUID	00000078-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.status-jammed
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 "Not Jammed" 1 "Jammed"

10.58 Status Low Battery

This characteristic describes an accessory's battery status. A status of 1 indicates that the battery level of the accessory is low. Value should return to 0 when the battery charges to a level thats above the low threshold.

This characteristic requires iOS 9 or later.

Property	Value
UUID	00000079-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.status-lo-batt
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 "Battery level is normal" 1 "Battery level is low"

10.59 Status Tampered

This characteristic describes an accessory which has been tampered with. A status of 1 indicates that the accessory has been tampered with. Value should return to 0 when the accessory has been reset to a non-tampered state.

This characteristic requires iOS 9 or later.

Property	Value
UUID	0000007A-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.status-tampered
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 "Accessory is not tampered" 1 "Accessory is tampered with"

10.60 Target Horizontal Tilt Angle

This characteristic describes the target angle of horizontal slats for accessories such as windows, fans, portable heater/coolers etc.

This characteristic requires iOS 9 or later.

Property	Value
UUID	0000007B-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.horizontal-tilt.target
Permissions	Paired Read, Paired Write, Notify
Format	int
Minimum Value	-90
Maximum Value	90
Step Value	1
Unit	arcdegrees

10.61 Target Position

This characteristic describes the target position of accessories. This characteristic can be used with doors, windows, awnings or window coverings. For windows and doors, a value of 0 indicates that a window (or door) is fully closed while a value of 100 indicates a fully open position. For blinds/shades/awnings, a value of 0 indicates a position that permits the least light and a value of 100 indicates a position that allows most light.

This characteristic requires iOS 9 or later.

Property	Value
UUID	0000007C-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.position.target
Permissions	Paired Read, Paired Write, Notify
Format	uint8
Minimum Value	0
Maximum Value	100
Step Value	1
Unit	percentage

10.62 Target Vertical Tilt Angle

This characteristic describes the target angle of vertical slats for accessories such as windows, fans, portable heater/coolers etc.

This characteristic requires iOS 9 or later.

Property	Value
UUID	0000007D-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.vertical-tilt.target
Permissions	Paired Read, Paired Write, Notify
Format	int
Minimum Value	-90
Maximum Value	90
Step Value	1
Unit	arcdegrees

10.63 Security System Alarm Type

This characteristic describes the type of alarm triggered by a security system. A value of 1 indicates an 'unknown' cause. Value should revert to 0 when the alarm conditions are cleared.

This characteristic requires iOS 9 or later.

Property	Value
UUID	0000008E-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.security-system.alarm-type
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1

10.64 Charging State

This characteristic describes the charging state of a battery or an accessory.

This characteristic requires iOS 9 or later.

Property	Value
UUID	0000008F-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.charging-state
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	2
Step Value	1
Valid Values	0 ”Not Charging” 1 ”Charging” 2 ”Not Chargeable”

10.65 Carbon Monoxide Level

This characteristic indicates the Carbon Monoxide levels detected in parts per million (ppm).

This characteristic requires iOS 9 or later.

Property	Value
UUID	00000090-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.carbon-monoxide.level
Permissions	Paired Read, Notify
Format	float
Minimum Value	0
Maximum Value	100

10.66 Carbon Monoxide Peak Level

This characteristic indicates the highest detected level (ppm) of Carbon Monoxide detected by a sensor.

This characteristic requires iOS 9 or later.

Property	Value
UUID	00000091-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.carbon-monoxide.peak-level
Permissions	Paired Read, Notify
Format	float
Minimum Value	0
Maximum Value	100

10.67 Carbon Dioxide Detected

This characteristic indicates if a sensor detects abnormal levels of Carbon Dioxide. Value should revert to 0 after the Carbon Dioxide levels drop to normal levels.

This characteristic requires iOS 9 or later.

Property	Value
UUID	00000092-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.carbon-dioxide.detected
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 "Carbon Dioxide levels are normal" 1 "Carbon Dioxide levels are abnormal"

10.68 Carbon Dioxide Level

This characteristic indicates the detected level of Carbon Dioxide in parts per million (ppm).

This characteristic requires iOS 9 or later.

Property	Value
UUID	00000093-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.carbon-dioxide.level
Permissions	Paired Read, Notify
Format	float
Minimum Value	0
Maximum Value	100000

10.69 Carbon Dioxide Peak Level

This characteristic indicates the highest detected level (ppm) of carbon dioxide detected by a sensor.

This characteristic requires iOS 9 or later.

Property	Value
UUID	00000094-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.carbon-dioxide.peak-level
Permissions	Paired Read, Notify
Format	float
Minimum Value	0
Maximum Value	100000

10.70 Air Quality

This characteristic describes the subject assessment of air quality by an accessory.

This characteristic requires iOS 9 or later.

Property	Value
UUID	00000095-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.air-quality
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	5
Step Value	1
Valid Values	0 ”Unknown” 1 ”Excellent” 2 ”Good” 3 ”Fair” 4 ”Inferior” 5 ”Poor”

10.71 Streaming Status

A Streaming Status characteristic allows an IP Camera accessory to describe the status of the RTP Stream Management service.

This characteristic requires iOS 10 or later.

Property	Value
UUID	00000120-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.streaming-status
Permissions	Paired Read, Notify
Format	tlv8

The value of this characteristic is a tlv8 encoded list of supported parameters. The list of types for the TLVs are as follows:

Table 10-1: Streaming Status

Name	Type	Length	Description
Status	1	1	Status of the stream RTP management service 0 - Available 1 - In Use 2 - Unavailable 3 - 255 Reserved for use by Apple

10.72 Supported Video Stream Configuration

A Supported Video Stream Configuration characteristic allows an IP Camera accessory to describe the parameters supported for streaming video over an RTP session. Status characteristic allows an IP Camera accessory to describe the status of the RTP Stream Management service.

This characteristic requires iOS 10 or later.

Property	Value
UUID	00000114-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.supported-video-stream-configuration
Permissions	Paired Read
Format	tlv8

The value of this characteristic is a tlv8 encoded list of supported parameters. The list of types for the TLVs are as follows:

Table 10-2: Supported Video Stream Configuration

Name	Type	Length	Description
Video Codec Configuration	1	N	Codec information and the configurations supported for the codec. There is one TLV of this type per supported codec

An IP Camera supporting multiple video codecs must include one instance of the above TLV per supported video codec.

The Video Codec Configuration is encoded as a tlv8. The list of types for this TLV's value are as follows:

Table 10-3: Video Codec Configuration

Name	Type	Length	Description
Video Codec Type	1	N	Type of video codec: 0 - H.264 1 - 255 Reserved for use by Apple
Video Codec Parameters	2	N	Video Codec-specific parameters
Video Attributes	3	N	Video Attributes supported for the codec.

The Video Codec parameters TLV is encoded as a tlv8 - the list of types for the value for a H.264 codec are as follows:

Table 10-4: Video Codec Parameters

Name	Type	Length	Description
ProfileID	1	1	Type of H.264 Profile: 0 - Constrained Baseline Profile 1 - Main Profile Note:Interlaced coding (PicAFF, MBAFF) must not be used 2 - High Profile Note:Interlaced coding (PicAFF, MBAFF) must not be used 3 - 255 Vendor-specific One instance of this TLV must be present for each supported profile
Level	2	1	Profile support level: 0 - 3.1 1 - 3.2 2 - 4 3 - 255 Reserved for use by Apple
Packetization mode	3	1	Packetization Mode 0 - Non-interleaved mode 1 - 255 Reserved for use by Apple One instance of this TLV must be present for each supported mode
CVO Enabled	4	1	0 - CVO not supported 1 - CVO supported
CVO ID	5	1	ID for CVO RTP extension. This must be a value in the range [1-14]

The Video attributes allow the accessory to indicate supported resolutions, frame rates etc., This information is encoded as a tlv8- the list of types for the value of this TLV are as follows:

Table 10-5: Video Attributes

Name	Type	Length	Description
Image width	1	2	Image width in pixels
Image height	2	2	Image height in pixels
Frame rate	3	1	Maximum frame rate

An IP camera accessory supporting encoding video at different resolutions must include multiple instances of Video attributes TLV.

10.73 Supported Audio Stream Configuration

A Supported Audio Stream Configuration characteristic allows an accessory to indicate the the parameters supported for streaming audio (from a microphone and/or to a speaker) over an RTP session.

This characteristic requires iOS 10 or later.

Property	**Value**
UUID	00000115-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.supported-audio-configuration
Permissions	Paired Read
Format	tlv8

The value of this characteristic is a tlv8 encoded list of supported parameters. The list of types for the TLVs are as follows:

Table 10-6: Supported Audio Stream Configuration

Name	Type	Length	Description
Audio Codec Configuration	1	N	Codec information and the configurations supported for the codec
Comfort Noise support	2	1	Boolean, indicating support for Comfort Noise Codec

An IP Camera supporting multiple audio codecs must include one instance of the above TLV per supported audio codec. The Audio Codec Configuration is encoded as a tlv8. The list of types for this TLV are as follows:

Table 10-7: Audio Codecs

Name	Type	Length	Description
Codec type	1	2	Type of codec: 2 - AAC-ELD 3 - Opus 5 - AMR 6- AMR-WB 7- 255 Reserved for use by Apple
Audio Codec Parameters	2	N	Codec-specific parameters

An IP camera must support the RTP Times listed in the table below

Table 10-8: Mandatory RTP Time Values

RTP Time	Audio Codecs
20ms, 40ms, 60ms	AMR, AMR-WB, AAC-ELD 24K, Opus 16K, Opus 24K
30ms, 60ms	AAC-ELD 16K

The Audio Codec parameters are encoded as a tlv8 - the list of types are as follows:

Table 10-9: Audio Codec Parameters

Name	Type	Length	Description
Audio channels	1	1	Number of audio channels. Default is 1
Bit-rate	2	1	0 - Variable bit-rate 1- Constant bit-rate
Sample rate	3	1	0 - 8KHz 1 - 16 KHz 2 - 24 KHz 3 - Reserved for use by Apple
RTP time	4	1	Packet Time - Length of time represented by the media in a packet RFC 4566. Supported values - 20ms, 30ms, 40 ms & 60ms *Note: This TLV will only be presented in the Selected Audio* *Codec Parameters TLV*

10.74 Supported RTP Configuration

The Supported RTP Configuration characteristic allows a controller to specify the selected video attributes for a video RTP service to be used for streaming or other operation

This characteristic requires iOS 10 or later.

Property	Value
UUID	00000116-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.supported-rtp-configuration
Permissions	Paired Read
Format	tlv8

The value of this characteristic is a tlv8 encoded list of supported RTP parameters and their values. The list of types for the TLVs are as follows:

Table 10-10: Supported RTP Configuration

Name	Type	Length	Description
SRTP Crypto Suite	2	1	Supported SRTP Crypto Suite: 0 - AES_CM_128_HMAC_SHA1_80 1 - AES_256_CM_HMAC_SHA1_80 2 - Disabled 3 - 255 Reserved for use by Apple If multiple crypto suites are supported, multiple instances of this TLV should be present.

The controller will configure the selected RTP parameters using the selected stream configuration characteristic.

10.75 Setup Endpoints

The Setup Endpoints characteristic allows a controller to exchange IP address and port information with the IP camera.

This characteristic requires iOS 10 or later.

Property	**Value**
UUID	00000118-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.setup-endpoints
Permissions	Paired Read, Paired Write
Format	tlv8

The write value of this characteristic is a tlv8 encoded list of the following parameters:

Table 10-11: Setup Endpoints

Name	Type	Length	Description
Session ID	1	16	UUID identifying the session
Controller Address	3	N	Address of the controller for the streaming session
SRTP Parameters for Video	4	N	RTP parameters selected for the Video streaming session
SRTP Parameters for Audio	5	N	RTP parameters selected for the Audio streaming session

The Controller Address TLV has a tlv8 encoded value that provides the IP address, and the port for the streaming session. The list of types for the TLVs is as follows

Table 10-12: Controller Address

Name	Type	Length	Description
IP address version	1	1	Version of IP Address: 0 - IPv4 1 - IPv6
IP Address	2	N	IP address of the controller
Video RTP port	3	2	Receive port of the controller for the video stream of the RTP session
Audio RTP port	4	2	Receive port of the controller for the audio stream of the RTP session

Both the audio and video RTP port TLVs must be present even in case audio and video are multiplexed on the same port.

The SRTP parameters TLV has a tlv8 encoded value that provides the SRTP crypto-suite and keys for the streaming session. The accessory must populate a unique instance of this TLV for each supported SRTP Crypto Suite. When the SRTP Crypto Suite is set to 2 (disabled), a zero-length SRTP Master Key as well as a zero-length SRTP Master Salt must be presented.

The list of types for the TLVs is as follows:

Table 10-13: SRTP Crypto Suite

Name	Type	Length	Description
SRTP Crypto Suite	1	1	Supported SRTP Crypto Suite: 0 - AES_CM_128_HMAC_SHA1_80 1 - AES_256_CM_HMAC_SHA1_80 2 - Disabled 3 - 255 Reserved for use by Apple If multiple crypto suites are supported, multiple instances of this TLV should be present.
SRTP Master Key	2	16 or 32	Master key for the SRTP session: 16 - AES_CM_128_HMAC_SHA1_80 32 - AES_256_CM_HMAC_SHA1_80
SRTP Master Salt	3	14	Master salt for the SRTP session

If the accessory is able to parse the write command, it must respond with Success (i.e with a HTTP 204 No Content if all write(s) in the request succeed, or a HTTP 207 Multi-Status response including HAP status code indicating success for this write - in the context of a multi-write request).

After a write is issued to this characteristic by the controller, the next read of this characteristic indicates the status of the write command. The read response is encoded as tlv8 list with the following types:

Table 10-14: Read Response

Name	Type	Length	Description
Session Identifier	1	16	UUID identifying the session that the command applies to
Status	2	1	0 - Success 1 - Busy 2 - Error 3 - 255 Reserved for use by Apple
Accessory Address	3	N	Address of the IP camera for the streaming session.
SRTP Parameters for Video	4	N	RTP parameters selected for the video streaming session
SRTP Parameters for Audio	5	N	RTP parameters selected for the audio streaming session
SynchronizationSource for Video	6	4	SSRC for video RTP stream
SynchronizationSource for Audio	7	4	SSRC for audio RTP stream

The Accessory Address TLV has a tlv8 encoded value with the same types as described for Controller Address TLV above.

The IP Address Version in the Accessory Address TLV must be the same as the IP Address Version in the Controller Address TLV.

If the accessory receives a read request on this characteristic prior to any write request, the accessory must:

- Respond with HAP Success, i.e with a HTTP 200 OK if all read(s) in the request succeed or a HTTP 207 Multi-Status response including HAP status code indicating success for this read if some of the reads fail
- Include Status = Error in the corresponding TLV (omitting the session identifier)

If the accessory is unable to setup a stream because all possible streams are being used, it must:

- Respond to the following read request with HAP Success, i.e with a HTTP 200 OK if all read(s) in the request succeed, or a HTTP 207 Multi-Status response including HAP status code indicating success for this read if some of the reads fail
- Include Status = Busy in the corresponding TLV (including the session identifier)

10.76 Selected RTP Stream Configuration

The Setup Endpoints characteristic allows a controller to exchange IP address and port information The Selected RTP Stream Configuration characteristic is a control point characteristic that allows a controller to specify the selected video and audio attributes to be used for streaming audio and video from an IP camera accessory.

This characteristic requires iOS 10 or later.

Property	**Value**
UUID	00000117-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.selected-rtp-stream-configuration
Permissions	Paired Read, Paired Write
Format	tlv8

The write value of this characteristic is a tlv8 encoded list of selected audio/video/RTP parameters and their values. The list of types for the TLVs are as follows:

Table 10-15: Selected RTP Stream Configuration

Name	Type	Length	Description
Session Control	1	16	Session Control Command and Identifier
Selected Video Parameters	2	N	Video parameters selected for the streaming session
Selected Audio Parameters	3	N	Input Audio parameters selected for the streaming session

The Session Control Command TLV has a tlv8 encoded value that provides the response and the session identifier. The list of types for the TLVs is as follows:

Table 10-16: Session Control Command

Name	Type	Length	Description
Session Identifier	1	16	UUID identifying the session that identifies the command
Command	2	1	Session control command: 0 - End streaming session 1 - Start streaming session 2 - Suspend streaming session 3 - Resume streaming session 4 - Reconfigure streaming sessions 5 - 255 Reserved for use by Apple

The Selected Video parameters TLV has a tlv8 encoded value that provides the selected video attributes such as codec information, image resolution, frame rate, RTP parameters etc., The list of types for the TLVs is as follows:

Table 10-17: Selected Video Parameters

Name	Type	Length	Description
Selected Video Codec type	1	1	Type of video codec
Selected Video Codec parameters	2	N	Video Codec-specific parameters for the streaming session
Selected Video attributes	3	N	Video attributes selected for the streaming session
Selected Video RTP parameters	4	N	RTP parameters selected for the video streaming session

The encoding of Selected Video Codec Parameters and Selected Video Attributes TLVs is as described in "10.72 Supported Video Stream Configuration" (page 211)

The value of Video RTP parameters TLV is a tlv8 encoded list with the following types:

Table 10-18: Video RTP Parameters

Name	Type	Length	Description
Payload type	1	1	Type of video codec
SynchronizationSource for Video	2	4	SSRC for video stream
Maximum Bitrate	3	2	Maximum bit rate generated by the codec in kbps and averaged over 1 second
Min RTCP interval	4	4	Minimum RTCP interval in seconds formatted as a 4 byte little endian ieee754 floating point value
Max MTU	5	2	MTU that the IP camera must use to transmit Video RTP packets. This value will be populated only if the controller intends the camera to use a non-default value of the MTU

The Selected Audio parameters TLV has a tlv8 encoded value that provides the selected audio attributes such as codec information, number of audio channels, RTP parameters etc., The list of types for the TLVs is as follows:

Table 10-19: Selected Audio Parameters

Name	Type	Length	Description
Selected Audio Codec type	1	1	Type of codec: 0 - PCMU 1 - PCMA 2 - AAC-ELD 3 - Opus 4 - MSBC 5 - AMR 6 - AMR-WB 7 - 255 Reserved for use by Apple
Selected Audio Codec parameters	2	N	Audio codec specific parameters
Selected Audio RTP parameters	3	N	RTP parameters selected for the streaming session
Comfort Noise	4	1	Boolean. A value of 1 indicates that Comfort Noise has been selected and that both Camera and iOS device will both use Comfort Noise codec

The Selected Audio Codec Parameters TLV value has the same TLV types as described in "10.73 Supported Audio Stream Configuration" (page 214)

The value of Audio RTP parameters TLV is a tlv8 encoded list with the following types:

Table 10-20: Audio RTP Parameters

Name	Type	Length	Description
Payload type	1	1	Payload type as defined in RFC 3551
SynchronizationSource for Audio	2	4	SSRC for audio stream
Maximum Bitrate	3	2	Maximum bit rate generated by the codec in kbps and averaged over 1 second
Min RTCP interval	4	4	Minimum RTCP interval in seconds formatted as a 4 byte little endian ieee754 floating point value
Comfort Noise Payload Type	6	1	Only required when Comfort Noise is chosen in the Selected Audio Parameters TLV

10.77 Volume

A Volume characteristic allows the control of input or output volume of an audio input or output device respectively.

This characteristic requires iOS 10 or later.

Property	**Value**
UUID	00000119-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.volume
Permissions	Paired Write, Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	100
Step Value	1
Unit	percentage

The value of this characteristic indicates the percentage of the maximum volume supported by the service.

If the audio RTP service can support controlling the volume, this characteristic must support Paired Write permission as well.

10.78 Mute

A Mute characteristic allows the control of audio input or output device respectively.

This characteristic requires iOS 10 or later.

Property	Value
UUID	0000011A-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.mute
Permissions	Paired Write, Paired Read, Notify
Format	bool
Valid Values	0 ”Mute is Off / Audio is On” 1 ”Mute is On / There is no Audio”

If the audio RTP service can support muting, this characteristic must support Paired Write permission as well.

10.79 Night Vision

A Night Vision characteristic allows the control of night vision mode on a video RTP service.

This characteristic requires iOS 10 or later.

Property	Value
UUID	0000011B-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.night-vision
Permissions	Paired Write, Paired Read, Notify
Format	bool
Valid Values	0 ”Disable night-vision mode” 1 ”Enable night-vision mode”

10.80 Optical Zoom

A Digital Zoom characteristic allows the control of digital zoom of a video RTP service.

This characteristic requires iOS 10 or later.

Property	Value
UUID	0000011C-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.zoom-optical
Permissions	Paired Write, Paired Read, Notify
Format	float

The value of this characteristic represents the optical zoom setting of the camera service that is sourcing the input image.

10.81 Digital Zoom

A Digital Zoom characteristic allows the control of digital zoom of a video RTP service.

This characteristic requires iOS 10 or later.

Property	Value
UUID	0000011D-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.zoom-digital
Permissions	Paired Write, Paired Read, Notify
Format	float

The value of this characteristic represents the digital zoom multiplier to be applied on the image sourced by the video RTP service that is sourcing the input image.

10.82 Image Rotation

An Image Rotation characteristic allows the control of rotation of the image of a video RTP service.

This characteristic requires iOS 10 or later.

Property	Value
UUID	0000011E-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.image-rotation
Permissions	Paired Write, Paired Read, Notify
Format	float
Valid Values	0 ”No rotation” 90 ”Rotated 90 degrees to the right” 180 ”Rotated 180 degrees to the right (flipped vertically)” 270 ”Rotated 270 degrees to the right”

10.83 Image Mirroring

An Image Mirroring characteristic allows the control of mirroring state of the image of a video RTP service.

This characteristic requires iOS 10 or later.

Property	Value
UUID	0000011F-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.image-mirror
Permissions	Paired Write, Paired Read, Notify
Format	bool
Valid Values	0 "Image is not mirrored" 1 "Image is mirrored"
Unit	arcdegrees

10.84 Accessory Flags

When set indicates accessory requires additional setup. Use of Accessory Flags requires written approval by Apple in advance.

Property	Value
UUID	000000A6-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.accessory-properties
Permissions	Paired Read, Notify
Format	uint32
Valid Values	0x0001 (bit0) "Requires additional setup" 0x0002 - 0xFFFF "Reserved"

10.85 Lock Physical Controls

This characteristic describes a way to lock a set of physical controls on an accessory (eg. child lock).

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000A7-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.lock-physical-controls
Permissions	Paired Write,Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 ”Control lock disabled” 1 ”Control lock enabled”

10.86 Current Air Purifier State

This characteristic describes the current state of the air purifier.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000A9-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.air-purifier.state.current
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	2
Step Value	1
Valid Values	0 ”Inactive” 1 ”Idle” 2 ”Purifying Air”

10.87 Current Slat State

This characteristic describes the current state of the slats.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000AA-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.slat.state.current
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	2
Step Value	1
Valid Values	0 "Fixed" 1 "Jammed" 2 "Swinging"

10.88 Slat Type

This characteristic describes the type of the slats. If the slats can tilt on a horizontal axis, the value of this characteristic must be set to `Horizontal`. If the slats can tilt on a vertical axis, the value of this characteristic must be set to `Vertical`.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000C0-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.type.slat
Permissions	Paired Read
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 "Horizontal" 1 "Vertical"

10.89 Filter Life Level

This characteristic describes the current filter life level.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000AB-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.filter.life-level
Permissions	Paired Read, Notify
Format	float
Minimum Value	0
Maximum Value	100
Step Value	1

10.90 Filter Change Indication

This characteristic describes if a filter needs to be changed.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000AC-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.filter.change-indication
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 ”Filter does not need to be changed” 1 ”Filter needs to be changed”

10.91 Reset Filter Indication

This characteristic allows a user to reset the filter indication. When the value of 1 is written to this characteristic by the user, the accessory should reset it to 0 once the relevant action to reset the filter indication is executed. If the accessory supports Filter Change Indication, the value of that characteristic should also reset back to 0.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000AD-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.filter.reset-indication
Permissions	Paired Write
Format	uint8
Minimum Value	1
Maximum Value	1

10.92 Target Air Purifier State

This characteristic describes the target state of the air purifier.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000A8-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.air-purifier.state.target
Permissions	Paired Write, Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 ”Manual” 1 ”Auto”

10.93 Target Fan State

This characteristic describes the target state of the fan.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000BF-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.fan.state.target
Permissions	Paired Write, Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 "Manual" 1 "Auto"

10.94 Current Fan State

This characteristic describes the current state of the fan.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000AF-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.fan.state.current
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	2
Step Value	1
Valid Values	0 "Inactive" 1 "Idle" 2 "Blowing Air"

10.95 Active

This characteristic describes if the service is currently active.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000B0-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.active
Permissions	Paired Write, Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 ”Inactive” 1 ”Active”

10.96 Swing Mode

This characteristic describes if swing mode is enabled.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000B6-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.swing-mode
Permissions	Paired Read, Notify, Paired Write
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 ”Swing disabled” 1 ”Swing enabled”

10.97 Current Tilt Angle

This characteristic describes the current angle of slats for accessories such as windows, fans, portable heater/coolers etc. This characteristic takes values between -90 and 90. A value of 0 indicates that the slats are rotated to be fully open. At value 0 the user-facing edge and the window-facing edge are perpendicular to the window.

For `Horizontal` slat (see ”10.88 Slat Type” (page 229)): A value of -90 indicates that the slats are rotated all the way in a direction where the user-facing edge is to the left of the window-facing edge.

For `Vertical` slat (see "10.88 Slat Type" (page 229)): A value of -90 indicates that the slats are rotated all the way in a direction where the user-facing edge is higher than the window-facing edge.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000C1-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.tilt.current
Permissions	Paired Read, Notify
Format	int
Minimum Value	-90
Maximum Value	90
Step Value	1
Unit	arcdegrees

10.98 Target Tilt Angle

This characteristic describes the target angle of slats for accessories such as windows, fans, portable heater/coolers etc.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000C2-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.tilt.target
Permissions	Paired Read, Paired Write, Notify
Format	int
Minimum Value	-90
Maximum Value	90
Step Value	1
Unit	arcdegrees

10.99 Ozone Density

This characteristic indicates the current ozone density in micrograms/m^3.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000C3-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.density.ozone
Permissions	Paired Read, Notify
Format	float
Minimum Value	0
Maximum Value	1000

10.100 Nitrogen Dioxide Density

This characteristic indicates the current NO2 density in micrograms/m^3.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000C4-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.density.no2
Permissions	Paired Read, Notify
Format	float
Minimum Value	0
Maximum Value	1000

10.101 Sulphur Dioxide Density

This characteristic indicates the current SO2 density in micrograms/m^3.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000C5-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.density.so2
Permissions	Paired Read, Notify
Format	float
Minimum Value	0
Maximum Value	1000

10.102 PM2.5 Density

This characteristic indicates the current PM2.5 micrometer particulate density in micrograms/m^3.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000C6-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.density.pm2_5
Permissions	Paired Read, Notify
Format	float
Minimum Value	0
Maximum Value	1000

10.103 PM10 Density

This characteristic indicates the current PM10 micrometer particulate density in micrograms/m^3.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000C7-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.density.pm10
Permissions	Paired Read, Notify
Format	float
Minimum Value	0
Maximum Value	1000

10.104 VOC Density

This characteristic indicates the current volatile organic compound density in micrograms/m^3.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000C8-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.density.voc
Permissions	Paired Read, Notify
Format	float
Minimum Value	0
Maximum Value	1000

10.105 Service Label Index

This characteristic should be used identify the index of the label that maps to "10.106 Service Label Namespace" (page 237)
used by the accessory.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000CB-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.service-label-index
Permissions	Paired Read
Format	uint8
Minimum Value	1
Step Value	1

10.106 Service Label Namespace

This characteristic describes the naming schema for an accessory. For example, this characteristic can be used to describe the type of labels used to identify individual services of an accessory.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000CD-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.service-label-namespace
Permissions	Paired Read
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 "Dots. For e.g "." ".." "..." """ 1 "Arabic numerals. For e.g. 0,1,2,3" 2-255 "Reserved"

10.107 Color Temperature

This characteristic describes color temperature which is represented in reciprocal megaKelvin (MK^{-1}) or mirek scale. (`M = 1,000,000 /` K where M is the desired mirek value and K is temperature in Kelvin)

If this characteristic is included in the "9.5 Lightbulb" (page 154), "10.13 Hue" (page 183) and "10.30 Saturation" (page 190) must not be included as optional characteristics in "9.5 Lightbulb" (page 154). This characteristic must not be used for lamps which support color.

This characteristic requires iOS 10.3 or later.

Property	Value
UUID	000000CE-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.color-temperature
Permissions	Paired Read, Paired Write, Notify
Format	uint32
Minimum Value	50
Maximum Value	400
Step Value	1

10.108 Current Heater Cooler State

This characteristic describes the current state of a heater cooler.

This characteristic requires iOS 11 or later.

Property	Value
UUID	000000B1-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.heater-cooler.state.current
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	3
Step Value	1
Valid Values	0 "Inactive" 1 "Idle" 2 "Heating" 3 "Cooling"

10.109 Target Heater Cooler State

This characteristic describes the target state of heater cooler.

`Heat or Cool` state must only be included for accessories which include both a cooler and a heater. `Heat or Cool` state (see "10.109 Target Heater Cooler State" (page 239)) implies that the accessory will always try to maintain the "10.9 Current Temperature" (page 181) between "10.12 Heating Threshold Temperature" (page 182) and "10.5 Cooling Threshold Temperature" (page 178) and if the "10.9 Current Temperature" (page 181) increases above/falls below the threshold temperatures the equipment will start heating or cooling respectively.

In `Heat` state the accessory will start heating if the current temperature is below the "10.12 Heating Threshold Temperature" (page 182). In `Cool` state the accessory will start cooling if the current temperature is greater than the "10.5 Cooling Threshold Temperature" (page 178).

This characteristic requires iOS 11 or later.

Property	Value
UUID	000000B2-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.heater-cooler.state.target
Permissions	Paired Write, Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	2
Step Value	1
Valid Values	0 ”Heat or Cool” 1 ”Heat” 2 ”Cool”

10.110 Current Humidifier Dehumidifier State

This characteristic describes the current state of a humidifier or/and a dehumidifier.

This characteristic requires iOS 11 or later.

Property	Value
UUID	000000B3-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.humidifier-dehumidifier.state.current
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	3
Step Value	1
Valid Values	0 ”Inactive” 1 ”Idle” 2 ”Humidifying” 3 ”Dehumidifying”

10.111 Target Humidifier Dehumidifier State

This characteristic describes the target state of a humidifier or/and a dehumidifier.

`Humidifier or Dehumidifier` state must only be included for accessories which include both a humidifier and a dehumidifier. `Humidifier or Dehumidifier` state (see ”10.111 Target Humidifier Dehumidifier State” (page 240)) implies that the accessory will always try to maintain the "10.8 Current Relative Humidity" (page 180) between "10.114 Relative Humidity Humidifier Threshold" (page 242) and "10.113 Relative Humidity Dehumidifier Threshold" (page 242) and if the "10.8 Current Relative Humidity" (page 180) increases above/falls below the threshold relative humidity the equipment will start dehumidifying or humidifying respectively.

In `Humidifier` state the accessory will start humidifying if the current humidity is below the "10.114 Relative Humidity Humidifier Threshold" (page 242). In `Dehumidifier` mode the accessory will start dehumidifying if the current humidity is greater than the "10.113 Relative Humidity Dehumidifier Threshold" (page 242).

This characteristic requires iOS 11 or later.

Property	Value
UUID	000000B4-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.humidifier-dehumidifier.state.target
Permissions	Paired Read, Notify, Paired Write
Format	uint8
Minimum Value	0
Maximum Value	2
Step Value	1
Valid Values	0 "Humidifier or Dehumidifier" 1 "Humidifier" 2 "Dehumidifier"

10.112 Water Level

This characteristic describes the current water level.

This characteristic requires iOS 11 or later.

Property	Value
UUID	000000B5-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.water-level
Permissions	Paired Read, Notify
Format	float
Minimum Value	0
Maximum Value	100
Step Value	1
Unit	percentage

10.113 Relative Humidity Dehumidifier Threshold

This characteristic describes the relative humidity dehumidifier threshold. The value of this characteristic represents the 'maximum relative humidity' that must be reached before dehumidifier is turned on.

This characteristic requires iOS 11 or later.

Property	Value
UUID	000000C9-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.relative-humidity.dehumidifier-threshold
Permissions	Paired Read, Paired Write, Notify
Format	float
Minimum Value	0
Maximum Value	100
Step Value	1
Unit	percentage

10.114 Relative Humidity Humidifier Threshold

This characteristic describes the relative humidity humidifier threshold. The value of this characteristic represents the 'minimum relative humidity' that must be reached before humidifier is turned on.

This characteristic requires iOS 11 or later.

Property	Value
UUID	000000CA-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.relative-humidity.humidifier-threshold
Permissions	Paired Read, Paired Write, Notify
Format	float
Minimum Value	0
Maximum Value	100
Step Value	1
Unit	percentage

10.115 Program Mode

This characteristic describes if there are programs scheduled on the accessory. If there are Programs scheduled on the accessory and the accessory is used for manual operation, the value of this characteristic must be `Program Scheduled, currently overridden to manual mode`.

This characteristic requires iOS 11.2 or later.

Property	Value
UUID	000000D1-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.program-mode
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	2
Step Value	1
Valid Values	0 ”No Programs Scheduled” 1 ”Program Scheduled” 2 ”Program Scheduled, currently overriden to manual mode” 3-255 ”Reserved”

10.116 In Use

This characteristic describes if the service is in use. The service must be `Active` before the value of this characteristic can be set to in use.

This characteristic requires iOS 11.2 or later.

Property	Value
UUID	000000D2-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.in-use
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 ”Not in use” 1 ”In use” 2-255 ”Reserved”

10.117 Set Duration

This characteristic describes the set duration. For a "9.40 Valve" (page 173) this duration defines how long a valve should be set to 'In Use'. Once the valve is 'In Use', any changes to this characteristic take affect in the next operation when the "9.40 Valve" (page 173) is `Active`.

This duration is defined in seconds.

This characteristic requires iOS 11.2 or later.

Property	Value
UUID	000000D3-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.set-duration
Permissions	Paired Write, Paired Read, Notify
Format	uint32
Minimum Value	0
Maximum Value	3600
Step Value	1

10.118 Remaining Duration

This characteristic describes the remaining duration on the accessory. Notifications on this characteristic must only be used if the remaining duration increases/decreases from the accessory's usual countdown of remaining duration and when the duration reaches 0. e.g. It must not send notifications when the remaining duration is ticking down from 100,99,98... if 100 was the initial Set duration. However, if the remaining duration changes to 95 from 92 (increase) or 85 from 92 (decrease which is not part of the usual duration countdown), it must send a notification.

This duration is defined in seconds.

This characteristic requires iOS 11.2 or later.

Property	Value
UUID	000000D4-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.remaining-duration
Permissions	Paired Read, Notify
Format	uint32
Minimum Value	0
Maximum Value	3600
Step Value	1

10.119 Valve Type

This characteristic describes the type of valve.

This characteristic requires iOS 11.2 or later.

Property	Value
UUID	000000D5-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.valve-type
Permissions	Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	3
Step Value	1
Valid Values	0 ”Generic valve” 1 ”Irrigation” 2 ”Shower head” 3 ”Water faucet” 4-255 ”Reserved”

10.120 Is Configured

This characteristic describes if the service is configured for use. (e.g. in an irrigation system, not all valves may be configured to use depending on physical wire connection)

This characteristic requires iOS 11.2 or later.

Property	Value
UUID	000000D6-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.is-configured
Permissions	Paired Write, Paired Read, Notify
Format	uint8
Minimum Value	0
Maximum Value	1
Step Value	1
Valid Values	0 "Not Configured" 1 "Configured" 2-255 "Reserved"

10.121 Target Control Supported Configuration

This characteristic allows the accessory to indicate the configuration it supports and is encoded as a list of TLV8 tuples.

This characteristic requires iOS 12 or later.

Property	Value
UUID	00000123-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.supported-target-configuration
Permissions	Paired Read
Format	tlv8

The value of this characteristic is a tlv8 encoded list of supported parameters. The list of types for the TLVs are as follows:

Table 10-21: Target Control Commands TLV8 Definition

Type	Name	Format	Description
1	Maximum Targets	1	Maximum number of targets that can be supported by this accessory. If a tuple of this TLV type is omitted, it is assumed that the accessory supports 16 targets.
2	Ticks Per Second	8	Resolution of the timestamp sent in the button value characteristic. The value is encoded as a uint64 value.
3	Supported Button Configuration	N	Configuration of the supported buttons - The value is a tlv8 encoded list.
4	Type	1	Must be set to 1 if the accessory is implemented as a hardware entity. Siri is only allowed for a hardware entity. See "13.3 Additional Security Requirements" (page 270) for more details.

The Supported Button Configuration describes multiple buttons encoded in the value. The configuration of a button starts with a Button ID TLV and ends at the end of the TLV value or the next Button ID TLV.

The Supported Button Configuration is expressed as a tlv8 encoding with the following types:

Table 10-22: Supported Button Configuration Format TLV8 Definition

Type	Name	Format	Description
1	Button ID	1	ID of a button; a value of 0 is invalid. When a TLV of this type is encountered, configuration of a new button is being described.
2	Button Type	2	Type of button: 0 - Undefined 1 - Menu 2 - Play/Pause 3 - TV/Home 4 - Select 5 - Arrow Up 6 - Arrow Right 7 - Arrow Down 8 - Arrow Left 9 - Volume Up 10 - Volume Down 11 - Siri 12 - Power 13 - Generic

10.122 Target Control List

This HAP characteristic allows a controller to manage the association of targets with the accessory. This characteristic must support write response and is implemented as a control point characteristic with the write value indicating the type of operation and the result expected in the write response. The write response must include all the target configurations present on the accessory after the write operation. The accessory must store the configuration from the Target Control List in its persistent store and this configuration must not be erased on reboot of the accessory. The configuration must be erased on factory reset of the accessory and when the last admin pairing is removed from the accessory.

Only admin controllers are allowed to perform any operations on this characteristic. Any read/writes to this characteristic by non admin controllers must result in error -70401 (Insufficient Privileges).

This characteristic requires iOS 12 or later.

Property	Value
UUID	00000124-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.target-list
Permissions	Paired Write, Paired Read, Write Response
Format	tlv8

The write value of this characteristic is a tlv8 encoded with the following parameters:

Table 10-23: Target Control List TLV8 Definition

Type	Name	Format	Description
1	Operation	1	Operation for the control point: 0 - Undefined 1 - List; response contains the current configuration 2 - Add; response contains the updated configuration 3 - Remove; response contains the updated configuration 4 - Reset; response contains no value 5 - Update; response contains the updated configuration
2	Target Configuration	tlv8	Configuration for a specific target. Value is tlv8 encoded. Not needed in the write value for `List` and `Reset` operations.

The Target Configuration describes multiple targets encoded in the value. The configuration of a target starts with a Target Identifier TLV and ends at the end of the TLV value or the next Target Identifier TLV. The Target Configuration is expressed as a tlv8 encoding with the following types:

Table 10-24: Target Configuration TLV8 Definition

Type	Name	Format	Description
1	Target Identifier	4	Identifier of the target; value is a uint32. 0 indicates an invalid Target or that no Target is currently selected (e.g., the target controller is controlling a non-HomeKit entity). When a TLV of this type is encountered, configuration of a new target is being described.
2	Target Name	N	Name of the target - value is a UTF8 string.
3	Target Category	2	Category of the target; value is a uint16
4	Button Configuration	tlv8	Configuration of an enabled button; value is encoded as a tlv8. One instance present per button enabled.

The Target Category has the following enumerations:

Table 10-25: Target Category Enumerations

Target Category	Target Description
0	Undefined
24	Apple TV

The Button Configuration describes multiple buttons ecoded in the value. The configuration of a button starts with a Button ID TLV and ends at the end of the TLV value or the next Button ID TLV.

The Button Configuration is expressed as a tlv8 encoding with the following types:

Table 10-26: Button Configuration TLV8 Definition

Type	Name	Format	Description
1	Button ID	1	ID of button; a value of 0 is invalid. When a TLV of this type is encountered, configuration of a new button is being described.
2	Button Type	2	Type of button. May be optionally present.
3	Button Name	N	Name of the Button - value is a UTF8 string. May be optionally present. If present, accessory must use this name instead of the name derived from the button type.

10.123 Button Event

Notifications on this characteristic can only be enabled by Admin controllers. Any requests to enable notification on this characteristic by non admin controllers must result in error -70401 (Insufficient Privileges).

This characteristic requires iOS 12 or later.

Property	Value
UUID	00000126-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.button-event
Permissions	Paired Read, Notify
Format	tlv8

The value of this characteristic is a tlv8 encoded list of supported parameters. The list of types for the TLVs area as follows:

Table 10-27: Button Event TLV8 Definition

Type	Name	Format	Description
1	Button ID	1	ID of button; a value of 0 is invalid.
2	Button State	1	State of the button; 0-Up 1-Down
3	Timestamp	8	Timestamp of the event. Units are ticks
4	Active Identifier	4	Identifier of the target; value is a uint32.

10.124 Active Identifier

This HAP characteristic allows the accessory to indicate the target that is currently selected in the UI of the accessory (e.g. remote which can control multiple Apple TVs).

This characteristic requires iOS 12 or later.

Property	Value
UUID	000000E7-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.active-identifier
Permissions	Paired Read, Notify
Format	uint32

The value of this characteristic is one of the Target Identifier values configured by a controller in the Target Control List HAP characteristic of the Target Control Management HAP service. If no target is currently selected, the value must be 0 (for e.g., a remote is currently controlling a non-HomeKit entity).

When the value of this characteristic is changed - either due to user selecting a different target or due to a management operation (such as removing the currently active target), the Active characteristic (000000B0) must be reset back to Inactive.

10.125 Selected Audio Stream Configuration

This is a control point characteristic that allows a controller to specify the selected audio attributes to be used for streaming audio from the accessory.

This characteristic requires iOS 12 or later.

Property	Value
UUID	00000128-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.selected-audio-stream-configuration
Permissions	Paired Read, Paired Write
Format	tlv8

The value of this characteristic is a tlv8 encoded list of supported parameters.

Table 10-28: Selected Audio Input Stream Configuration TLV8 Definition

Type	Name	Format	Description
1	Selected Audio Input Stream Configuration	N	The codec that is to be used for input-audio (ie, audio sent from accessory to controller). This has the format described below as the Selected Audio Stream Configuration TLV.

The "10.125 Selected Audio Stream Configuration" (page 250) TLV has the following form (which shares the same format as `Selected Audio Parameters` in the `Selected RTP Stream Configuration` characteristic:

Table 10-29: Selected Audio Stream Configuration TLV TLV8 Definition

Type	Name	Format	Description
1	Selected Audio Codec Type	1	(same as Selected Audio Parameter TLV 1, where Opus = 3)
2	Selected Audio Codec Parameters	N	Same as Selected Audio Parameter TLV 2, which is defined to be the same as the Codec Param in the Supported Audio Stream Configuration

10.126 Supported Data Stream Transport Configuration

This characteristics describes the data stream transport supported by the accessory.

This characteristic requires iOS 12 or later.

Property	Value
UUID	00000130-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.supported-data-stream-transport-configuration
Permissions	Paired Read
Format	tlv8

The value of this characteristic is a tlv8 encoded list of supported parameters.

Table 10-30: Transfer Transport Configuration TLV8 Definition

Type	Name	Format	Description
1	Transfer Transport Configuration	N	The configuration supported for the transport. There is one TLV of this type per supported transport type.

An accessory supporting multiple transport types must include one instance of the above TLV per supported transport type. The "10.126 Supported Data Stream Transport Configuration" (page 251) is encoded as a tlv8.

The list of types for this TLV are as follows:

Table 10-31: Transport Type TLV8 Definition

Type	Name	Format	Description
1	Transport Type	1	Type of transport: 0 - HomeKit Data Stream (See "6.10 HomeKit Data Stream" (page 93)) over TCP 1 - 255 Reserved for use by Apple

10.127 Setup Data Stream Transport

This is a control point characteristic which allows the controller to setup the data stream.

This characteristic requires iOS 12 or later.

Property	Value
UUID	00000131-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.setup-data-stream-transport
Permissions	Paired Read, Paired Write, Write Response
Format	tlv8

The value of this characteristic is a tlv8 encoded list of supported parameters.

Table 10-32: Setup Data Steam Session TLV8 Definition

Type	Name	Format	Description
1	Session Command Type	1	Session Command Type: 0 - Start Session 1 - 255 Reserved for use by Apple
2	Transport Type	1	See Transport Type in "Supported" characteristic.
3	Controller Key Salt	32	See "6.10.1 Data Stream Transport Security" (page 94)

After a write is issued to this characteristic by the controller, the accessory must respond with a tlv8 encoded value with the following parameters:

Table 10-33: Setup Data Stream Write Response TLV8 Definition

Type	Name	Format	Description
1	Status	1	Status of the previous write issued: 0 - Success 1 - Generic error 2 - Busy, maximum number of transfer transport sessions reached. 3 - 255 reserved for use by Apple
2	Transport Type Session Parameters	N	The transport type specific session parameters.
3	Accessory Key Salt	32	See "6.10.1 Data Stream Transport Security" (page 94)

The Transport Type Session Parameters TLV is encoded as a tlv8 - the list of types for the value for HomeKit Data Stream over TCP transport are as follows:

Table 10-34: Setup Data Stream Transport Configuration 3 TLV8 Definition

Type	Name	Format	Description
1	TCP Listening Port	2	The port the accessory is listening to accept the transport connection.

After the accessory recieves the `Start Session` command, it must receive the first packet from the controller on that port within 10 seconds from the reception of this command.

After receiving the write-response, the controller must open a TCP connection to the provided port and send an initial `hello` request. The controller will wait up to 10 seconds to receive the response and must not send any other message until it receives it. If no message is received, or the controller receives any other message but a successful response, then it must close the socket.

After receiving the connection, the accessory must validate that the first message received is a valid `hello` request and it must immediately reply. If no message is received before its 10 second timeout, or the accessory receives any other message, then it must close the socket.

If the accessory has multiple pending connections, and there is ambiguity in which controller is connecting, e.g. it provides the same port in all setup requests, it must attempt decrypting the first packet with each key that is a possible match. If none match, then it must close the socket since it received a connection that did not have a valid `hello` request as its first message. If a key is tested but is not a match, the nonce for that key must not be changed, so that when the actual controller with that key connects, that key is still usable.

10.128 Siri Input Type

This characteristic describes the type of Siri input used by the accessory. This characteristic requires iOS 12 or later.

Property	Value
UUID	00000132-0000-1000-8000-0026BB765291
Type	public.hap.characteristic.siri-input-type
Permissions	Paired Read
Format	uint8
Valid Values	0 ”Push button triggered Apple TV” 1-255 ”Reserved”

11 Apple-defined Profiles

11.1 Overview

An Apple-defined profile is comprised of Apple-defined characteristics and Apple-defined services. A custom profile may be comprised of custom characteristics, Apple-defined characteristics, and custom services. Accessories must use Apple-defined characteristics to expose functionality of the accessory if they are available, e.g. a temperature sensor must use the Apple-defined "10.9 Current Temperature" (page 181) characteristic rather than defining its own custom characteristic to expose the same functionality.

11.2 Lock

The purpose of the HomeKit Accessory Lock Profile is to provide a set of standard services and characteristics to describe and interact with locks, such as a door lock, or lock on an automobile. The physical equivalent of the functionality this service provides is turning a key inside a lock after it has been inserted. Services and characteristics contained herein only cover lock interaction. The services and characteristics defined in this document are designed to be used on both IP and Bluetooth Low Energy devices, unless otherwise noted. All transactions are secured with HAP session security, which is described in the "7 HomeKit Accessory Protocol for Bluetooth LE Accessories" (page 101).

11.2.1 Lock Mechanism Service

The "9.7 Lock Mechanism" (page 155) service is a primary service, designed to expose and control the physical lock mechanism on a device. Implementation of this service is mandatory.

An implementation should not add vendor-specific characteristics to the Lock Mechanism Service because doing so will increase characteristic discovery time due to additional protocol transactions.

11.2.2 Lock Management Service

The "9.6 Lock Management" (page 155) service is a secondary service, designed to expose deeper interaction with a Lock device. Implementation of this service is mandatory. However, portions of this service are optional.

An implementation may add vendor-specific characteristics to the Lock Management Service but doing so may increase characteristic discovery time due to additional protocol transactions.

11.2.2.1 Control Point Characteristic Commands

The "10.15 Lock Control Point" (page 184) characteristic accepts the following TLV8 commands:

Table 11-1: Lock Control Point Characteristic Commands TLV8 Definition

Type	Name	Format	Description
0x00	readLogsFromTime	int	Read the logs starting at the value, which is in seconds since epoch time. A zero-length value will read all logs since the last wipe.
0x02	clearLogs	int	Clear the Lock's logs.
0x03	setCurrentTime	int	Set the Lock's current time, defined as seconds since epoch time.

11.2.2.2 Version Characteristic

The version of the Lock Profile that the device supports, as a string. For example, if the version of the supported Apple Lock Profile is "2.3", that value should be returned when the characteristic is read. Inclusion of this characteristic is mandatory.

The value of this characteristic must be the string "1.0".

11.2.2.3 Logs Characteristic Response Format

The "10.20 Logs" (page 186) characteristic returns a list of logs provided in the following TLV8 format, with a separator in between each log entry.

Table 11-2: Logs Characteristic Response Format TLV8 Definition

Type	Name	Format	Description
0x01	accessor	string	The user name that accessed the lock (provided by the HomeKit Accessory Pairing Profile).
0x02	time	int	The time the accessor accessed the Lock, defined as seconds since epoch time.
0x03	action	int	The action the accessor took when accessing the Lock Mechanism. Bits: 1 "Lock mechanism state read" 2 "Lock Mechanism Action occurred" 3 "All logs cleared" 6-8 "Reserved"
0x04	vendorSpecific	data	Vendor-specific log elements. Limited to 255 bytes.
0x05	separator	none	Empty element to separate groups of items within a single payload.

12 IP Cameras

12.1 Overview

This document specifies the HAP based services, characteristics and communication protocol for IP camera accessories that support RTP streaming of audio/video.

12.2 Requirements

12.2.1 Streaming Requirements

RTP for media transport

- RTP - A Transport Protocol for Real-Time Applications - RFC 3550
- RTP Profile for Audio and Video Conferences with Minimal Control - RFC 3551
- Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF) - RFC 4585
- Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF) - RFC 5104
- Multiplexing RTP Data and Control Packets on a Single Port - RFC 5761
- RTP Payload Format for H.264 Video - RFC6184
- [3GPPTS 26.114] "Coordination of Video Orientation" Section 7.4.5
- RTP Payload Format for Opus Speech and Audio Codec RFC 7587 with an exception that Opus audio RTP Timestamp shall be based on RFC 3550
- The Secure Real-time Transport Protocol (SRTP) - RFC 3711
- The Use of AES-192 and AES-256 in Secure RTP - RFS 6188
- RTCP Sender Report for both Audio and Video must adhere to Section 6.4.1 of RFC 3550. Both Audio and Video NTP should be derived from the same "wall clock".
- Audio RTP timestamp reflects the sampling instant of the first octet in the RTP data packet and must adhere to section 5.1 of RFC 3550 for all codecs including OPUS
- Video RTP timestamp is set to the sampling timestamp of the content and must adhere to section 5.1 of RFC 6184

12.2.2 Optional Requirements

The following requirements may be supported by IP camera accessories

- Multiple audio/video sources need to be published as different accessories
- Support for accessory to publish supported audio output decode formats:
  - The assumption for now is that this is the same as the audio codec used for sourcing input audio

12.3 Service Definitions

12.3.1 IP Camera

The following services are required to support IP camera using HomeKit Accessory Protocol:

- "9.28 Camera RTP Stream Management" (page 165) services describe and allow configuration/control of at least two RTP session for streaming audio and video from the accessory
- "9.29 Microphone" (page 166) service provides control over audio sourcing

Optionally, an IP camera accessory can support additional services such as:

- "9.30 Speaker" (page 166) service for two-way audio

12.3.2 Video Doorbell Profile

This service describes the features of a video doorbell.

- "9.31 Doorbell" (page 167)
- Video Doorbell Secondary Services:

The following secondary services are designed to expose the IP camera functionality of a video doorbell:

Table 12-1: Video Doorbell Secondary Services

Service	Required or Optional
"9.28 Camera RTP Stream Management" (page 165)	Required
"9.30 Speaker" (page 166)	Required
"9.29 Microphone" (page 166)	Required

12.4 RTP configuration

12.4.1 RTCP Based Feedback

The IP Camera must support the reception of following RTCP-FB messages defined in RFC 4585 and RFC 5104 in addition to the regular RTCP reports mentioned in RFC 3550

- FIR - Upon receiving FIR, IP camera must send a decoder refresh point at the earliest opportunity. See section 3.5.1 of RFC 5104 for more details
- PLI - IP Camera must calculate Round trip time (RTT) using the incoming RTCP reports as detailed in RFC 3550. The IP Camera shall adhere to the following behavior when handling incoming PLI RFC 4585 requests,
  - If a PLI is received after 2 x RTT from when the last key frame was sent, the IP Camera must respond with a key frame at the earliest opportunity. In doing so, the subsequent periodic keyframes should be delayed by "Keyframe interval"

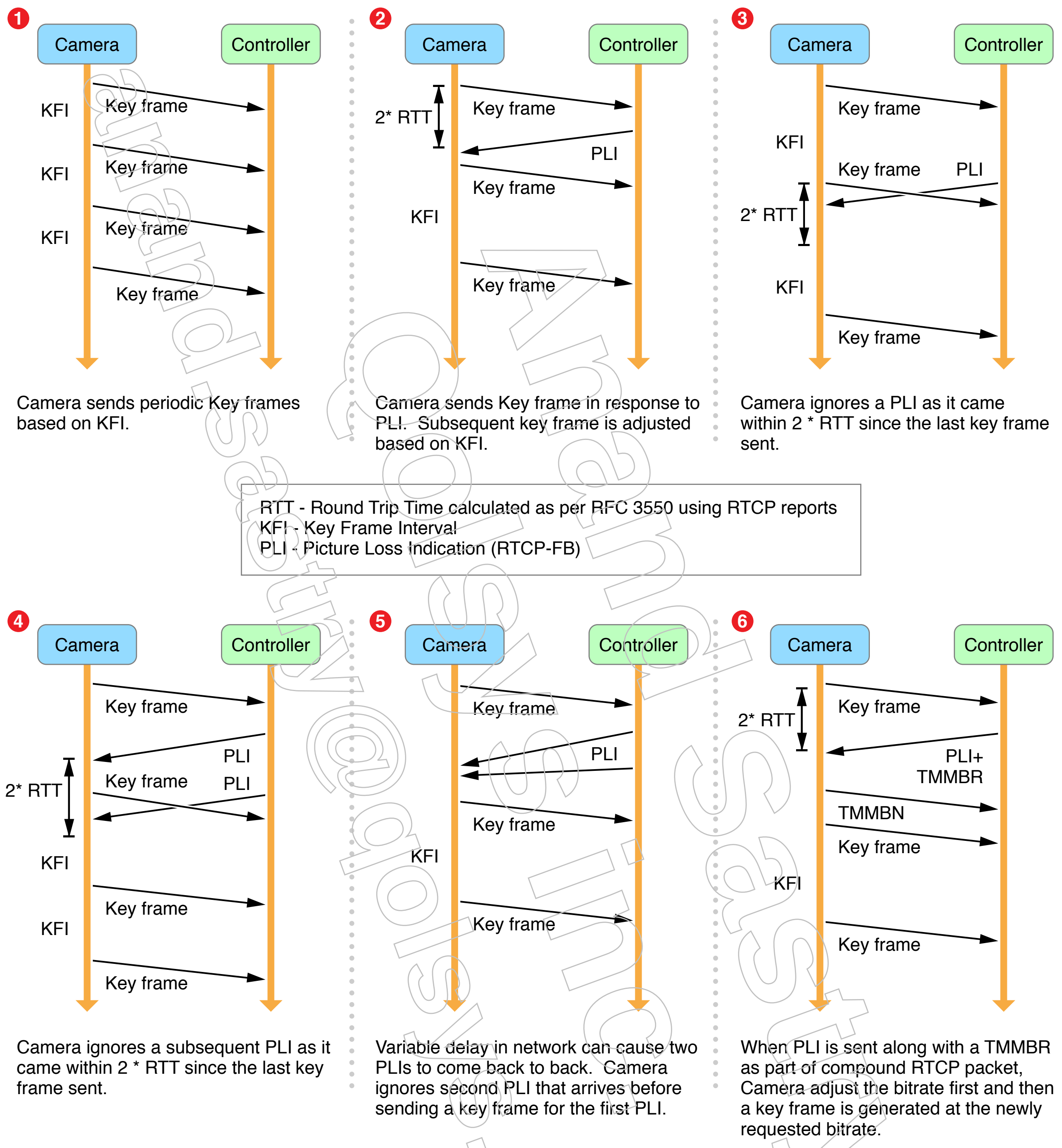
  - If a PLI is received after 2 x RTT along with TMMBR as part of compound RTCP packet, the IP Camera shall adjust the bitrate first and then generate the key frame at the newly requested bitrate
  - If a PLI is received within 2 x RTT from when the last keyframe was sent, the IP Camera must ignore the PLI request. The next periodic keyframe shall be sent as usual after the Keyframe interval elapses

  IP Camera must implement the PLI Handling as described in the below figure IP Camera - PLI Handling

- TMMBR/TMMBN - Upon receiving TMMBR, IP camera must cap its video bitrate to the bitrate requested in the TMMBR. Upon receiving the TMMBR, IP camera must acknowledge the receipt for TMMBR by sending a TMMBN immediately. See section 3.5.4 of RFC 5104 for more details.
  - TMMBR must be assigned a higher priority than PLI in case they are compounded in a single RTCP packet
  - When the IP camera sends TMMBN upon receiving a TMMBR, it must reset its RTCP send time and the next RTCP regular report should be sent after one RTCP interval period.

The IP Camera should support the reception of following RTCP-FB messages defined in RFC 5104.

- TSTR - Upon receiving TSTR, IP camera should adjust its temporal-spatial trade-off to what is requested in TSTR
- TSTN -Upon receiving the TSTR, IP camera should acknowledge the receipt for TSTR by sending a TSTN in the next RTCP interval



Camera sends periodic Key frames based on KFI.

Camera sends Key frame in response to PLI. Subsequent key frame is adjusted based on KFI.

Camera ignores a PLI as it came within 2 * RTT since the last key frame sent.

RTT - Round Trip Time calculated as per RFC 3550 using RTCP reports
KFI - Key Frame Interval
PLI - Picture Loss Indication (RTCP-FB)

Camera ignores a subsequent PLI as it came within 2 * RTT since the last key frame sent.

Variable delay in network can cause two PLIs to come back to back. Camera ignores second PLI that arrives before sending a key frame for the first PLI.

When PLI is sent along with a TMMBR as part of compound RTCP packet, Camera adjust the bitrate first and then a key frame is generated at the newly requested bitrate.

Figure 12-1: IP Camera - PLI Handling

12.4.2 RTP Extension for Coordination of Video Orientation

If the IP Camera can support rotation while streaming, it must support RTP extension for Coordination of Video Orientation [3GPP TS 26.114] for video RTP.

An IP Camera which supports the RTP extension for Co-ordination of Video Orientation, must:

- Add the payload bytes as defined in [3GPP TS 26.114 - Co-ordination of Video Orientation] onto the last RTP packet in each group of packets which make up a key frame

- Add the payload bytes onto the last RTP packet in each group of packets which make up another type of frame (e.g. a P-Frame) only if the current value is different from the previous value sent.

12.4.3 Reconfiguring a Streaming Session

The controller may issue a 'Reconfigure Streaming Session' command to change the parameters of the video stream. The controller will not issue this command to change the parameters of the audio stream.

When the controller issues this command, the IP camera must reconfigure the attributes of the stream without affecting the RTP session.

- The RTP stream must not be restarted
  - The RTP packet that is generated after the reconfigure command is executed at the IP camera (the next RTP packet) will be an IDR packet at a different resolution
- The sequence numbers must not be reset
  - The sequence number of the next RTP packet will increment by one
- The time-stamp of the next RTP packet may account for the time it took for this next RTP packet to be generated

Example of a Reconfigure Streaming Session Command:

```
Selected Stream Configuration:
        {
              SessionControl
{
            controlCommand = Reconfigure
            sessionID = 567FAAB5-BBB7-46FF-914A-E56B474FCDB4
}
videoParameters =
             {
                     attributes =
                     {
                        imageWidth = 320
                         imageHeight = 240
                     }
                     rtpParameters =
                     {
                         maximumBandwidth = 422
minimumRTCPInterval = 1
frameRate = 15
                     }
              }
}
```

12.5 Image Snapshot

An Image snapshot feature allows a controller to request the IP Camera accessory to capture and retrieve a static image from the camera.

When a controller sends an HTTP POST request with the desired image-width and image-height in the body the accessory must run the resource routine:

POST /resource HTTP/1.1

Example of a snapshot request in POST:

```
/resource
aid: <number> (Optional)
resource-type : image
image-width : <number>
image-height: <number>
```

The IP camera must set the MIMEType in the HTTP response to image and must provide the snapshot in JPEG format.

The IP Camera must return a snapshot only in the resolution that the controller requests for.

Note: aid field is optional and used to identify the camera that the snapshot request targets when applicable.

12.6 IP Camera Streaming Procedure

The sequence diagram below indicates the sequence of steps that a controller will perform to set up a streaming session with the IP camera accessory. A secure (pair-verified) session must be established before a streaming session can be established.

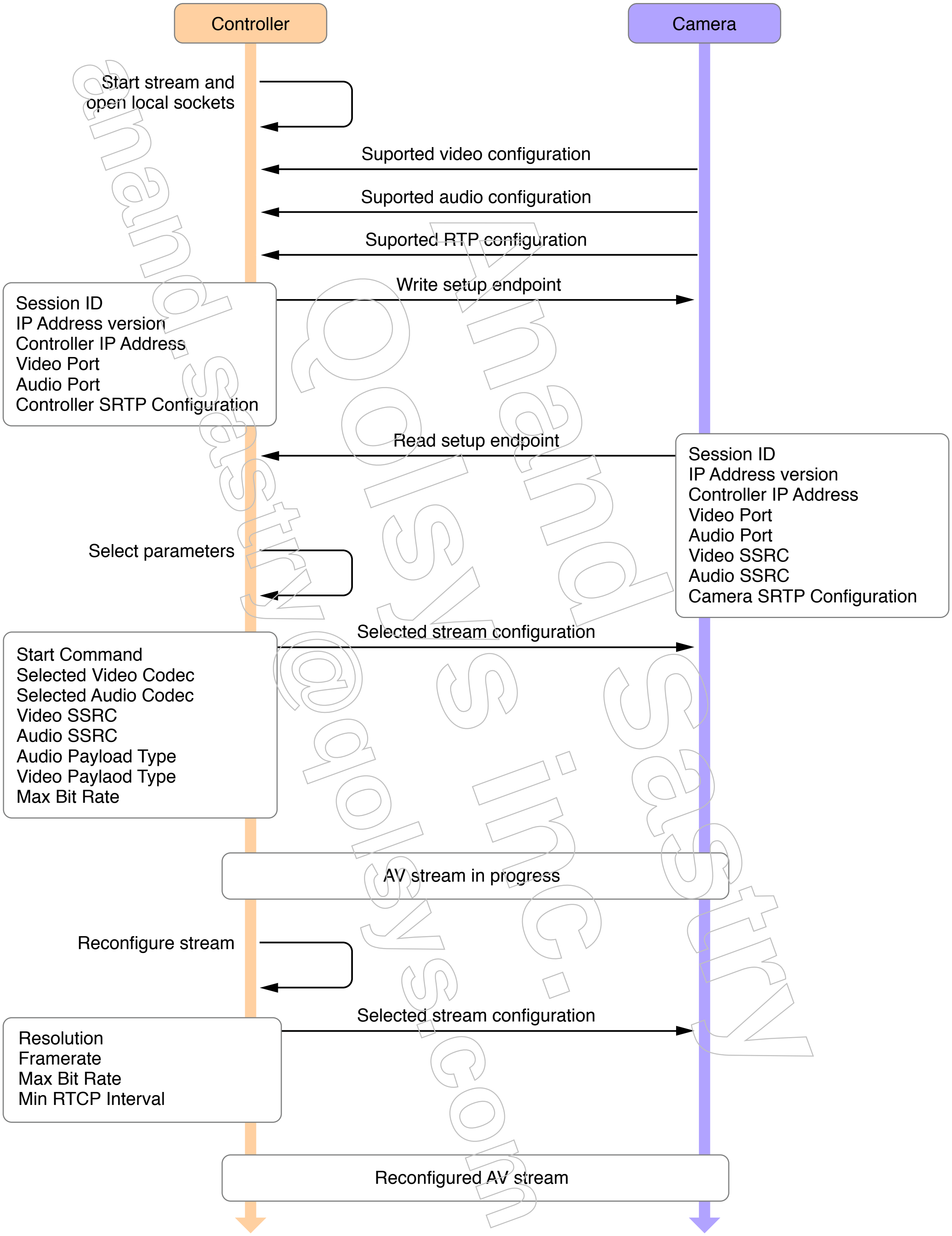


Figure 12-2: IP Camera Streaming Procedure

12.7 Multiple Camera RTP Stream Management

The IP camera must advertise a unique instance of the Camera RTP Stream Management Service for each of the Audio and Video stream configurations that it can simultaneously stream to iOS controllers.

- One stream must support a resolution of 1080p
- The minimum advertised resolution for any stream must be 720p
- The minimum supported frame rate for any stream must be 24fps, the minimum recommended frame rate for any stream is 30fps
- The IP camera must support at least two simultaneous streams viewable at two different controllers. Each stream configuration must include the video attributes TLVs for the highest supported resolution as well as all the lower resolutions that the camera can support at any given instance
- Instances of the Camera RTP Stream Management services must remain static through the lifetime of the IP camera. Services can only be added or removed via a firmware update to the accessory

12.8 Media Codecs

12.8.1 Mandatory Video RTP Service Settings

An IP camera accessory must support the following settings for the video RTP service:

- Codec type: H.264
    - Profile ID: Main Profile, Level: 4
- SPS PPS must use STAP-A fragmentation
- Video resolutions

Table 12-2: Video Resolutions

Aspect Ratio	Resolution
16:9	1920x1080 (1080p)
	1280x720 (720p)
	640x360
	480x270
	320x180
4:3	1280x960
	1024x768 (XGA)
	640x480 (VGA)
	480x360 (HVGA)
	320x240 (QVGA)

- Default MTU for Video RTP Stream
  - The IP camera must use the following values as default MTU's for Video RTP streams
    * IPv4 - 1378 bytes
    * IPv6 - 1228 bytes
  - The controller may request the camera to choose a different MTU in the Selected RTP Stream Configuration characteristic
- The minimum keyframe interval shall be 5 seconds

12.8.2 Mandatory Audio RTP service settings

- An IP camera must support AAC-ELD or Opus at 16K or 24K sample rates.
- AAC-ELD or Opus must be supported in Variable Bit Rate mode.
- The block size for AAC-ELD must be 480 samples

12.8.3 Mandatory RTP Profile

An IP camera must support SAVPF RFC 5124

12.9 Media Transport

Audio and video packets are delivered using RFC 3550. The RTP payload format is dependent on the codecs, and is described through the appropriate IETF documents (e.g. RFC 6184 for H.264, RFC 6716 for Opus and RFC 3640 - Section.3.3.6 - High Bit-rate AAC for AAC-ELD).

RTP and RTCP packets are multiplexed over a single port RFC 5761. Separate ports are used to stream audio and video.

SRTP RFC 3711 must be used to provide media security. The SRTP crypto-suite of the service is indicated by the SRTP Crypto-Suite characteristic. The SRTP master key and master salt are exchanged when setting up a stream through the Session Start characteristic.

12.10 Testing IP Cameras

This section outlines three kinds of certification tests that are needed for an IP camera accessory:

- HAP compliance: These tests validate that the accessory publishes it supported video and audio RTP services as per this specification and behaves per spec for setting up and managing a streaming session. These tests include validating the streaming session management and control services (such as volume of microphone/speaker or tilt/zoom of the camera).
- Encoder quality compliance: These tests validate that the audio video encoders supported by the accessory generate valid encoded data with the codec and audio/video parameters configured by the controller. This may involve well-defined source audio/video streams through the encoders with pre-defined configurations.
- Stream session behavior: These tests validate that the behavior of the streaming session when audio/video encoding/streaming parameters are modified on the fly. For example, when the controller commands the accessory to drop the bitrate of an audio/video bit stream, the accessory should do so in a manner not affecting the user experience by much.

13 Remotes for Apple TV

13.1 Button Control Requirements

Any accessory which controls an Apple TV must advertise the following additional services:

- "9.42 Target Control Management" (page 175)
- "9.43 Target Control" (page 175)

13.2 Siri Control Requirements

This section specifies the requirements for accessories which have a voice input button to control Apple TV.

The accessory must advertise the following additional services to support Siri:

- "9.46 Siri" (page 176)
- "9.45 Data Stream Transport Management" (page 176)
- "9.44 Audio Stream Management" (page 176)

When the user holds down this button, the accessory sends an audio stream to the active target. When the user releases the button, the target processes the full Siri request.

The controller creates a HomeKit Data Stream (HDS, see "6.10 HomeKit Data Stream" (page 93)) connection for each target configuration received by the accessory.

When voice input button is pressed, the accessory encodes the audio and sends it over the HomeKit Data Stream.

Siri audio must be sent using the following codec and parameters:

- Opus format (CELT encoding preferred)
- Variable Bitrate
- 16kHz sampling rate
- 20ms packet time

13.2.1 Packet Formats

13.2.1.1 Start

The accessory opens a new voice stream using the open request. The header should use `dataSend` as the protocol and `open` as the topic.

The request has the following message fields:

Table 13-1: Request Properties

Key	Type	Description
`target`	string	`controller` - to signify the direction of the send
`type`	string	`audio.siri` - the type of the stream

The response has the following message fields, if successful:

Table 13-2: Response Properties

Key	Type	Description
`streamId`	int64	used to identify this stream; chosen by the controller

This packet is sent by the accessory to signal that it wants to being a new transfer of audio. If the accessory does not receive a response within 5 seconds, the accessory should treat the stream as a failed setup. If the accessory receives a response after 5 seconds (but before the 10-second limit), then it must close the stream without sending any data (and use `timed out` as the reason).

If the controller accepts the stream, then the response includes a stream identifier that must be included in all other `dataSend` messages for this stream. The accessory can immediately start sending audio packets using the `dataSend.data packet`.

The controller must choose a stream identifier that is unambiguous and that does not conflict with any other active or recently-active stream (if there are any).

13.2.1.2 Binary Data

The accessory sends voice packets using the data event. The header should use `dataSend` as the protocol and `data` as the topic.

The event has the following message fields:

Table 13-3: Event Properties

Key	Type	Description
`streamId`	int64	Same identifier used in the `dataSend.open`
`packets`	Array	Array of dictionaries
`endOfStream`	boolean	(optional) Indicates the final frame; set to true.

Packet Dictionary:

Table 13-4: Packet Dictionary

Key	Type	Description
`data`	Bytes	Packet data
`metadata`	Dictionary	Meta data for the packet (Optional. This is required when type is audio.siri.)

For Siri audio data, metadata is defined as:

Table 13-5: Siri Audio Data

Key	Type	Description
`rms`	Float	The RMS value is set to each Opus frame (320 samples, or 20 ms). The following formula computes the RMS value: $RMS = sqrt(((s0)^2 + (s1)^2 ... + (s319)^2)/320)$
`sequenceNumber`	Int64	Integer that starts at 0 and counts up in each frame. Frames must still be delivered in order without gaps.

This packet is sent by the accessory to deliver the audio to the controller. The accessory is free to send any number of bytes in each frame, up to the frame maximum.

The packets field is required, but it is allowed to be zero-length even if more data will be delivered in later packets. For example, this packet may be used to send one of the other flags even if it does not yet have a new fragment to deliver. The `endOfStream` field indicates that the bulk transfer is complete and no further data will be delivered. The final frame must still contain the packets field, which will contain the last fragment of data in the transmission. If all data had already been sent in prior frames, it must still be present but will have zero-elements.

13.2.1.3 Acknowledgement

The controller will acknowledge the stream with this event. The header should use `dataSend` as the protocol and `ack` as the topic.

The event has the following message fields:

Table 13-6: Acknowledgement Event

Key	Type	Description
`streamId`	int64	Same identifier used in the `dataSend.open`
`endOfStream`	boolean	Indicates acknowledgement of the `endOfStream` flag. Set to true.

This packet must be sent by the controller after it has received the final frame with the `endOfStream` flag set. This is an indication to the accessory that the transfer has been accepted and can be considered successful.

13.2.1.4 Close

This event closes the stream. The header should use `dataSend` as the protocol and `close` as the topic.

The event has the following message fields:

Table 13-7: Close Event

Key	Type	Description
`streamId`	int64	Same identifier used in the `dataSend.open`
`reason`	int64	Example reasons: 0 - Normal - Normal Close 1 - Not Allowed - Controller will not allow the Accessory to send this transfer 2 - Busy - Controller cannot accept this transfer right now 3 - Cancelled - Accessory will not finish the transfer 4 - Unsupported - Controller does not support this stream type. 5 - Unexpected Failure - Some other protocol error occurred and the stream has failed. 6 - Timeout - Accessory could not start the session.

The accessory must send this packet to the controller in order to balance the `open` operation. After the accessory sends or receives a close event, it must not attempt to send any further packets using this stream identifier. After receiving this packet, the controller must not attempt to send any other packets (eg, cancel) with this identifier. If either side receives a packet after this one using the same identifier, it must ignore it.

If the stream was completed successfully, the accessory must send this packet after receiving the final stream acknowledgement from the controller.

The accessory may send this packet to the controller to indicate that this transmission is being cancelled and the data received is not considered complete. This could be due to an irrecoverable error on the accessory, but can also represent the user doing something that cancels the operation.

The controller may send this packet to the accessory to indicate that it is not interested in this transmission and that it will not be used.

13.2.1.5 Target Control Identifier

If the accessory is capable of doing Siri input, then the controller will send an advertisement of its Target Control identifier. The Accessory can use this identifier to know whicn Data Stream connection matches the Apple TV it intends to send Siri audio to.

The protocol is `targetControl` and event topic is `whoami`

The event has the following message fields:

Table 13-8: Target Control Event

Key	Type	Description
`identifier`	uint32	The Target Control identifier

13.3 Additional Security Requirements

- Button press events from remote accessories must not be stored on servers.
- Any servers used to relay button press events from the remote accessory to a HomeKit bridge must be App Transport Security compliant.
- Siri for Apple TV control is allowed for remote accessories implemented as a hardware entity (see type in "10.121 Target Control Supported Configuration" (page 246)) only.
- A remote accessory implemented as a hardware entity (see type in "10.121 Target Control Supported Configuration" (page 246)) must send Siri audio (if applicable) and button press events over the local network only and not through a server.

14 Accessory Categories

This chapter provides a list supported accessory categories in HomeKit.

Value	Accessory Category
1	Other
2	Bridges
3	Fans
4	Garage Door Openers *
5	Lighting
6	Locks *
7	Outlets
8	Switches
9	Thermostats
10	Sensors
11	Security Systems *
12	Doors *
13	Windows *
14	Window Coverings
15	Programmable Switches
16	Reserved
17	IP Cameras *
18	Video Doorbells *
19	Air Purifiers
20	Heaters
21	Air Conditioners
22	Humidifiers
23	Dehumidifiers
24-27	Reserved
28	Sprinklers
29	Faucets
30	Shower Systems
31	Reserved
32	Remotes
32+	Reserved

Note: An accessory with support for multiple categories should advertise the primary category. An accessory for which a primary category cannot be determined or the primary category isn't among the well defined categories (2-9) falls in the 'Other' category.
*: These accessory categories must use programmable tags if they support NFC.

15 App Requirements

15.1 Overview

This chapter describes the requirements for iOS apps that communicate with HomeKit-enabled accessories. All HomeKit accessories, except for those that use in-field provisioning through software token-based authentication, must be able to be set up and controlled (out of the box or after a factory reset) using the Apple Home app. An app that asks users to create an account must provide an option for users to defer this step until HomeKit setup is complete and the HomeKit accessory is usable. The account login state must not block users from controlling the HomeKit accessory or receiving firmware updates.

15.2 General Requirements

This section explains the scenarios in which an app is required, and what features must be supported.

15.2.1 App with full HomeKit API Support

An app with full HomeKit API support is optional. If this type of app is offered, it must adhere to all "musts" in "15.3 Functional Requirements" (page 274).

15.2.2 App with limited HomeKit API Support

An app with limited or no HomeKit support is allowed if (1) the accessory uses an app to receive firmware updates, or (2) the accessory is a bridge that uses an app to associate endpoints. The app must however direct users to the Apple Home app for accessory control and configuration.

15.2.3 App for In-Field Provisioning through Software Authentication

An app is required if the accessory supports in-field provisioning. The app must support at least the following:

- Update the accessory firmware to HomeKit-enabled firmware that supports software authentication.
- Retrieve the setup code and pass the setup payload "4.3 Format of Setup Payload" (page 34) to HomeKit to initiate pairing by using the addAndSetupAccessoriesWithPayload API.

15.2.4 App Not Required

An app is not required if the following conditions are met:

- The accessory receives firmware updates automatically from a manufacturer-hosted server without user interaction, i.e., pushed by a server.
- The accessory does not have any user-interactive custom characteristics i.e., a custom characteristic that would need to be configured by the user for desired operation.
- (For bridge accessories) The bridge can add endpoints through a physical, non-app based process, e.g., pairing or link button.

15.3 Functional Requirements

1. App must support adding a HomeKit home using the addHome API.
2. App must support the addAndSetupAccessories API, which includes – pairing with the accessory, association of the accessory to an existing room, and the option to create a new room if one does not exist.
3. App must allow the user to create rooms in the home using addRoom API.
4. App must support identifying a HomeKit accessory that is part of a home using the identify API.
5. App must allow the user to reassign the accessory to a different room if they choose using assignAccessory.
6. App must not create or delete rooms, zones, action sets, service groups, triggers, etc., without a user's consent.
   - App must not remove, reset or add/remove pairings on a HomeKit accessory without user consent.
   - App must not allow users to delete homes.
7. App must support removing HomeKit accessories using the removeAccessory API.
8. App must allow user configuration of "Associated Service Type" for outlets and switches using the updateAssociatedServiceType API.
9. App must not change values (e.g.,turning off a lightbulb) without user intent.
10. App must not auto-refresh values of characteristics at a frequency less than 2 minutes.
11. Accessories must be able to be set up and controlled (out of the box or after factory reset) using the Apple Home app. This is not applicable for in-field provisioned software token authentication accessories.
12. App should include HomeKit APIs for users to control HomeKit accessories such as readValue and writeValue.
13. App should not require the iOS device to be connected to the Internet for the HomeKit setup process to be completed. This is not applicable for accessories that support software based authentication.
14. App that asks users to create an account must provide an option for users to defer this step until HomeKit setup is complete and the HomeKit accessory is usable.
    - Account login state must not block users from control of the HomeKit accessory or receiving firmware updates.
15. App must allow changes to an accessory's service name(s) after setup is complete using updateName for the service name.
16. App must support a shared home created by the Apple Home app.
17. App must ensure that HomeKit accessories are not updatable to a firmware version that is incompatible with the iOS version of the controller.
18. An App that only supports iOS 10 or later must not use the External Accessory framework to browse for and/or configure unpaired accessories. Instead the addAndSetupAccessories API must be used.

16 Appendix

16.1 TLVs

Pairing messages frame data into 1 or more items, each containing an 8-bit type followed by an 8-bit length followed by a variable length value format (TLV8).

Table 16-1: TLV8 Structure

Name	Size in Bytes	Description
Type	1	Type of value.
Length	1	Number of value bytes, excluding type and length fields. 0 means there is no value data.
Value	Variable	Contains <Length> bytes of data for the value. May be empty if length is 0.

Table 16-2: TLV8 Value Formats

Name	Description
Bytes	Raw binary data.
Integer	Little endian integer. This must use the minimum number of bytes to encode the integer.
UTF-8	UTF-8 encoded string. This must not contain a NUL terminator (size is specified by the TLV length field).

16.1.1 TLV Rules

The following rules apply:

- TLV items with unrecognized types must be silently ignored.
- TLV item length describes the number of value bytes, excluding the type and length bytes. 0 is a valid length.
- Values less than or equal to 255 bytes must be contained in a single TLV item and not fragmented.
- Values larger than 255 bytes must be fragmented into multiple, TLV fragment items.
- Each TLV fragment item must begin with an 8-bit type followed by an 8-bit length.
- Each TLV fragment item's type must be same.
- Each TLV fragment item's length must describe only that fragment's portion of the value.
- Each TLV fragment item must have a non-0 length.
- TLV fragment items must be contiguous.
- Only the last TLV fragment item in series of contiguous TLV fragment items may have non-255 byte length.
- There may be multiple, separate TLV items of the same type if separated by a TLV item of a different type.

16.1.2 TLV Examples

TLV Example 1 (2 small TLVs)

```
Byte 0: 0x06 (state)
Byte 1: 0x01 (1 byte value)
Byte 2: 0x03 (M3)
Byte 3: 0x01 (identifier)
Byte 4: 0x05 (5 byte value)
Byte 5: 0x68 (ASCII ’h’)
Byte 6: 0x65 (ASCII ’e’)
Byte 7: 0x6C (ASCII ’l’)
Byte 8: 0x6C (ASCII ’l’)
Byte 9: 0x6F (ASCII ’o’)
Total:  10 bytes

Raw hex dump:
00000000  07 01 03 01 05 68 65 6c  6c 6f                    |.....hello|
0000000a
```

TLV Example 2 (1 small TLV, 1 300-byte value split into 2 TLVs, 1 small TLV)

```
Byte   0: 0x06 (state)
Byte   1: 0x01 (1 byte value)
Byte   2: 0x03 (M3)
Byte   3: 0x09 (certificate)
Byte   4: 0xFF (255 byte value)
Byte   5: 0x61 (ASCII ’a’)
... 254 more bytes containing 0x61 (ASCII ’a’)
Byte 260: 0x09 (certificate...continuation of previous TLV)
Byte 261: 0x2D (45 byte value)
Byte 262: 0x61 (ASCII ’a’)
... 44 more bytes containing 0x61 (ASCII ’a’)
Byte 307: 0x01 (identifier...new TLV item)
Byte 308: 0x05 (5 byte value)
Byte 309: 0x68 (ASCII ’h’)
Byte 310: 0x65 (ASCII ’e’)
Byte 311: 0x6C (ASCII ’l’)
Byte 312: 0x6C (ASCII ’l’)
Byte 313: 0x6F (ASCII ’o’)
Total: 314 bytes

Raw hex dump:
00000000  07 01 03 0a ff 61 61 61  |.....aaa|
00000008  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000010  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000018  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000020  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000028  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000030  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000038  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000040  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000048  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000050  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000058  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000060  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000068  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000070  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000078  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000080  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000088  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000090  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000098  61 61 61 61 61 61 61 61  |aaaaaaaa|
000000A0  61 61 61 61 61 61 61 61  |aaaaaaaa|
000000A8  61 61 61 61 61 61 61 61  |aaaaaaaa|
000000B0  61 61 61 61 61 61 61 61  |aaaaaaaa|
000000B8  61 61 61 61 61 61 61 61  |aaaaaaaa|
000000C0  61 61 61 61 61 61 61 61  |aaaaaaaa|
000000C8  61 61 61 61 61 61 61 61  |aaaaaaaa|
000000D0  61 61 61 61 61 61 61 61  |aaaaaaaa|
000000D8  61 61 61 61 61 61 61 61  |aaaaaaaa|
000000E0  61 61 61 61 61 61 61 61  |aaaaaaaa|
000000E8  61 61 61 61 61 61 61 61  |aaaaaaaa|
000000F0  61 61 61 61 61 61 61 61  |aaaaaaaa|
000000F8  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000100  61 61 61 61 0a 2d 61 61  |aaaa.-aa|
00000108  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000110  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000118  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000120  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000128  61 61 61 61 61 61 61 61  |aaaaaaaa|
00000130  61 61 61 01 05 68 65 6c  |aaa..hel|
00000138  6c 6f                    |lo|
```

17 Document Revision History

This chapter describes the changes to *HomeKit Accessory Protocol Specification* from the previous revision.

New Content:

Section "6.7.3 Writing a Characteristic with Write Response" (page 87)

Section "6.10 HomeKit Data Stream" (page 93)

Section "9.42 Target Control Management" (page 175)

Section "9.43 Target Control" (page 175)

Section "9.44 Audio Stream Management" (page 176)

Section "9.45 Data Stream Transport Management" (page 176)

Section "9.46 Siri" (page 176)

Section "10.121 Target Control Supported Configuration" (page 246)

Section "10.122 Target Control List" (page 247)

Section "10.123 Button Event" (page 249)

Section "10.124 Active Identifier" (page 250)

Section "10.125 Selected Audio Stream Configuration" (page 250)

Section "10.126 Supported Data Stream Transport Configuration" (page 251)

Section "10.127 Setup Data Stream Transport" (page 252)

Section "10.128 Siri Input Type" (page 253)

Chapter "13 Remotes for Apple TV" (page 266)

Chapter "15 App Requirements" (page 273)

Updated Content:

Section "3.6 Software Authentication" (page 30)

Section "4.2.2 Setup ID" (page 33)

Table Table 4-1 (page 37)

Section "4.4.2.1 Requirements" (page 36)

Section "5.11 Remove Pairing" (page 53)

Section "6-3 Characteristic Objects" (page 68)

Section "6-4 Characteristic Properties" (page 70)

Section "6.7.2 Writing Characteristics" (page 81)

Section "6.8 Notifications" (page 90)

Section "6.8.2 Accessory Sends Events to Controller" (page 91)

Section "6-14 HTTP URLs" (page 93)

Section "7.3.4.15 HAP-Characteristic-Configuration-Response" (page 111)

Section "10.75 Setup Endpoints" (page 216)

Section "12.5 Image Snapshot" (page 261)

Section "12.7 Multiple Camera RTP Stream Management" (page 264)

Section "14 Accessory Categories" (page 271)

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to be used in the development of solutions for Apple-branded products.

Apple Inc.
One Apple Park Way
Cupertino, CA 95014
408-996-1010

APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT, ERROR OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

Some jurisdictions do not allow the exclusion of implied warranties or liability, so the above exclusion may not apply to you.