



# **LTP / BlueHDP+**

User Guide

Release r01

---

**Table of contents**

1	Introduction .....	4
2	Glossary .....	5
3	Use Cases for the LTP protocol.....	7
4	Use Cases for the BlueHDP+ API .....	8
5	The Concept of LTP / BlueHDP+ .....	9
6	MDH Implementation Guidelines .....	12
6.1	Device Visibility .....	13
6.1.1	Pairable Mode .....	13
6.1.2	Non Pairable Mode .....	13
6.2	System Startup and Configuration .....	14
6.2.1	MDC Startup .....	14
6.2.2	Defining Security .....	15
6.2.3	Defining Connectivity .....	15
6.2.4	Defining Visibility .....	15
6.3	Connection Management .....	15
6.3.1	Incoming Connection and Disconnection .....	16
6.3.2	Outgoing Connection and Disconnection .....	18
6.4	Security Management .....	20
6.4.1	Common Terms for Security .....	20
6.4.1.1	SSP: Secure Simple Pairing .....	20
6.4.1.2	Legacy Pairing .....	20
6.4.1.3	MITM: Man-In-The-Middle .....	20
6.4.1.4	OOB: Out Of Band.....	21
6.4.1.5	Pairing .....	21
6.4.1.6	Bonding .....	21
6.4.1.7	Link Key .....	21
6.4.1.8	Authentication.....	21
6.4.1.9	Authorization .....	21
6.4.2	SSP Device Types .....	22
6.4.2.1	No I/O.....	22

---

6.4.2.2	Display Only .....	22
6.4.2.3	Display Yes/No .....	22
6.4.2.4	Keyboard Only .....	22
6.4.3	Security Methods .....	23
6.4.3.1	Legacy Pairing .....	23
6.4.3.2	Just Works .....	24
6.4.3.3	Numeric Comparison .....	25
6.4.3.4	Passkey Entry .....	26
6.4.3.5	Out of Band (OOB) .....	27
6.4.4	Data Exchange Management .....	28
6.4.4.1	Flow Control .....	28
6.4.4.2	APDU Segmentation and Reassembly .....	29
6.5	LTP Specifics .....	30
6.6	BlueHDP+ Specifics .....	30
7	Example for GUI Representation .....	31
7.1	Example GUI: Inquiry and Service Discovery .....	31
7.1.1	Example GUI: Graphical Representation .....	34
7.1.2	Flowchart for Inquiry with LTP/BlueHDP+ .....	35
7.1.3	Flowchart for HDP Service Discovery with LTP .....	36
7.1.4	Flowchart for HDP Service Discovery with BlueHDP+ .....	37
7.1.5	Logical data Structures for HDP Service Discovery .....	38
7.1.6	Flowchart for SPP Service Discovery with LTP .....	39
7.1.7	Flowchart for SPP Service Discovery with BlueHDP+ .....	40
7.1.8	Logical Data Structures for SPP Service Discovery .....	41
7.2	Connecting to Remote Devices .....	42
7.2.1	Connection to HDP Endpoint .....	42
7.2.2	Connecting to SPP Endpoint .....	42
7.3	References .....	42
8	History .....	43

---

## 1 Introduction

This document is meant to help developers to get some understanding of how the LTP protocol and the BlueHDP+ API work and how a system should be designed that uses these interfaces.

If information in this document conflicts with information in the BlueHDP+ or LTP specification, the information in the specification takes priority over this User Guide.

The intention is that a user of the LTP protocol or the BlueHDP+ API reads through this document briefly once before starting design and implementation of his application.

Dependent on the system's concept the interface type LTP protocol (serial link) or BlueHDP+ API (system internal software API) is appropriate. The use case LTP protocol is explained in chapter [3 Use Cases for the LTP protocol], the use case BlueHDP+ API in chapter [4 Use Cases for the BlueHDP+ API].

All following chapters are valid for both interfaces.

## 2 Glossary

APDU	An application PDU (APDU) is a block of data that is exchanged via LTP or BlueHDP+. For HDP the maximum size to be sent by a manager to an agent is 8KB and the maximum size of the APDU to send from an agent to a manager is 64KB. For SPP there is no general role for the maximum size since SPP is not defined for data block exchange but for byte stream data exchange. For each MDL connected the maximum APDU size for that given MDL is signaled separately as part of the <b>ConnectMDLInfo</b> message.
ACL	The asynchronous logical (ACL) link shall carry control information exchanged between the link managers of the Bluetooth master and the Bluetooth slave and L2CAP asynchronous and isochronous user data.
FSM	Finite State Machine
HDP	The Health Device Profile (HDP) is an application profile that defines the requirements for qualified Bluetooth healthcare and fitness (referred to as 'health') device implementations. This profile is used for connecting application data <i>source</i> devices such as blood pressure monitors, weight scales, glucose meters, thermometers, and pulse ox meters to application data <i>sink</i> devices such as mobile phones, laptops, desktop computers, and health appliances without the need for cables.
MCL	A Mediated Communications Link (MCL) identifies the full collection of L2CAP or RFCOMM connections between two instances of HDP or SPP, comprising a control channel and zero or more data channels. The control channel is always present while the MCL is active.
MDC	The Medical Device Controller (MDC), part of a medical device, is a Stollmann LTP capable module or BlueHDP+ stack and provides all "transport specific" functionality. This includes all Bluetooth related functionality like service exposure and discovery, connection handling, security, ensuring data integrity and flow-control.
MDEP	A Mediated Data End Point (MDEP) represents the logical function of a device and includes such information as MDEP ID, MDEP Role ( <i>source/sink</i> ), data type (device data specialization) and description. If a given implementation wants to accept MDL connections it must register at least one MDEP.
MDL	A Mediated Data Link (MDL) is a connection to an MDEP and is explicitly created by one of the two participating devices. An MDL can be seen as a data channel that addresses two specific endpoints of a connection. Multiple MDLs to one or multiple endpoints to or from a single remote

---

	device are allowed.
MDH	The Medical Device Host (MDH), part of a medical device, is vendor specific and provides all “medical” related functionality like measurement of medical parameters and calculations.
PSM	A Protocol Service Multiplexer (PSM) is an identifier that allows a remote device to address specific functionality within a given Bluetooth Stack.
SPP	The Serial Port Profile (SPP) is an application profile that defines the requirements for qualified Bluetooth device implementations for data exchange. This profile is used for connecting devices such as embedded desktop devices, mobile phones, laptops, desktop computers, and appliances without the need for cables.

### 3 Use Cases for the LTP protocol

LTP is a communication protocol for host / client device interaction that allows integrating Bluetooth functionality like SDP, HDP and SPP in a given system without having to integrate the complete Bluetooth stack software on the host platform.

Basically, this approach allows the integration of Bluetooth functionality by adding hardware running Bluetooth to a given system whereas the required software changes for the host system are rather minimal.

Communication on LTP is handled via serialized byte stream message exchange on a serial interface (e.g. UART). For further information regarding the LTP messages described in this document please refer to [X1] or [X2].

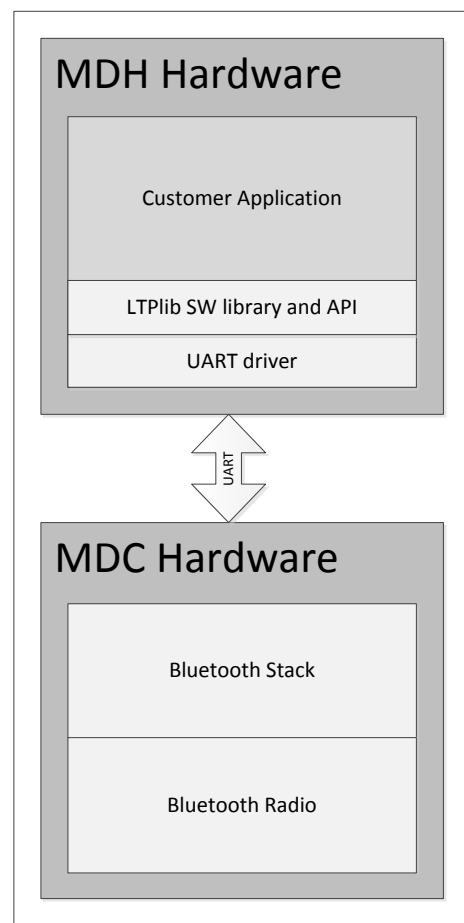
LTP defines two roles, the MDH, and the MDC role:

**MDH** stands for “Medical Device Host” and describes the host role of the LTP protocol where the customer application is located.

In an LTP setup the MDH may be a PC running various operating systems (e.g. Desktop PC, tablet PC), an embedded system with various IO capabilities (e.g. Smartphone, PDA style device) or a medical device that is able to take measurements (e.g. thermometer, weighing scale)

**MDC** stands for Medical Device Controller and describes the client role of the LTP protocol where the Bluetooth stack is located.

In an LTP setup the MDC might be a USB-Dongle type device (e.g. Stollmann BlueHDP+USB) or a serial module (e.g. Stollmann BlueMod+P24/G2, BlueMod+P25/G2).



## 4 Use Cases for the BlueHDP+ API

The BlueHDP+ API is a system internal API for embedded or PC based implementations that allows integrating Bluetooth functionality like SDP, HDP and SPP support in a given system by integrating the complete Bluetooth stack on the host platform so that the Bluetooth stack can run on the host MCU system.

Basically this approach allows the integration of Bluetooth functionality by adding software to a given system while the required hardware changes for the system are minimal.

Communication on BlueHDP+ is handled via system internal message exchange within the system's operating system. For further information regarding the BlueHDP+ messages described in this document please refer to [X1] or [X2].

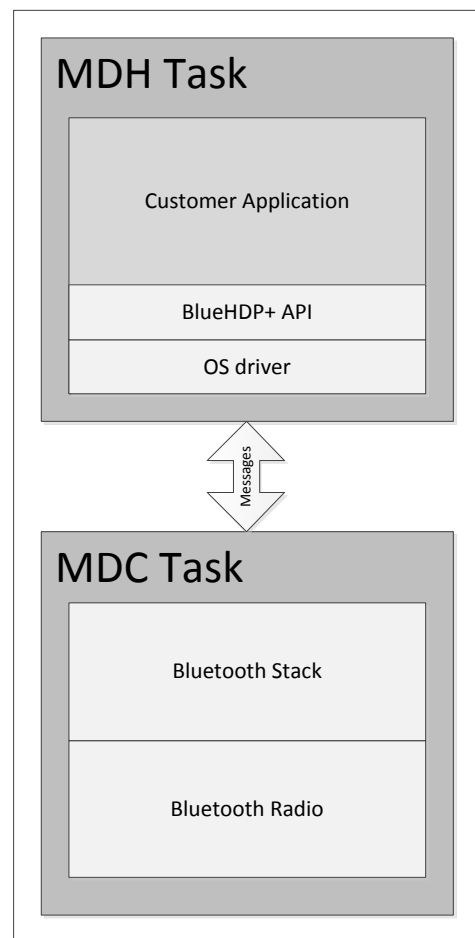
BlueHDP+ defines two roles, the "MDH", and the "MDC" role:

**MDH** stands for "Medical Device Host" and describes the host application role of the BlueHDP+ API where the customer application is located.

For a BlueHDP+ setup the MDH may be a customer application on a PC (e.g. Desktop PC, tablet PC) or a separate task within the operating system of an embedded system (e.g. Smartphone, PDA style device, thermometer, weighing scale)

**MDC** stands for "Medical Device Controller" and describes the system driver of the BlueHDP+ API where the Bluetooth Stack is located.

For a BlueHDP+ setup the MDC may be a DLL type driver within a PC based system or a separate task within the operating system of an embedded customer device.



## 5 The Concept of LTP / BlueHDP+

Both LTP protocol and BlueHDP+ API are based on the same message set, exchanged through different interface types. All following chapters are valid for both interfaces.

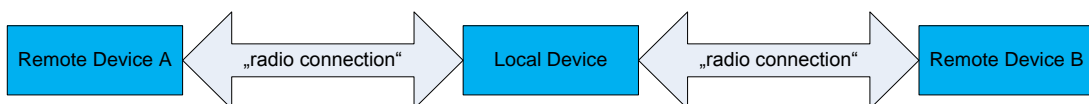
LTP and BlueHDP+ are designed to support multiple connections and communication channels in parallel. On each of this possible communication channels data packets can be exchanged and flow control has to be obeyed and can be applied.

Due to this LTP and BlueHDP+ do not support anything simple like “just connect to a remote device and send a bunch of bytes” but always require to address a specific communication channel for actions like channel setup, data exchange, flow control and disconnection. On the first look this may sound complicated, but you will find out that it is not once you get some understanding of the concept and follow some of the advices in this document.

A “communication channel” can be seen as a pipeline with one “end” on the local device and the other “end” on the remote device. The purpose of such a “communication channel” is to allow bi-directional “packet” exchange between both involved devices.

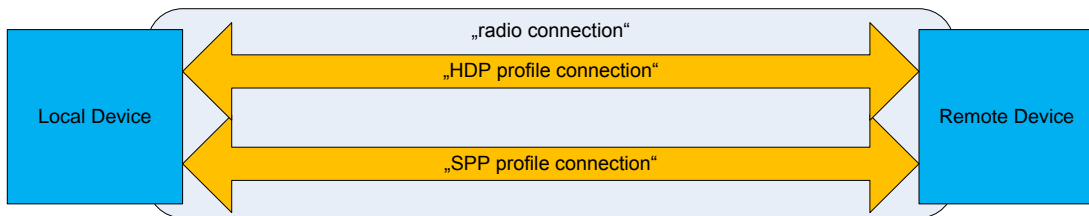
In order to establish a “communication channel” to a remote device a lot of things have to happen on the underlying stack layers. You don’t have to care about that a lot, but to understand the signaling some basics may be of your interest.

First of all a “radio connection” has to be established, regardless if just a remote device name is requested (***DeviceNameReq***), a request for services of the remote device is issued (***HDPDiscoveryReq***, ***SPPDiscoveryReq***) or a “communication channel” connection is requested. There is always exactly one “radio connection” to a given connected remote device. Multiple remote devices may be connected in parallel.

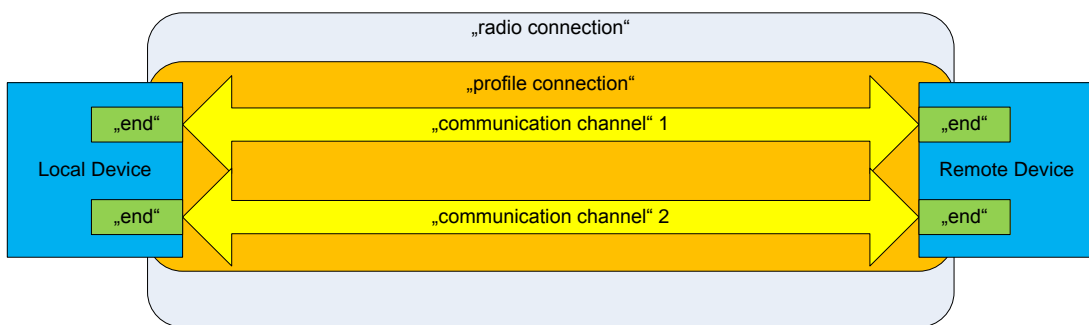


Once the “radio connection” to a remote device is established successfully, a “profile connection” has to be established if a “communication channel” connection is requested (actually for service discovery this is also true but completely hidden for your implementation). There is always exactly one “profile connection” for each used profile (e.g. HDP or SPP) to a connected remote device. Multiple “profile

connections” for different profiles (e.g. one for HDP, one for SPP) to one remote device may be connected in parallel.

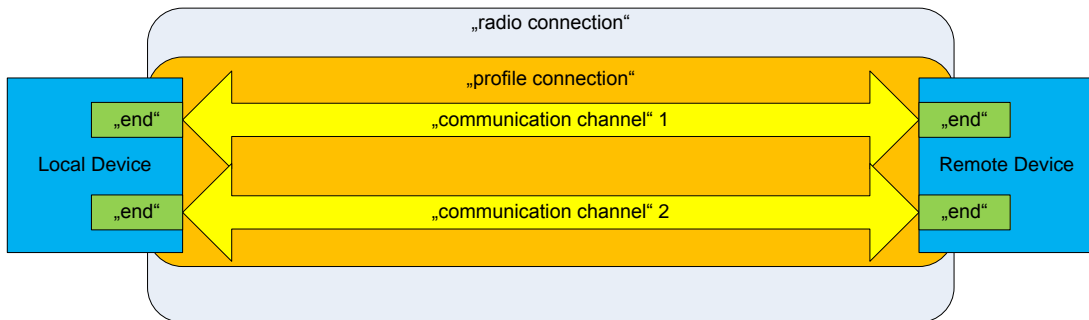


Once a “profile connection” is established, one or multiple “communication channels” can be established for each “profile connection”.

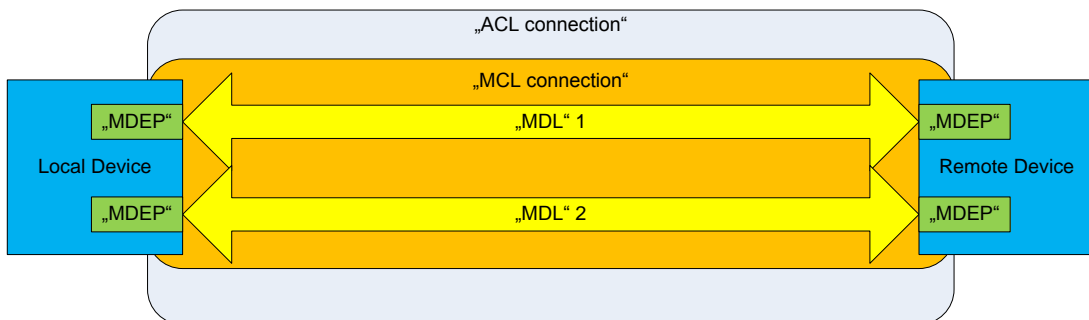


The LTP and BlueHDP+ specification define own terms for the “end” (MDEP) of the “communication channels” (MDL connections), the “profile connection” (MCL connection) and the “radio connection” (ACL connection).

If you translate the drawing with the “old” terms



to the terms that are used in the LTP and BlueHDP+ specifications you will get:



All these different connection types (ACL connection, MCL connection, MDL connection) are built on top of each other. If an ACL connection breaks down (e.g. the remote device comes out of range) all connections on top of this ACL connection (in the drawing one MCL connection with two MDL connections) will break down too. If an MCL connection is disconnected by the remote device, all MDL connections of that MCL will be closed automatically while the ACL connection can still stay connected.

To keep it simple for your implementation, all relevant signaling will be done for MDL connections only, the ACL and MCL signaling (**ACLStatusInfo**, **MCLStatusInfo**) are informal only and may be used for some kind of GUI status mechanisms. This means that if the ACL connection of the example above is disconnected or breaks down, both MDL connections of the example would be indicated to be disconnected separately (one **DisconnectMDLInd** each).

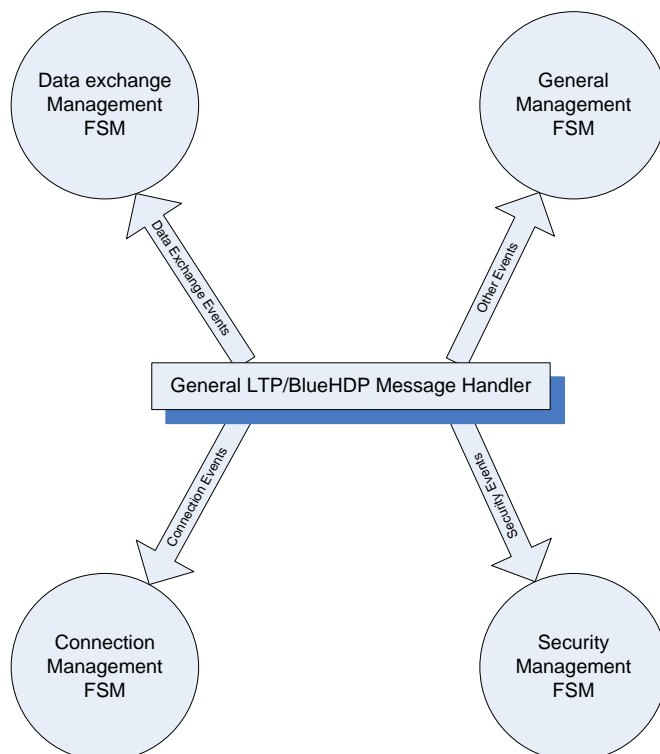
## 6 MDH Implementation Guidelines

BlueHDP+ and LTP are designed to be driven by an MDH (Medical Device Host, the controlling application) that is implemented in a finite-state machine (FSM) style, in fact the best way to implement an MDH to handle security aspects, connection setup and data exchange in logically separated FSMs.

The reason is that these functional aspects are on Bluetooth level independent from each other so it can't be expected that they are synchronized on signaling level.

A good example is security setup during connection setup and data exchange. Depending on the configuration of the local device and the capabilities and timing of the remote device different security procedures must be expected during an ongoing connection setup. The signaling for the connection setup and the security procedures are well defined and deterministic within their own scope, but how they interleave with each other appears to be somehow "random". Sure, if you really dig into it you will find that they are not random at all, but there are many dependencies of the timing and features of both involved devices. Even during data exchange on an already established connection, security procedures must be expected.

Due to this it is much easier to handle connection setup, data exchange and security procedures separately.



### **Connection Events:**

All events defined in chapter "Connection Management,, of the LTP/BlueHDP+ Specification

### **Security Events:**

All events defined in chapter "Security Management,, of the LTP/BlueHDP+ Specification

### **Data Exchange Events:**

All events defined in chapter "Data exchange,, of the LTP/BlueHDP+ Specification

### **Other Events:**

All other events defined in the LTP/BlueHDP+ Specification

---

## 6.1 Device Visibility

The term device visibility describes how other devices can “see” or “find” a given device. The visibility of your device can be controlled with the “**RadioModeSetReq**” message.

In general the Bluetooth SIG and other organizations like Continua recommend that Bluetooth devices should be not visible by default. That means that they should not react to inquiry procedures of other Bluetooth devices. The reason is, that as soon as you have multiple Bluetooth devices in range that do not follow this recommendation, searching for other devices and establishing a connection to one of them can become quite difficult. A device that is designed to be able to search for, pair with and connect to new devices in range needs a minimum GUI capability (see chapter 7.1 Example GUI: Inquiry and Service Discovery), but to minimize user/GUI interaction it is still helpful to be only visible if necessary.

In addition the Bluetooth SIG and other organizations like Continua also recommend that Bluetooth devices should not accept pairing attempts of remote devices by default. Due to this recommendation Bluetooth devices usually have two operation modes, the pairable and the non pairable mode.

### 6.1.1 Pairable Mode

In pairable mode a device is visible and allows pairing with remote devices. In this mode a device can be found by any other remote device in range and allows pairing attempts from remote devices. It may also allow the user to search for remote devices and attempt a pairing with them. In pairable mode usually some Bluetooth specific user interactions and GUI support (for authentication and the search for remote devices) are required. Usually this mode is only used for initial setup but not in the “every day use” of a device. Due to this, the pairable mode should not be the default operating mode of a device and should require some specific user interaction to be enabled. This could be the pressing of a specific button or button combination. Another example might be that a device is in pairable mode for a defined time period after the insertion of batteries.

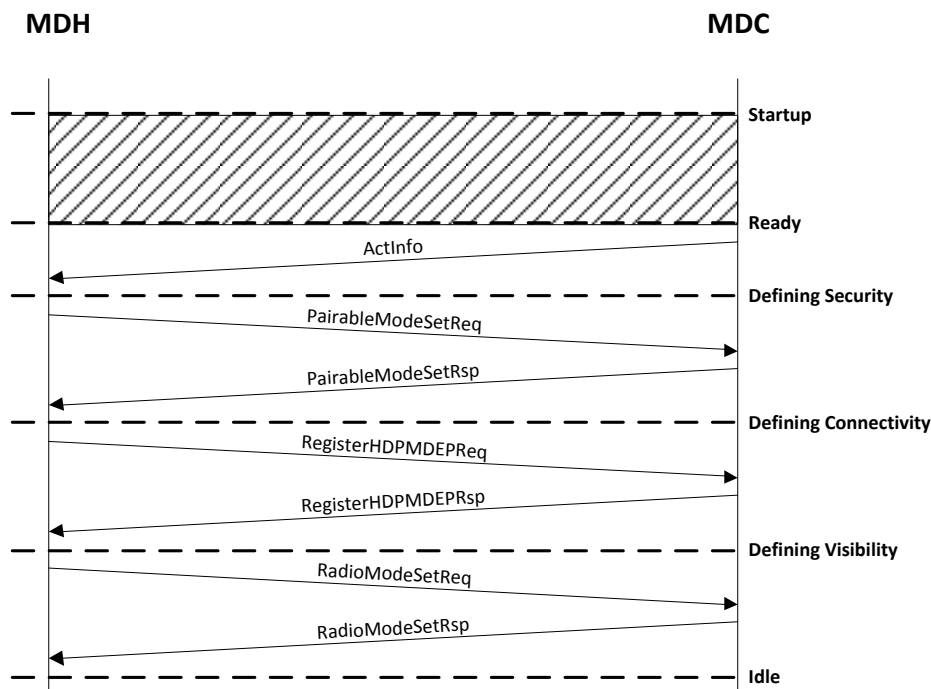
Usually the pairable mode is automatically terminated after a certain time interval (e.g. 2 Minutes) or when a pairing procedure is completed (successfully or not successfully).

### 6.1.2 Non Pairable Mode

In non pairable mode a device is connectable and allows connections from or to already known remote devices, but is not visible and does not allow pairing with new remote devices. In this mode a device does not need any Bluetooth specific user interaction or GUI support. It can automatically accept or build up connections with already known and paired remote devices. This mode should be the default operating mode of a Bluetooth device.

## 6.2 System Startup and Configuration

To communicate with the MDC the first thing that has to be done is to configure the MDC. This MDC configuration should be done in the following order:



For recommendations on how to perform these configurations please refer to the following chapters.

### 6.2.1 MDC Startup

The MDH should first perform its own startup, then enable communication with the MDC (release UART hardware flow control for LTP designs or perform BlueHDP+ registering procedure for BlueHDP+ designs) and then wait for the **ActInfo** messages. In some LTP designs there might be a “hen and egg” situation here for the case that the MDC startup did happen while the MDH was not ready to receive the **ActInfo** message of the MDC.

In case an LTP driven MDH suspects that it missed the **ActInfo** message of the MDC, this LTP driven MDH can initiate an “asynchronous” LTP **ResetReq** to synchronize with the MDC. For details see LTP **ResetReq** message description in [X1].

The first message an MDH will receive after a MDC startup is the **ActInfo** message. This message indicates that the Bluetooth stack is running and it includes some

---

useful information that is necessary to communicate with the MDC. The MDH must wait for this **ActInfo** message before making an attempt to communicate with the MDC. For further information regarding the **ActInfo** message and all other messages described in this document please refer to [X1] or [X2].

### 6.2.2 Defining Security

As described in chapter [6.4 Security Management] and [6.4.2 SSP Device Types] some aspects have to be considered regarding security configuration of the own device. Please refer to those chapters.

### 6.2.3 Defining Connectivity

As described in chapter [6.3 Connection Management] some aspects have to be considered regarding Connectivity configuration of the own device. Please refer to that chapter.

### 6.2.4 Defining Visibility

As described in chapter [6.1 Device Visibility] some aspects have to be considered regarding visibility configuration of the own device. Please refer to that chapter.

## 6.3 Connection Management

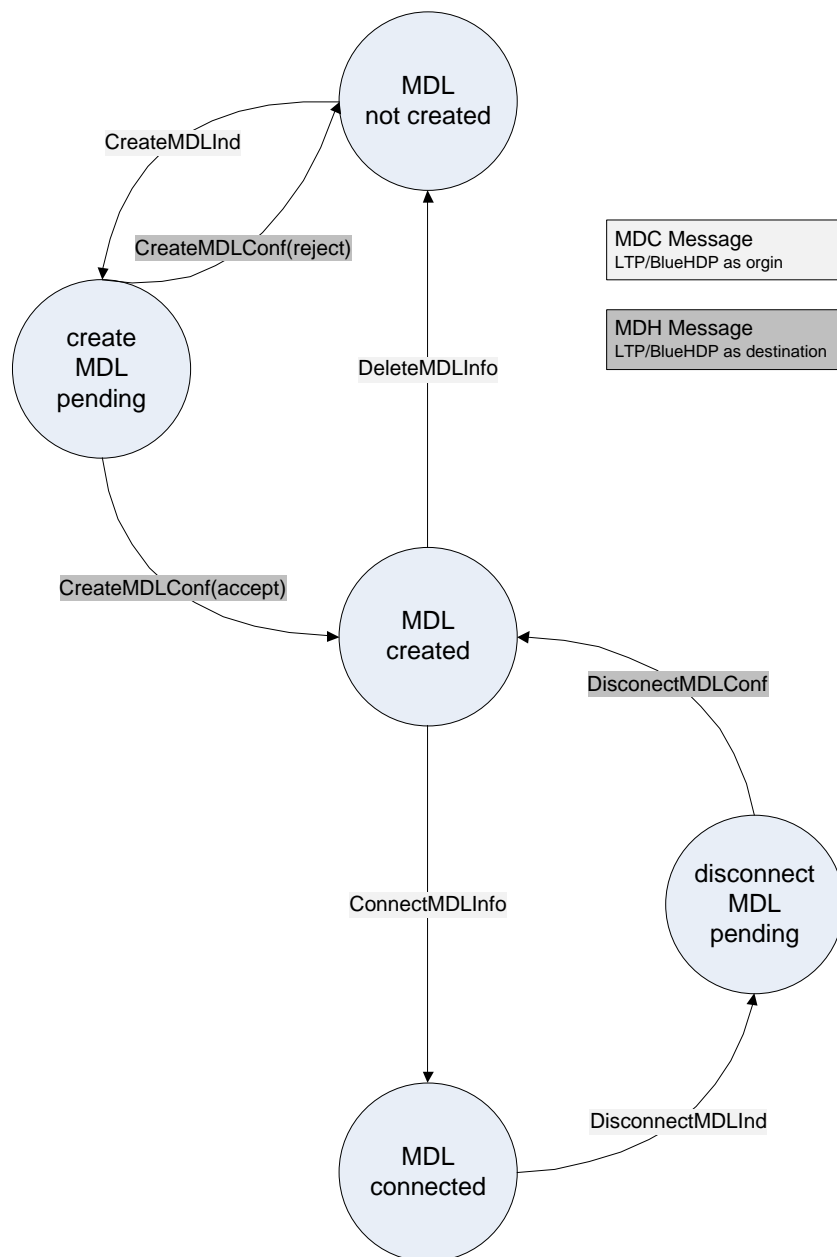
Before a connection can be initiated, some information from the remote peer is needed. This includes the Bluetooth address, an MDEP identifier and some PSM identifiers. How to find this information is described in chapter [7.1.2 Flowchart for Inquiry with LTP/BlueHDP+] and following chapters. Due to this needed information it is much simpler to accept an incoming connection than to initiate a connection since all what is needed to accept an incoming connection is registering an MDEP (**RegHDPMDEPReq** or **RegSPPMDEPReq**) that a remote device can connect to.

Since all signaling is done for each MDL individually, this can be considered by designing your connection management FSM in an MDL context aware way, so it handles the signaling for each MDL separately and does not try to be a monolithic mastermind FSM that keeps track of all possible event combinations in its one single state.

### 6.3.1 Incoming Connection and Disconnection

To allow a remote device to initiate a connection to the local device at least one MDEP **must be registered** first by use of the **RegHDPMDPEPReq** and/or **RegSPPMDPEPReq** message.

Here is an example how such an MDL FSM for **incoming connections** might look like. Data exchange is only possible in MDL connected state. This diagram is valid for HDP and SPP connections but does not consider the optional HDP reconnect functionality.

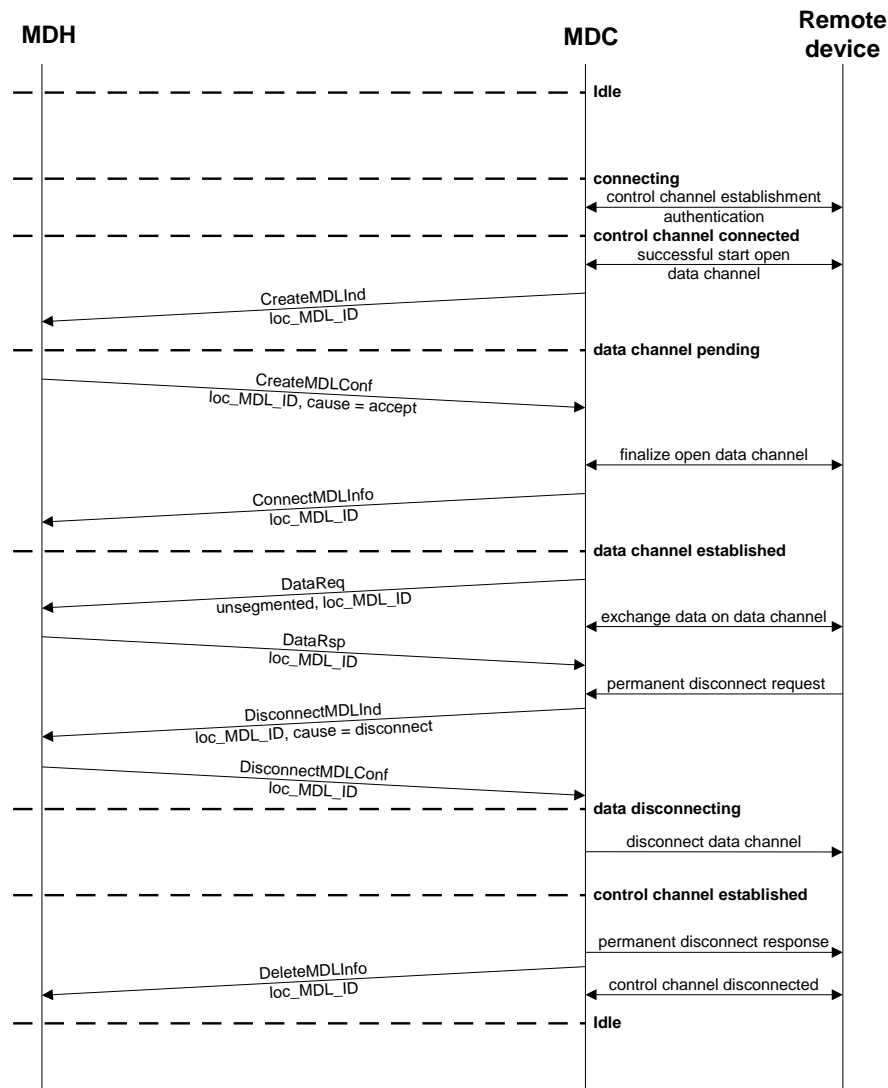


Following is an example how a signaling for a successful incoming HDP connection might look like. For an SPP connection the MDH <-> MDC signaling is equal to a HDP connection but the MDC <-> Remote device communication is deferred due to different procedures on profile level.

**Description:** The creation of a new MDL context is indicated (**CreateMDLInd**) and the new MDL is indicated to be ready for data traffic (**ConnectMDLInfo**).

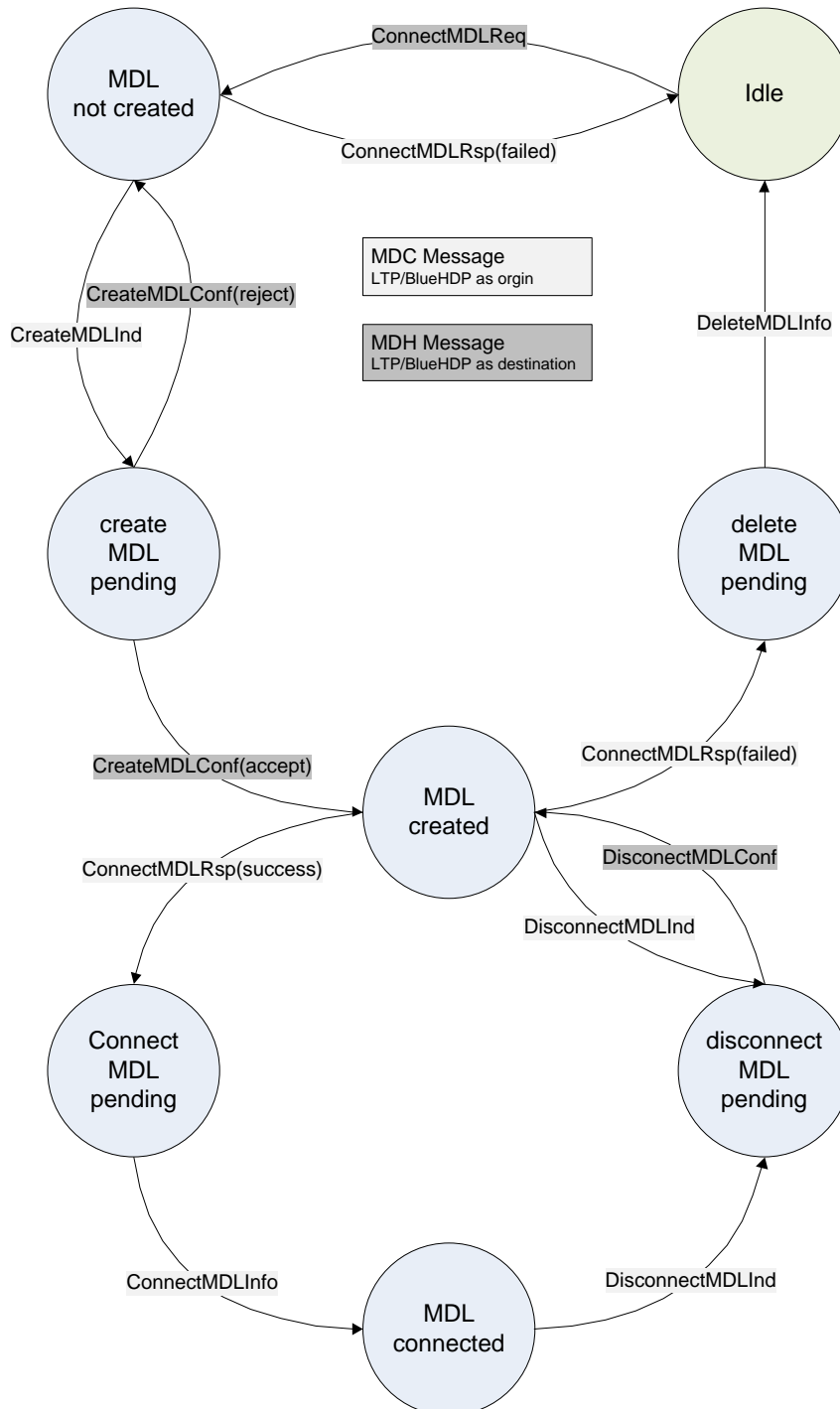
After some data exchange a disconnection is indicated (**DisconnectMDLInd**) and the invalidation of the MDL context is notified (**DeleteMDLInfo**).

During this process multiple **ACLStatusInfo** and **MCLStatusInfo** messages may be received. The ACL/MCLInfo messages that are indicated depend on the actual configuration of the target system.



### 6.3.2 Outgoing Connection and Disconnection

Here is an example how such MDL FSM for **outgoing connections** might look like. Data exchange is only possible in MDL connected state. This diagram is valid for HDP and SPP connections but does not consider the optional HDP reconnect functionality.

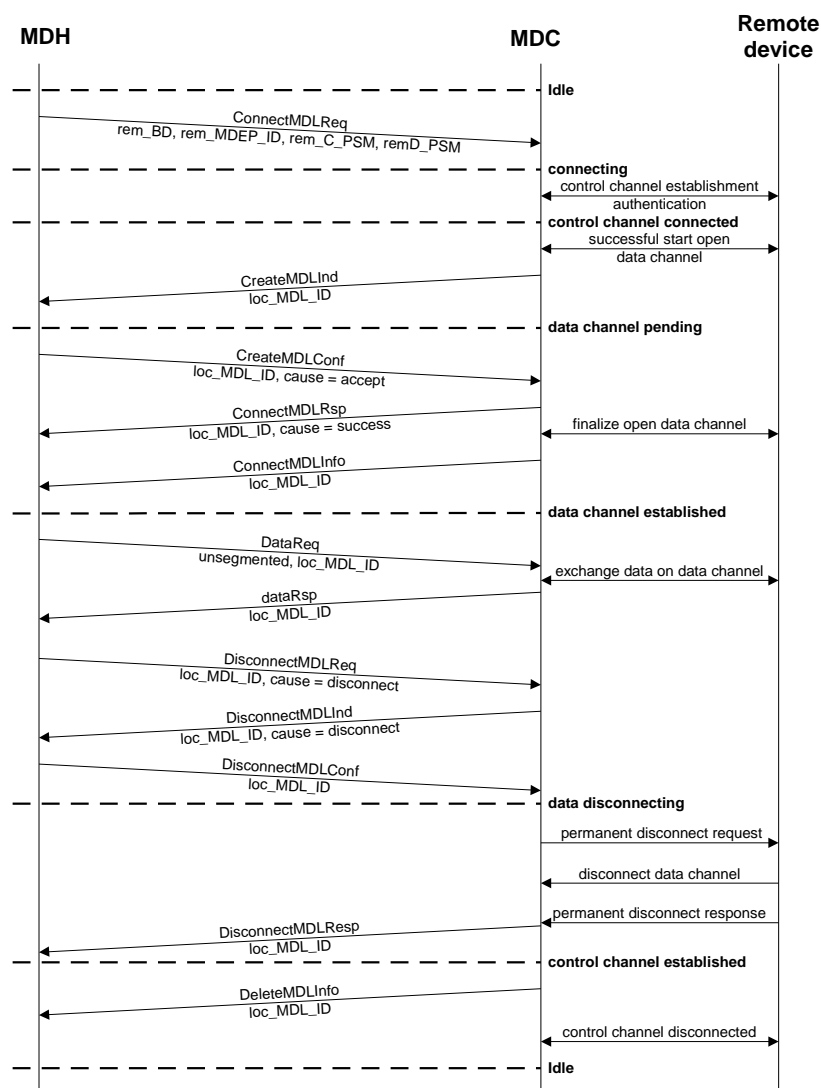


Following is an example how a signaling for a successful outgoing HDP connection might look like. For an SPP connection the MDH <-> MDC signaling is equal to a HDP connection but the MDC <-> Remote device communication is deferred due to different procedures on profile level.

**Description:** A connection to a remote endpoint is initiated (**ConnectMDLReq**), the creation of a new MDL context is indicated (**CreateMDLInd**), the connection setup is completed (**ConnectMDLRsp**) and the new MDL is indicated to be ready for data traffic (**ConnectMDLInfo**).

After some data exchange a disconnection is indicated (**DisconnectMDLInd**) and the invalidation of the MDL context is notified (**DeleteMDLInfo**)

During this process multiple **ACLStatusInfo** and **MCLStatusInfo** messages may be received. The ACL/MCLInfo messages that are indicated depend on the actual configuration of the target system.



---

## 6.4 Security Management

The most important thing to know about the required security support in your implementation is that you only have to support the I/O functionality you claim to support.

Due to that, as a mandatory requirement you only have to support the legacy procedure (**UserAuthRequestInd**) for PIN exchange. If you configure your MDC to “no I/O” (set configurator parameter “biocap” (for descriptions please refer to [X3]) or set parameter “IOCapabilities” of “**PairableModeSetReq**” message to noIOCapabilities) no additional functionality support is required since the only possible SSP scenario is “Just Works” (please refer to [6.4.3.2 Just Works]).

So your security FSM implementation could be quite simple, the only backdrop is that neither BT2.0 legacy pairing (PIN entry) nor BT 2.1 SSP just works pairing do provide “man-in-the-middle” protection. On this point at least some of the readers of this document will raise an eyebrow due to this “man-in-the-middle” term since it might be not completely clear what it means. There are some white papers to describe the security functionality of Bluetooth so it is not intended to fully cover that topic in this document, but some initial information that makes the reading of this SIG white papers and the LTP/BlueHDP specification much easier should be given in the following chapters.

### 6.4.1 Common Terms for Security

#### 6.4.1.1 SSP: Secure Simple Pairing

This term stands for secure simple pairing and describes several pairing mechanisms that are introduced with the Bluetooth v2.1 specification. Details and mechanisms are explained in the next chapters.

#### 6.4.1.2 Legacy Pairing

Legacy pairing is the only method available in Bluetooth v2.0 and before. Each device must enter a PIN code; pairing is only successful if both devices enter the same PIN code. Any 16-byte UTF-8 string may be used as a PIN code, however not all devices may be capable of entering all possible PIN codes, typically a 4 to 6 digit numeric PIN is used

#### 6.4.1.3 MITM: Man-In-The-Middle

In cryptography, the man-in-the-middle attack (often abbreviated to MITM), bucket-brigade attack, or sometimes Janus attack, is a form of active eavesdropping in which the attacker makes independent connections with the victims and relays messages between them, making them believe that they are talking directly to each other over a private connection, when in fact the entire conversation is controlled by the attacker. The attacker must be able to intercept all messages going between the two victims and inject new ones, which is straightforward in many circumstances (for

---

example, an attacker within reception range of an unencrypted Bluetooth connection, can insert himself as a man-in-the-middle).

#### **6.4.1.4 OOB: Out Of Band**

This method uses an external means of communication such as Near Field Communication (NFC) to exchange some information used in the pairing process. Pairing is completed using the Bluetooth radio, but requires information from the OOB mechanism. This provides only the level of MITM protection that is present in the OOB mechanism.

#### **6.4.1.5 Pairing**

A pairing is generally manually started by a device user. If authentication and/or encryption is required either by the profile (e.g. HDP requires authentication and encryption) or the use case, two devices need to be paired to communicate with each other. The pairing process is typically triggered automatically the first time a device receives a connection request from a device with which it is not yet paired. Once a pairing has been established it can be stored non-volatile by the devices (see 5.4.1.6 Bonding”). If two devices are paired (bonded) they can afterwards connect securely to each other without user action unless the pairing relationship was later removed by the user on at least one of the devices.

#### **6.4.1.6 Bonding**

Two devices are “bonded” with each other if they performed a successful pairing procedure and stored the result (see 5.4.1.7 Link key) non-volatile.

#### **6.4.1.7 Link Key**

A link key is the result of a successful pairing procedure and is represented as a 16 byte binary field. A link key does not include any information of the pairing procedure it results from, in can be seen as a 128 bit random number. Together with a link key a device might store additional information from a successful pairing procedure (e.g. device name of the remote device and MITM protection of that procedure). There can be only one link key for a given remote device.

#### **6.4.1.8 Authentication**

In a Bluetooth authentication procedure a common link key is used to verify the identity of the peer device. Only if both devices share the same link key, an authentication can be performed successfully. If an authentication is performed successfully, the authenticated Bluetooth connection provides the same level of security as the pairing procedure that was performed to create the link key. An authenticated link is a prerequisite for an encrypted Bluetooth connection.

#### **6.4.1.9 Authorization**

In Bluetooth context, an authorization is completely independent of all other security procedures discussed here and basically means that the user is asked if a

---

connection from/to a remote peer device is permitted during any connection establishment. No pairing, bonding or authentication is required.

### **6.4.2 SSP Device Types**

Secure simple pairing takes the input and output capabilities of devices in account. For this purpose SSP defines the following Device Types:

#### **6.4.2.1 No I/O**

No user interaction is required. This method is typically used by devices with very limited IO capabilities. It is more secure than the fixed PIN mechanism which is typically used for legacy pairing by this set of limited devices. This appears confusing on the first look but is related to the fact that the key generation mechanism of SSP is much more sophisticated than the mechanism used by legacy pairing. Due to that new mechanism the “randomness” of SSP link keys is much higher (and by that safer) in general. This method provides no man in the middle (MITM) protection.

#### **6.4.2.2 Display Only**

This term describes devices that provide a decimal display with at least 6 digits that can be used for user interaction. If only this display is used for authentication purposes in a security procedure, than this security procedure provides no man-in-the-middle (MITM) protection.

#### **6.4.2.3 Display Yes/No**

This term describes devices that provide a decimal display with at least 6 digits and two keys (“Yes” and “No”) that can be used for user interaction. If the display and the keys are used for authentication purposes in a security procedure, than this security procedure provides man-in-the-middle (MITM) protection. Alternatively to two keys only one key is allowed (“Yes”) if the functionality of the second key (“No”) is realized as a timeout (e.g. 10 seconds no “Yes” key means “No” key pressed)

#### **6.4.2.4 Keyboard Only**

This term describes devices that provide a decimal keyboard (Numbers 0 – 9) that can be used for user interaction. If this keyboard is used for authentication purposes in a security procedure, than that security procedure provides man-in-the-middle (MITM) protection.

### 6.4.3 Security Methods

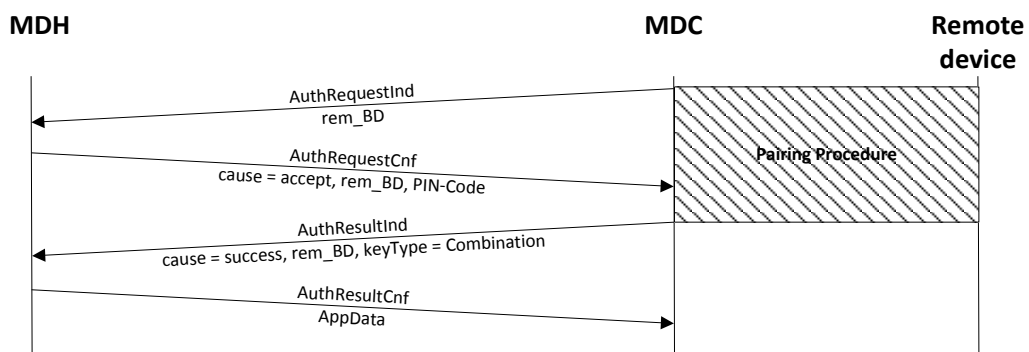
This chapter shows flowcharts for different pairing methods. Please be aware that most of the message naming is in general defined by the Bluetooth specifications and white papers and not result of any Stollmann proprietary naming schemes. If you are interested in more details about these methods please refer to the SIG specifications and whitepapers for this topic.

#### 6.4.3.1 Legacy Pairing

This is the only method available in Bluetooth v2.0 and before. Each device must enter a PIN code; pairing is only successful if both devices enter the same PIN code. Any 16-byte UTF-8 string may be used as a PIN code, however not all devices may be capable of entering all possible PIN codes.

**Here is an example flowchart for a successful legacy pairing:**

If a legacy pairing is initiated (either from the local or from the remote device) an **AuthRequestInd** message is generated by the MDC to signal that a legacy pairing procedure is started. The MDH shall then respond with an **AuthRequestCnf** message that indicates in its cause that either the authentication is rejected or accepted. If the authentication is accepted a PIN that shall be used for the legacy pairing process is included in that message. Finally the result of the pairing attempt is indicated with an **AuthResultInd** message.

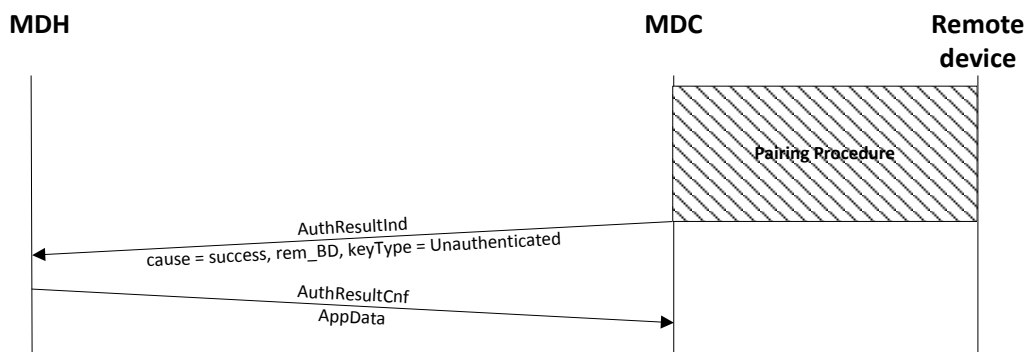


### 6.4.3.2 Just Works

As implied by the name, this method just works and no user interaction is required

#### Here is an example flowchart for a successful just works pairing:

If a SSP pairing is initiated (either from the local or from the remote device) and the combined I/O capabilities of both involved devices just allow just works pairing then this pairing method will be performed without any host or user interaction required. Finally the result of the pairing attempt is indicated with an ***AuthResultInd*** message. Since just works pairing always works success will be indicated.



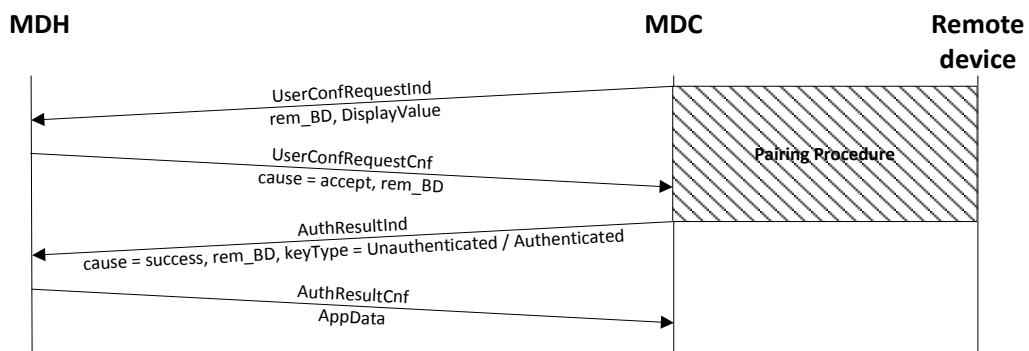
### 6.4.3.3 Numeric Comparison

If both devices have a display and at least one can accept a binary Yes/No user input, they may use numeric comparison. This method displays a 6-digit numeric code on each device. The user should compare the numbers to ensure they are identical. If the comparison succeeds, the user(s) should confirm pairing on the device(s) that can accept an input. This method provides MITM protection, assuming the user confirms on both devices and actually performs the comparison properly.

**Here is an example flowchart for a successful numeric comparison pairing with a local I/O capability “Display Yes/No”:**

If a SSP pairing is initiated (either from the local or from the remote device) and the combined I/O capabilities of both involved devices allow numeric comparison pairing then this pairing method will be performed.

A **UserConfirmationRequestInd** message is generated by the MDC to signal that a numeric comparison pairing procedure is started. The MDH shall then respond with a **UserConfirmationRequestCnf** message that indicates in its cause that either the authentication is rejected or accepted. Finally the result of the pairing attempt is indicated with an **AuthResultInd** message.

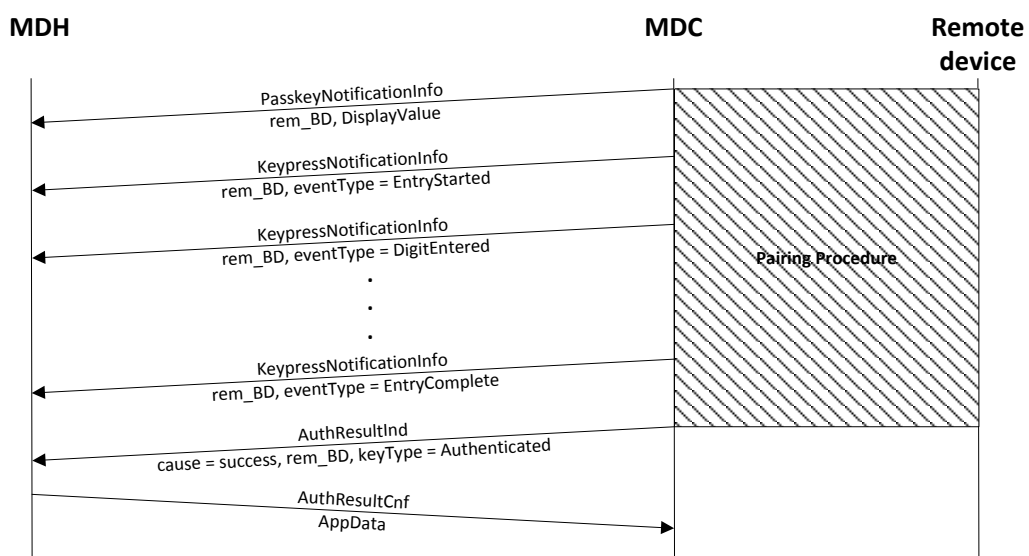


#### 6.4.3.4 Passkey Entry

This method may be used between a device with a display and a device with numeric keypad entry (such as a keyboard), or two devices with numeric keypad entry. In the first case, the display is used to show a 6-digit numeric code to the user, who then enters the code on the keypad of the other device. In the second case, the user of each device enters the same 6-digit number. Both cases provide MITM protection.

**Here is an example flowchart for a successful pairing procedure with SSP “Passkey Entry” and the local device I/O Capabilities “DisplayYesNo” or “DisplayOnly”:**

If an SSP pairing is initiated (either from the local or from the remote device) and the combined I/O capabilities of both involved devices allow “Passkey Entry” pairing then this pairing method will be performed. A **PasskeyNotificationInfo** message is generated by the MDC to signal that a passkey entry pairing procedure is started and provides a value that shall be displayed to the user. Then several **KeypressNotificationInfo** messages can be initiated by the MDC as a result of a corresponding signal from the remote device. The MDH can use this information to provide feedback onto its display (e.g. “greying out” of digits that are typed in on the remote devices keyboard). It is important to understand that only the information that ANY key was pressed on the remote device, but not WHICH key was pressed is indicated here. Finally the result of the pairing attempt is indicated with an **AuthResultInd** message.



---

#### **6.4.3.5 Out of Band (OOB)**

This method uses external means of communication, such as Near Field Communication (NFC) to exchange information used in the pairing process. Pairing is completed using the Bluetooth radio, but requires information from the OOB mechanism. This provides only the level of MITM protection that is present in the OOB mechanism.

---

#### 6.4.4 Data Exchange Management

Since all signaling is done for each MDL individually, this can be considered by designing your connection management FSM in an MDL context aware way, so it handles the signaling for each MDL separately and does not try to be a monolithic mastermind FSM that keeps track of all possible event combinations in its one single state.

During data exchange the most important things to be monitored are MDL flow control and APDU segmentation and reassembly.

##### 6.4.4.1 Flow Control

MDL connections support credit based flow control in downstream direction (from MDH to the MDC) and in upstream direction (from MDC to the MDH).

For downstream data direction the **ConnectMDLInfo** message includes a `maxTPDUUsCredits` parameter that indicates how many unacknowledged data messages are allowed to be sent by the MDH at any time.

For upstream direction the **ConnectMDLInfo** message includes a `maxTPDUUsCredits` parameter that indicates how many unacknowledged data messages will be indicated by the MDC to the MDH before it applies backpressure to the remote device by not allowing the transfer of additional data over the Bluetooth air interface.

Both parameters, `maxTPDUUsCredits` and `maxTPDUUsCredits`, are MDL specific and are a result of the MDL creation process (see **CreateMDLInd** and **CreateMDLConf** messages) based on the general capabilities and resources of the local and the remote device.

For more detailed information about flow control mechanisms please refer to the corresponding chapters of the BlueHDP+ [X2] or LTP specification [X1].

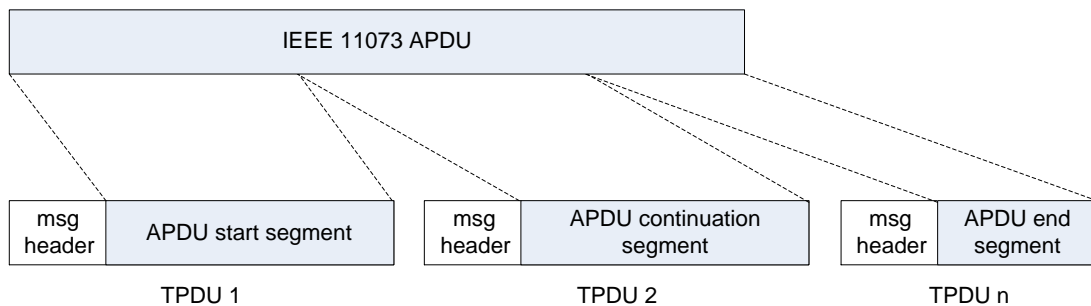
#### 6.4.4.2 APDU Segmentation and Reassembly

LTP and BlueHDP+ is designed to allow packet based data exchange. While SPP itself is defined as a “byte stream” transport (like a serial cable), HDP mandates that IEEE 11073 packets (APDUs) are exchanged by use of this packet exchange mechanism. Small APDUs can be exchanged as simple “not segmented” data exchange messages but since IEEE 11073 APDUs can be as large as 63kB a segmentation and reassembly mechanism is needed.

To exchange IEEE 11073 APDUs via a given HDP based MDL connection that are larger than the signaled TPDU size of that MDL connection a segmentation and reassembly mechanism has to be used and implemented.

To send such a large 11073 APDU, that APDU has to be splitted into multiple data exchange messages that indicate if they provide the first data of a new APDU (start segment), continuation data of an already started APDU (continuation segment) or the last data of an already started or continued APDU (end segment).

Received large APDUs will be signaled in segments with the same mechanisms.



If multiple MDL connections are established, segments of APDUs can/will be signaled interleaved in both directions simultaneously.

Since SPP does not define an APDU segmentation and reassembly mechanism, SPP based MDL connection support only not segmented APDU exchange so the maximum APDU size for a given MDL is limited to the TPDU size of that MDL.

For more detailed information about APDU segmentation and reassembly mechanisms please refer to the corresponding chapters of the BlueHDP+ [X2] or LTP specification [X1].

---

## 6.5 LTP Specifics

For LTP communication Stollmann provides a software library “lplib” in source that is designed to for an easy integration into your application. The library enables you to send and receive LTP messages without having to deal with a lot of issues that are related to a reliable asynchronous communication via a UART. It is highly recommended that this library is used by any customer that intends to build a LTP based product.

Regarding flow control mechanisms it is highly recommended to implement the “credit based” software flow control mechanism for payload data provided by the LTP protocol and not to rely on the hardware flow control instead. One of the reasons that lead to this recommendation is that you will not be allowed to send a ***DisconnectMDLReq*** or any other message when hardware flow control is triggered by the MDC since hardware flow control does not only affect payload data exchange but also signaling traffic.

## 6.6 BlueHDP+ Specifics

For BlueHDP+ payload data exchange it has to be highlighted, that ***DataReq*** messages differ from all other BlueHDP+ messages regarding the buffer management. The intention is to enhance system speed and efficiency by avoiding the need to copy payload data. For more details regarding this topic please refer to chapter [Data Handling and Flow Control] of the BlueHDP+ Specification [X2].

---

## 7 Example for GUI Representation

This chapter helps you to understand what is necessary and possible with a Bluetooth transport and what kind of user / software interaction is / can be required.

If your use case just requires the acceptance of (incoming) connections only security related GUI interactions are required (e.g. PIN entry, manual confirmation of connection by user).

If your use case requires the establishment of (outgoing) connections from your implementation than a lot of additional user interaction is required.

### Example:

If you want to use a USB cable based HDP device, you enter the room where the device is located. You look at all devices in that room and choose a device you wish to use.

If you see a candidate you pick it up and check the label if the device has the necessary capabilities.

If it supports the services you need you may have to ask someone if you are allowed to use that device.

If you get an OK (you may have to answer some security related questions) you look for the USB plug and plug in your cable, the USB shim/driver establishes a transport connection and, in case the device supports IEEE 11073, the IEEE layer kicks in.

This whole process has its analogy in Bluetooth, it is just that you can sit at your desk while doing all the things since Bluetooth is wireless and does not need physical access. The keywords here are inquiry, service discovery, endpoint and security procedures.

### 7.1 Example GUI: Inquiry and Service Discovery

In Bluetooth context an **inquiry** is a mechanism where the inquiring device initiates a broadcast call for devices in range. All devices that are set to “visible” and notice the call will answer with some device specific information. Such an inquiry takes about 10 seconds, regardless of how many devices are in range.

The information for each device is usually quite limited and includes:

- a worldwide unique “Bluetooth device address” (also called “BD”)
- a 24 bit class of device (also called “COD”) that can give some hints of the type of device that is found
- a 8 bit remote signal strength indicator (also called “RSSI”) that represents the signal quality of the device found

- and optionally a UTF-8 coded device name of the device

On the “pro” side an **inquiry** is quite fast. For a user the time should be acceptable, especially if a GUI shows some type of progress indication (e.g. live updated list of devices found, live updated number of devices found,...)

On the “contra” side an **inquiry** does not give sufficient information to allow the user to select the device he wants to connect to, especially in an environment where multiple devices are found.

Experience shows that in an environment with many remote devices in a large room (e.g. an (Un)Plugfest) it can happen that the devices that are almost out of range can still respond to an inquiry (a broadcast mechanism that makes the air glow without requiring a point to point connection) but might not be in save range for any activity that requires a reliable point to point connection (e.g. **DeviceNameReq**, **HDPDiscoveryReq**, **SPPDiscoveryReq** or **CreateMDLReq**).

In such a case the success rate for connection attempts to such devices can drop significantly. Unfortunately there is no sharp line for devices in range and devices out of range but an uncertainty zone in between where connection attempts might or might not succeed.

One possible solution here is that inquiry results are filtered by their RSSI value that is part of an **InquiryDeviceInfo** message.

The RSSI value can be seen as a hint if it makes sense to try to connect to a device found by an inquiry. Devices with a bad RSSI Value are probably almost out of range and might be filtered away or grayed out on GUI level.

At least it might be considered to not connect to such an “almost out of range” device automatically to refine an inquiry result with additional information (e.g. device name) since every failed connection attempt requires a substantial amount of time (by default about 5 seconds) that can sum up to a significant amount of time for a larger list of found devices.

To gather additional information of a remote device a service discovery to that specific device is required.

In Bluetooth context a service discovery is a mechanism to read out service and device specific information from a specific device. The main information needed to perform a service discovery is the Bluetooth address of the remote device. To perform a service discovery, a bi-directional point-to-point SDP profile level connection is established. The SDP functionality is then used to read specific information out of the remote devices SDP service records (e.g. DID Service Record, HDP Service Records, SPP Service records...).

On the “pro side” a service discovery can gather all information of a remote device that is necessary to fully represent that device on GUI level and allow the user to choose the exact required functionality to connect to.

---

On the “contra” side a service discovery requires a point-to-point connection (in opposite to the broadcast mechanism of an inquiry) that can take some time (per device in average about 2 seconds, worst case 20 seconds). Due to that an automatic service discovery for all devices that were found by the inquiry can take an unacceptable (user point on view) amount of time in cases where a lot of Bluetooth devices are in range.

So it may be a good idea to perform inquiry and service discovery in a two-step procedure:

First the user initiates an inquiry. This may be done by pressing a GUI button.

As result an inquiry is started (LTP/BlueHDP+ Message **InquiryReq**) and the devices that are found will be indicated (LTP/BlueHDP+ message **InquiryDeviceInfo**). Our recommendation is that the GUI has a progress indication (e.g. progress bar, live update of device list).

If the inquiry process is finished the GUI asks the user to initiate the service discovery (LTP/BlueHDP+ messages **HDPDiscoveryReq** and/or **SPPDiscoveryReq**) and/or initiate device pairing (LTP/BlueHDP+ message **AuthReq**) for specific devices that the user chooses from the inquiry results.

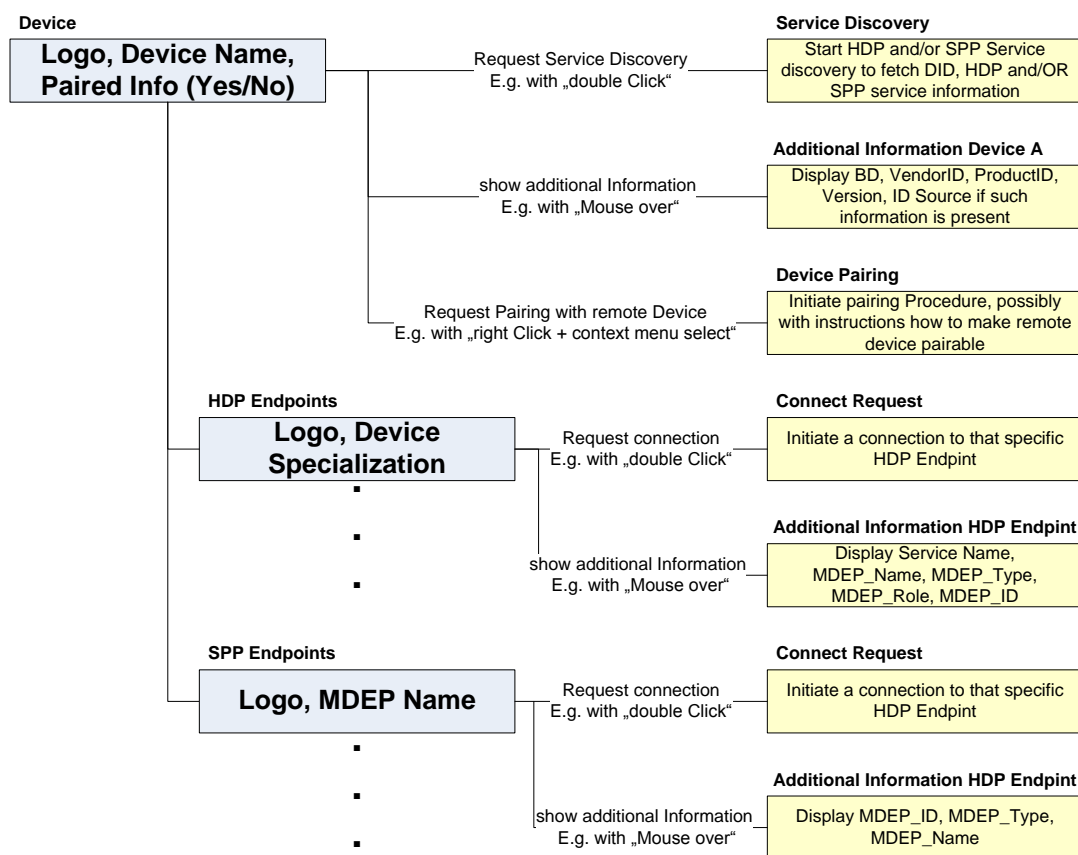
If a service discovery or device pairing is completed, the inquiry result list should be updated with the device specific information (including the endpoints) to represent the specific functionality of the connectable remote device.

### 7.1.1 Example GUI: Graphical Representation

The following drawing shows a proposal for a single device and service representation (blue boxes) on GUI level. The GUI may allow the user to initiate an inquiry by pressing a button. As a result a list of found devices and services are shown; optionally this list may be live updated while processing a user request (e.g. inquiry, service discovery).

If the processing of a user request is finished the drawing shows a proposal for some user interactions (text between lines) with interactive reactions of the system (yellow boxes).

Such a device list should be designed to show a large number of devices, each device presenting several services/endpoints:

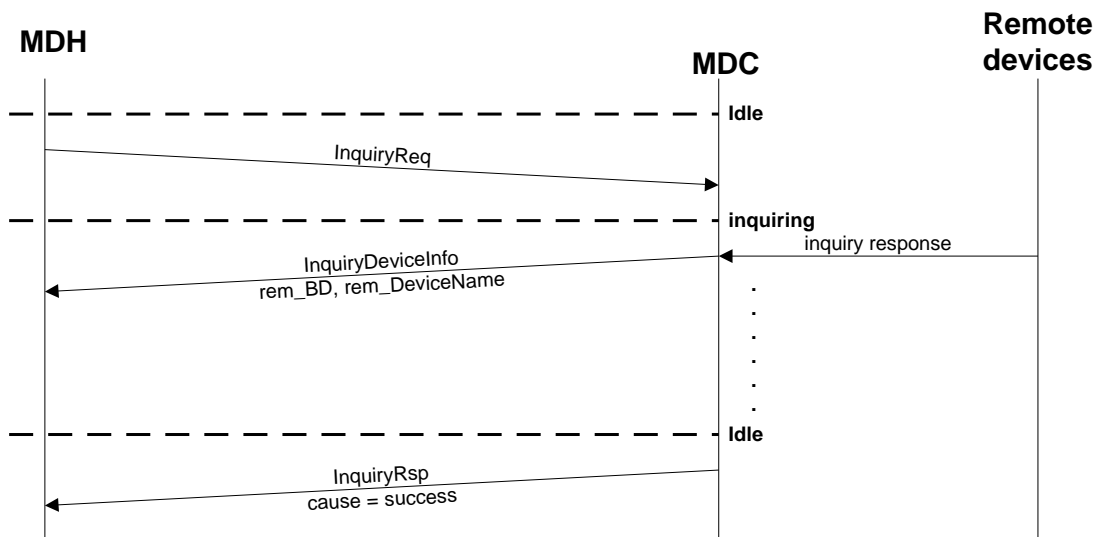


To gather the information required for this representation the inquiry and service discovery mechanisms of LTP/BlueHDP+ have to be used.

### 7.1.2 Flowchart for Inquiry with LTP/BlueHDP+

This chapter shows a flowchart drawing for an inquiry process:

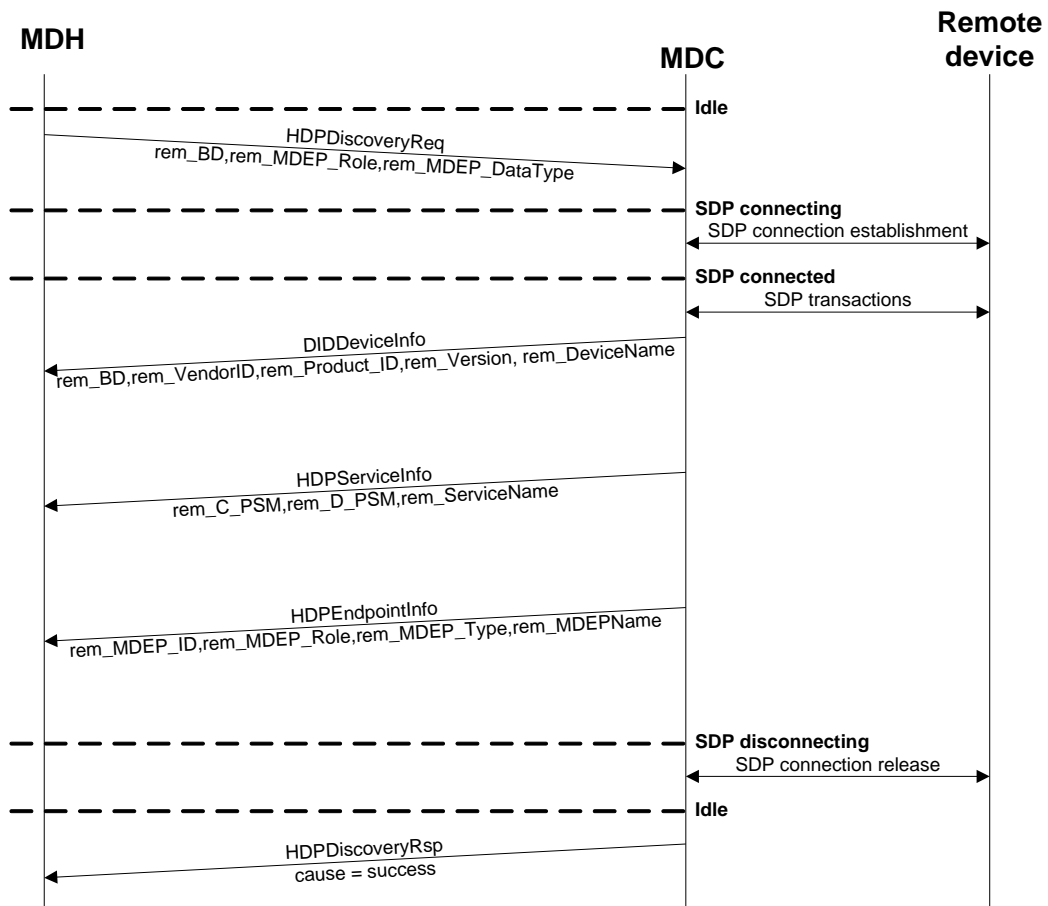
An inquiry is initiated (*InquiryReq*), multiple devices are found (*InquiryDeviceInfo*), the inquiry ends (*InquiryRsp*).



### 7.1.3 Flowchart for HDP Service Discovery with LTP

This chapter shows a flowchart drawing for an HDP service discovery process.

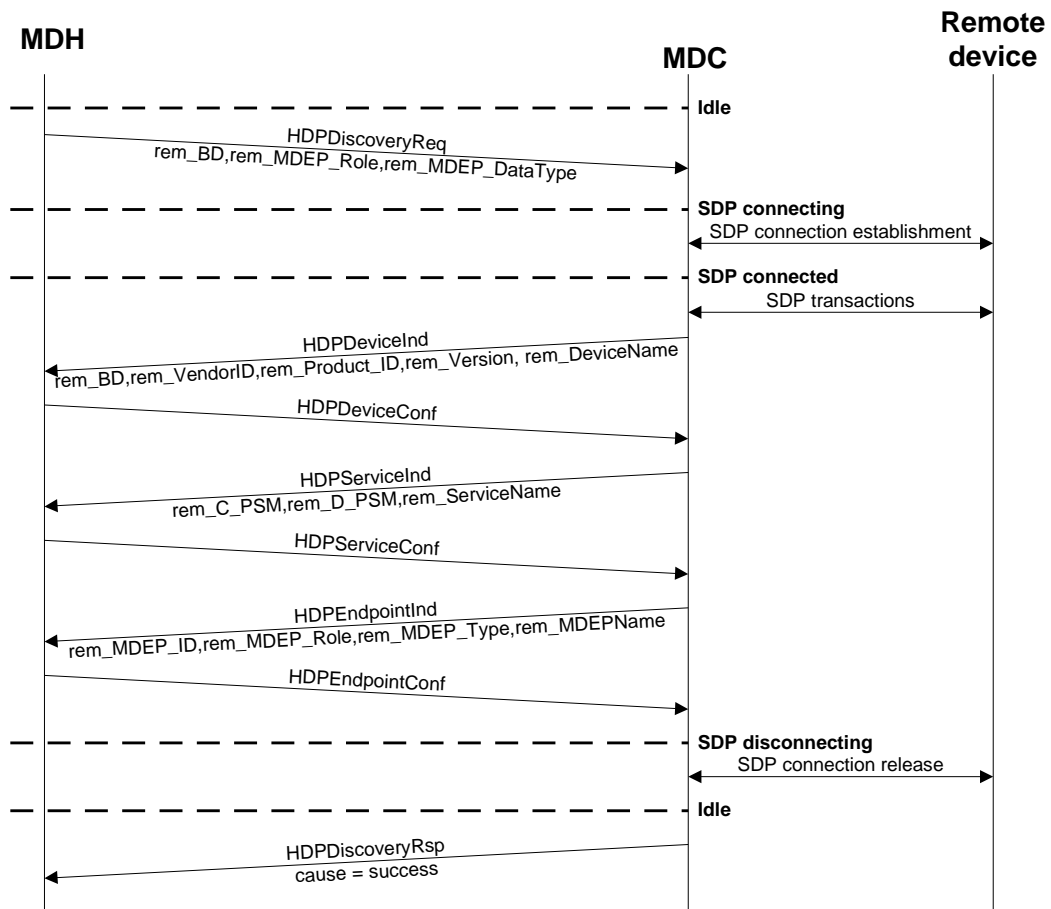
An HDP service discovery is initiated (**HDPDiscoveryReq**), device information is found (**DIDDeviceInfo**), one HDP service is indicated (**HDPServiceInfo**, please be aware that multiple services can be present), one HDP endpoint is indicated (**HDPEndpointInfo**, please be aware that multiple endpoints can be present per service) and the HDP service discovery ends (**HDPDiscoveryRsp**):



### 7.1.4 Flowchart for HDP Service Discovery with BlueHDP+

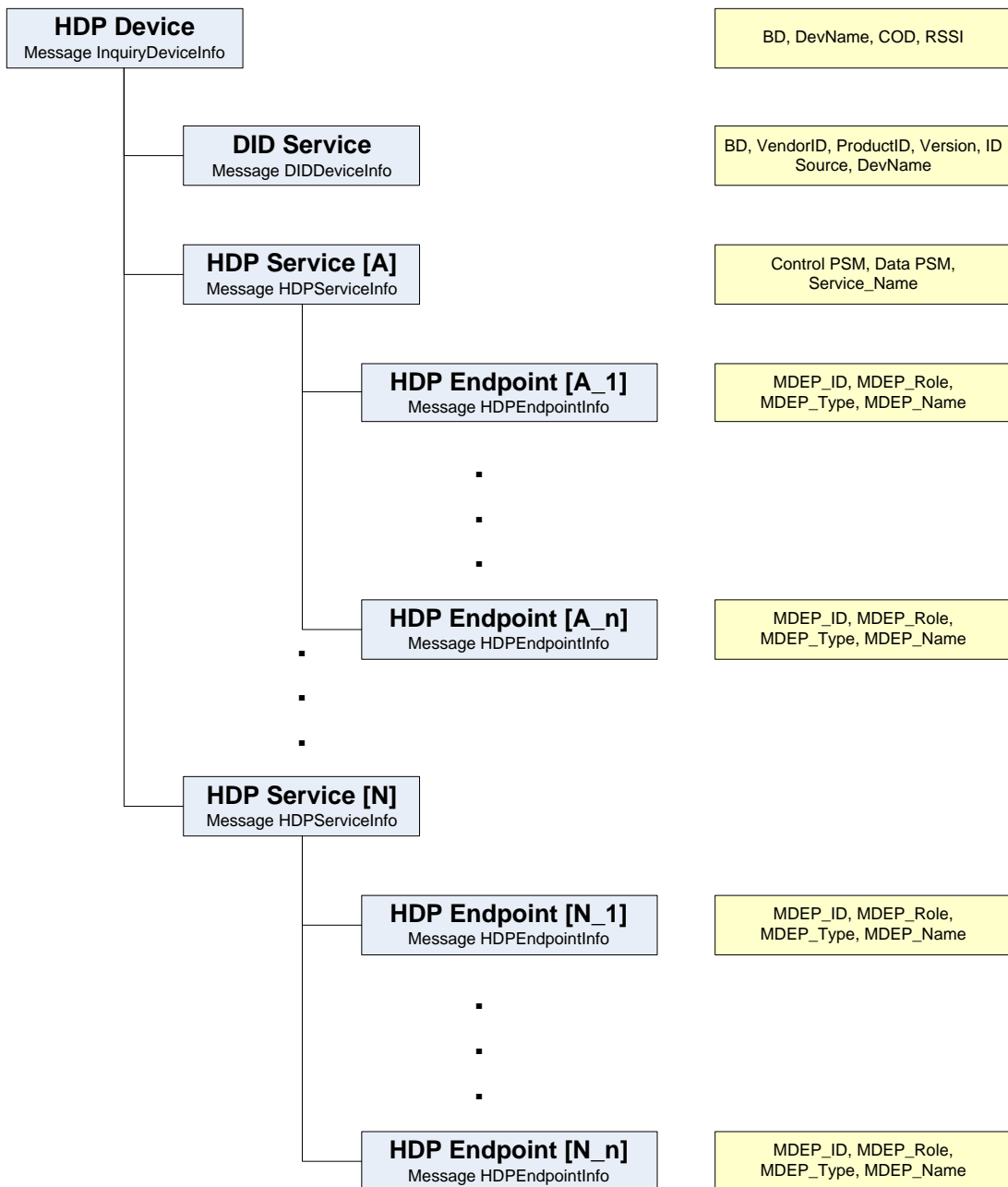
This chapter shows a flowchart drawing for an HDP service discovery process.

An HDP service discovery is initiated (**HDPDiscoveryReq**), device information is found (**DIDDeviceInd**), one HDP service is indicated (**HDPServiceInd**, please be aware that multiple services can be present), one HDP endpoint is indicated (**HDPEndpointInd**, please be aware that multiple endpoints per service can be present) and the HDP service discovery ends (**HDPDiscoveryRsp**):



### 7.1.5 Logical data Structures for HDP Service Discovery

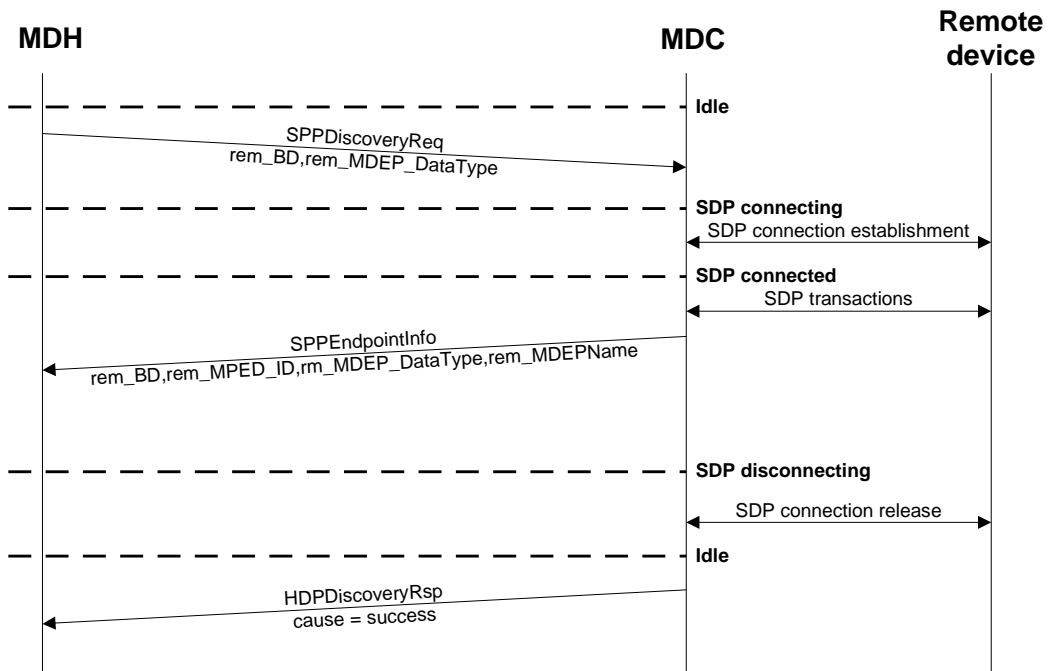
An overview of the relation between LTP/BlueHDP+ messages (in the blue boxes) and included information (in the yellow boxes) is shown in the following drawing:



### 7.1.6 Flowchart for SPP Service Discovery with LTP

This chapter shows a flowchart drawing for an SPP service discovery process.

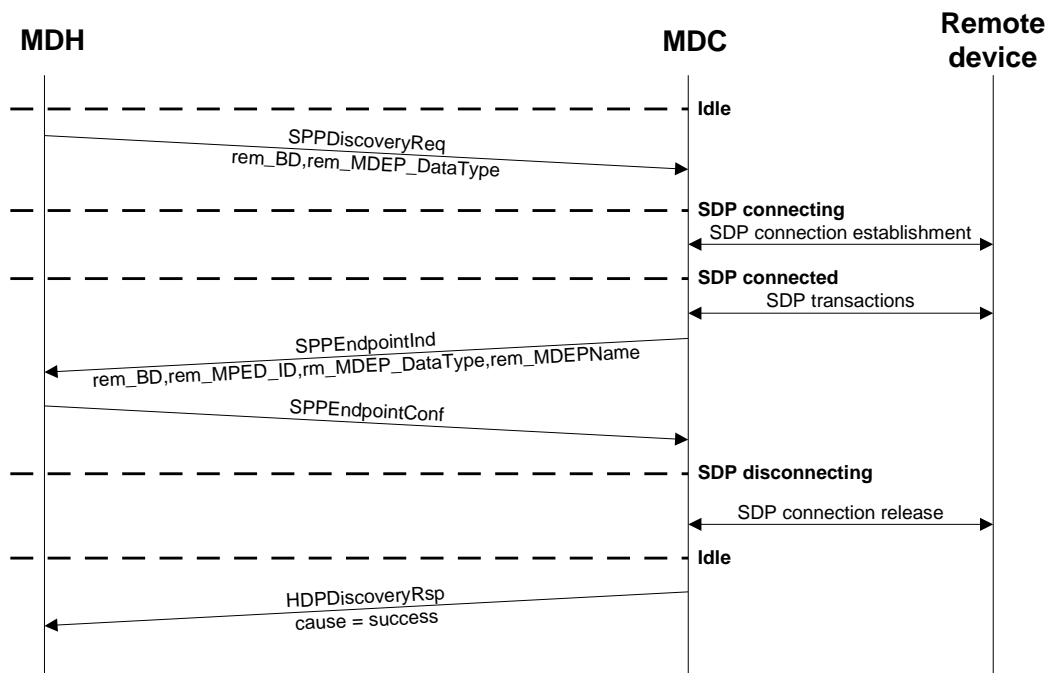
An SPP service discovery is initiated (**SPPDiscoveryReq**), one SPP endpoint is indicated (**SPPEndpointInfo**, please be aware that multiple endpoints can be present) and the SPP service discovery ends (**SPPDiscoveryRsp**):



### 7.1.7 Flowchart for SPP Service Discovery with BlueHDP+

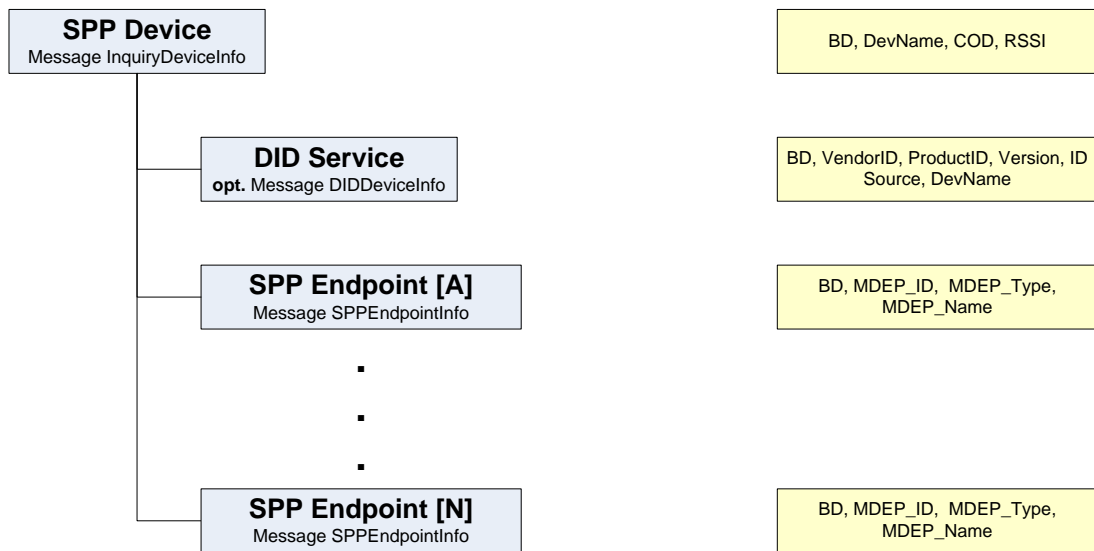
This chapter shows a flowchart drawing for an SPP service discovery process.

An SPP service discovery is initiated (**SPPDiscoveryReq**), one SPP endpoint is indicated (**SPPEndpointInd**, please be aware that multiple endpoints can be present) and the SPP service discovery ends (**SPPDiscoveryRsp**):



### 7.1.8 Logical Data Structures for SPP Service Discovery

An overview of the relation between LTP/BlueHDP+ messages (in the blue boxes) and included information (in the yellow boxes) is shown in the following drawing:

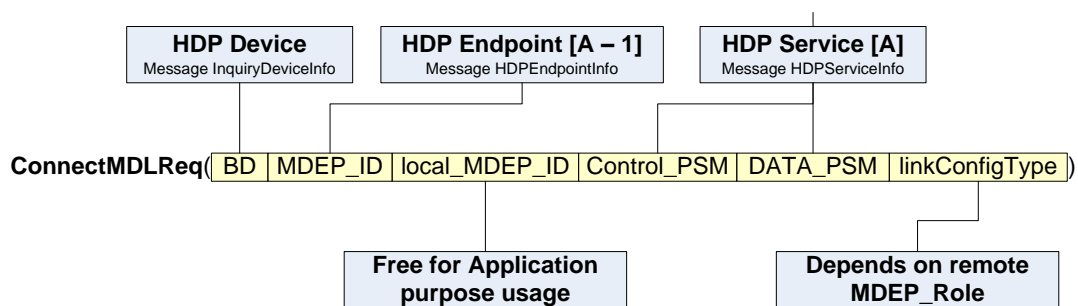


## 7.2 Connecting to Remote Devices

A complex GUI as proposed in this document is only necessary if it is intended to enable the user to find remote devices and connect to them. To request a connection several parameters are required, which must be a result of a previously performed inquiry and service discovery. The following chapters show the relation between inquiry/service discovery information and a connection request.

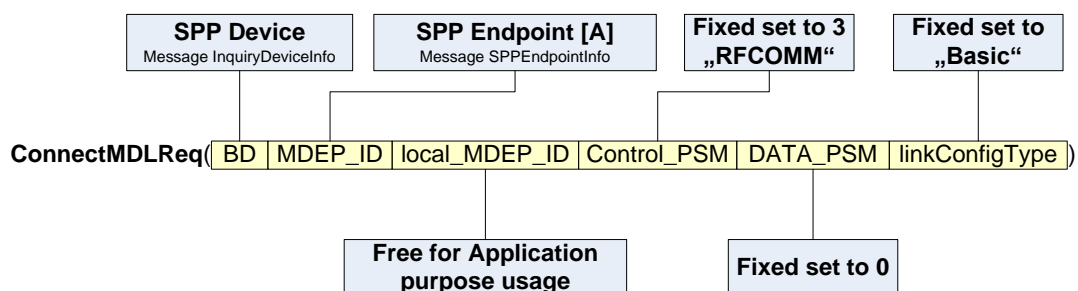
### 7.2.1 Connection to HDP Endpoint

An overview of the relation between LTP/BlueHDP+ messages (in the blue boxes) and included information (in the yellow boxes) is shown in the following drawing:



### 7.2.2 Connecting to SPP Endpoint

An overview of the relation between LTP/BlueHDP+ messages (in the blue boxes) and included information (in the yellow boxes) is shown in the following drawing:



## 7.3 References

- [X1] LTP Interface Specification, Stollmann
- [X2] BlueHDP+ API Documentation, Stollmann
- [X3] BlueMod+HDP\_Command\_Reference, Stollmann

---

## 8 History

Version	Release Date	By	Change description
r01	15.02.2011	ka	Initial Version

Stollmann Entwicklungs- und Vertriebs-GmbH  
Mendelssohnstraße 15 D  
22761 Hamburg  
Germany

Phone: +49 (0)40 890 88-0  
Fax: +49 (0)40 890 88-444  
E-mail: [info@stollmann.de](mailto:info@stollmann.de)  
[www.stollmann.de](http://www.stollmann.de)