25 September 2008

Status
Approved

Revision
V10r00

Document File Name
Discovery_WP_V10.doc

Document Owner
BARB

E-mail Address
BARB-main@bluetooth.org

Bluetooth SIG

DISCOVERY WHITEPAPER

Service Discovery Applications

ABSTRACT: This document defines the features and procedures for an application in a *Bluetooth*® device to discover services registered in other Bluetooth devices and retrieve any desired available information pertinent to these services.

Revision History		
Revision	**Date**	**Description**
D01r01	3 August 2007	Adapted from SDAP spec
D01r02	17 August 2007	Converted to white paper format
D01r15	2 Feb 2008	Incorporated comments from Burch, Henrik, Len, Tim and Terry as much as possible and accepted all changes to clean up document
D01r16	2 April 2008	Remaining BARB comments addressed with responses-in-comments and text edits
D01r17	July 2008	Updated following comments from Joel Linsky
D01r18	August 2008	Adjust figures 4.1 and 4.2
D01r19	August 2008	Includes edits from Burch Seymour
D01r20	August 2008	Includes edits from Terry Bourk
V10r00	September 2008	Approved for publishing

Contributors	
Name	**Company**
Terry Bourk	QUALCOMM
Tim Howes	Symbian
Burch Seymour	Continental Automotive Systems

CONTENTS

1 Introduction

1.1 SCOPE

The purpose of this document is to provide advice to implementers of service discovery applications. The document covers the use of SDP and EIR for passing device and service information from one device to another.

1.2 DISCOVERY SERVICES

As the number of services that can be provided over Bluetooth links increases it is becoming more important to help the users to locate, identify and accept the desired services.

The Bluetooth protocol stack defined in the core specification contains:

1. Device discovery services via the Inquiry and Inquiry Scan procedures

2. Extended Inquiry Response (EIR) information packets that may be sent during the Inquiry response procedure to provide the device name and other identifying information during the inquiry procedure (only possible if both devices support EIR)

3. Service Discovery Protocol that is used to locate services that are available on devices in the vicinity of the inquiring device.

Having obtained information about the services available on a remote device, the inquiring device (with or without user involvement) may then elect to use one or more of them. Selecting, accessing, and using a service is outside the scope of this document.

SDP does not initiate a profile connection with a service on a remote device. Instead, SDP returns the information needed by the inquiring device to create a profile to profile connection. For example, in the case of Hands-Free Profile, this information includes the RFCOMM channel number used for the service level connection and a bit mask that describes optional features supported by the service.

This document recommends the protocols and procedures that should be used by a service discovery application to locate services in other Bluetooth-enabled devices using EIR and SDP.

EIR provides the opportunity for devices to obtain the user defined device name and possibly device type and/or service information during the inquiry procedure. Without EIR, name discovery requires a partial connection to the remote device, and service discovery requires a complete connection. The use of EIR will enable a more rapid display of device information to the user. This allows the user to more quickly select devices of interest as targets of full service discovery, provided by SDP.

The service discovery procedures covered in this document can be utilized in profiles. When used by profiles, these procedures are initiated by application-level actions as opposed to user-level actions for service discovery applications.

SDP provides direct support for the following set of service inquiries:

- Search for specified service identifiers;
- Service browsing.

The service discovery application described in this whitepaper covers these scenarios.

The first case represents searching for known and specific services. They provide answers to user questions like: “Is service A available, or is service A with characteristics B and C available?”

The second case represents a general service search and provides answers to questions like: “What services are available?” or “What services of type A are available?”

These service inquiry scenarios can be implemented in two ways:

- By performing the service searches on a single connected device selected by the user, and/or
- By performing the service searches on all devices discovered by an inquiry operation.

1.3 SYMBOLS AND CONVENTIONS

This whitepaper uses the symbols and conventions specified in Section 1.2 of the Generic Access Profile **[3]**.

2 Application Overview

2.1 APPLICATION STACK

Figure 2.1 shows the Bluetooth protocols and supporting entities utilized by service discovery applications.

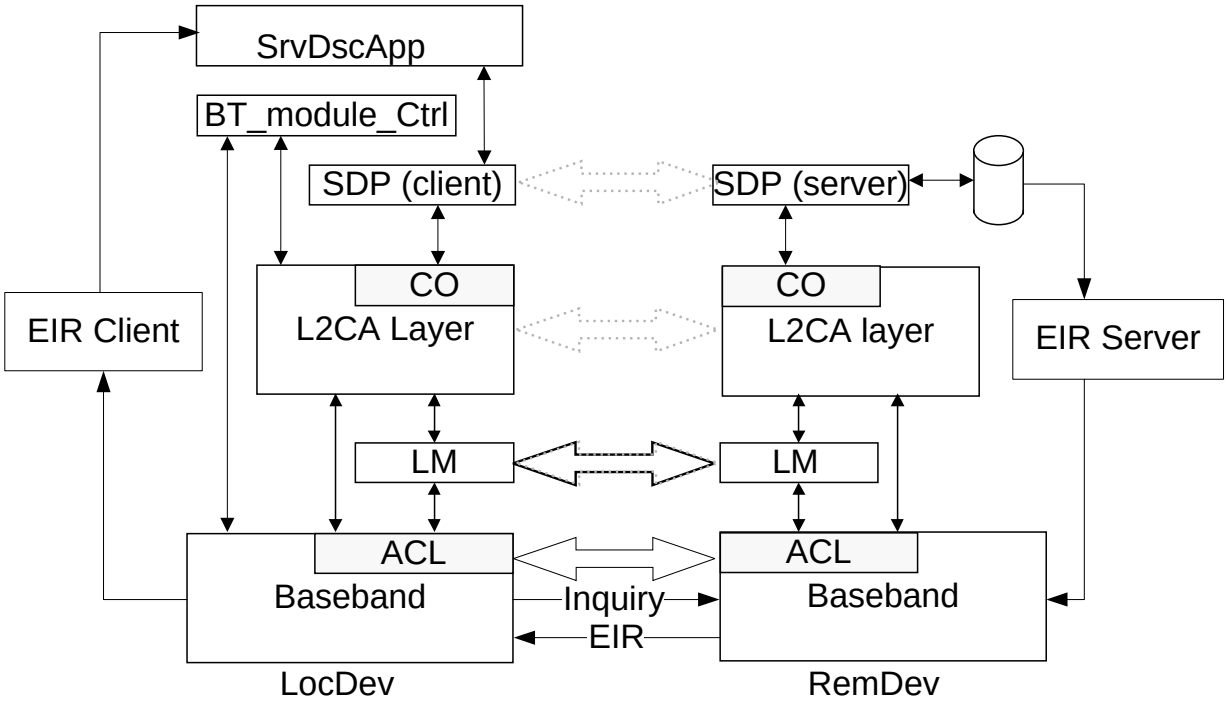

Figure 2.1 The Bluetooth protocol stack for the service discovery application

The service discovery user application (SrvDscApp) in a local device interfaces with the Bluetooth SDP client to send service inquiries to, and receive service inquiry responses from, the SDP servers of remote devices . SDP is described in the Bluetooth Core specification [Host volume], Part B **[7]**. SDP uses the connection-oriented L2CAP to carry the SDP PDUs over the air.

In a typical sequence of operations, a device will use inquiry to locate discoverable Bluetooth devices in the area. The inquiring device may then optionally use a name request to determine the user-friendly name of the discovered devices. Finally, the inquiring device may create a connection with one or more of the discovered device and use SDP to look for a specific service, or to request a list of all supported services. Thus, the SrvDscApp interfaces with the Controller via the BT_module_Ctrl entity, which controls the Bluetooth module and its use of the various search procedures.[1]

The service discovery database (DB), shown in Figure 2.1 next to the SDP server, is a logical entity that serves as a repository of service discovery-related information. The 'physical form' of this database is an implementation issue that is outside the scope of this whitepaper.

1. The BT_module_Ctrl may be part of a Bluetooth stack implementation (and thus be shared by multiple Bluetooth-aware applications) or a 'lower part' of the SrvDscApp. Since, no assumptions are made about any particular stack or SrvDscApp implementations,, the BT_module_Ctrl entity represents a logical entity separate from the SrvDscApp, which may be a part of the SrvDscApp, a stack component, or any other appropriate piece of code.

When both the initiating and responding devices support the Bluetooth Core Specification 2.1 + EDR (or later), it may be possible for the initiator to obtain information about the responder without the need to create a connection. The EIR feature provides a mechanism for the responding device to include name and supported-services information in its inquiry response message. Prior to the Bluetooth Core 2.1 + EDR specification, the only information available was the Bluetooth device address and class of device. Best practices now strongly discourage the use of the Class of Device fields for device filtering. Regarding EIR, the following recommendations are made:

1. The device name, or a portion thereof, is a required element in the EIR data
2. As much of the EIR data as possible, while still maintaining a useful device name, should be devoted to supported service UUIDS
3. The inquiring device should use the EIR data to categorize and/or filter the list of responding devices when presenting a list of responding devices to the user interface. Note, however, that EIR responses are not guaranteed – so implementers should still allow users to find “all” devices in the area.

2.2 CONFIGURATIONS AND ROLES

For the purposes of this whitepaper it is helpful to distinguish the two device roles in discovery:

- **Local device (initiator):** An initiator is the device that initiates the service discovery procedure. An **initiator** must contain at least the client portion of the Bluetooth SDP architecture as defined in the Core System Package [Host volume], Part B **[7]**. An **initiator** contains the service discovery application (SrvDscApp) used to initiate discoveries and display the results of these discoveries.

- **Remote Device(s) (responder(s))**: A **responder** is any device that participates in the service discovery process by responding to the device and service inquiries generated by an **initiator**. A **responder** must contain at least the *server* portion of the Bluetooth SDP architecture as defined in the Core System Package [Host volume], Part B **[7]**. A **responder** contains the service records database used by the server portion of SDP to create responses to service discovery requests.

The Initiator or Responder role assigned to a device is neither permanent nor exclusive. A Responder may implement a SrvDscApp as well as an SDP client, and an Initiator may also have an SDP server. In other words, the assignment of devices to the above roles is relative to which device initiates the SDP transaction. Thus, a device could be an Initiator for a particular SDP transaction, while at the very same time be a Responder for another SDP transaction.

With respect to service discovery, a device without a UI for accepting user input and returning the results of service searches is, by definition, not considered an Initiator. Nevertheless, even if such a device is not considered an Initiator, the procedures presented in the following sections can still apply if applications running in such a device need to execute a service discovery transaction.

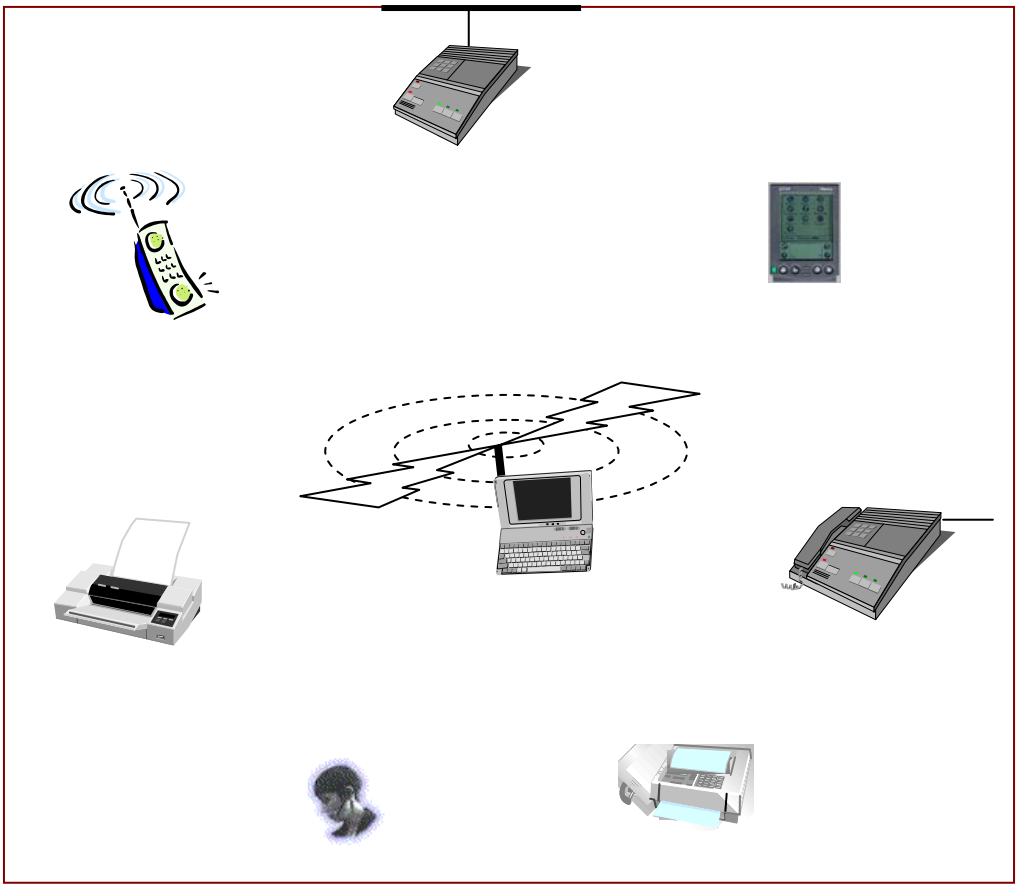

Figure 2.2: A typical service discovery scenario with the PC searching for services

2.3 USER REQUIREMENT AND SCENARIOS

The scenarios covered by this whitepaper are:

- Search for services by a specified service identifier
- Service browsing

The first case represents searching for known and specific services, as part of the user question “Is service A, or is service A with characteristics B and C, available?” The second case represents a general service search that is a response to the user question “What services are available?”

This whitepaper assumes that an initiator implements a user-level application, the SrvDscApp, that interfaces with the SDP protocol, which is used for locating services. The service discovery application differs from a profile’s use of SDP as the SrvDscApp obtains information a number of devices that will be helpful to the end-user in selecting a specific device and service A profile only needs to request the information required to connect with a particular profile on a specific device.

2.4 DISCOVERY FUNDAMENTALS

Before SDP can be used, the two devices must be connected using the establishment procedures described in the Generic Access Profile (GAP). The basic elements of forming a connection are:

- At least one device (Dev B) is performing inquiry scan
- One device (Dev A) performs an inquiry
- The other device (Dev B) responds to the inquiry

- Dev A pages Dev B and Dev B responds
- A baseband (ACL) connection is established
- Optionally one or both devices may request the user-friendly device name from the other
- An SDP search is started, typically by Dev A, but Dev B can also initiate SDP exchanges targeting Dev A
- The devices may then disconnect, or remain connected and start a profile connection, this is up to the particular implementation

Since the SDP operations require a connection, it is strongly recommended that security mode 3 should not be used. Mode 3 requires pairing/authentication before an ACL connection is established. This means that devices must be paired before SDP can be done. This typically requires user intervention, which is annoying, and results in a paired state with a device that may turn out to be of no interest.

If the two devices are based on Bluetooth V2.1 + EDR (or later) Core specifications, the inquiry process may have caused the transfer of a some portion of the device name and other information, including the UUIDs for supported services, from the Responder to the Initiator. This information may be adequate for the user to make their device and service selections. Note: SDP should still be used for any subsequent user-initiated profile connections and profiles should use SDP to query versioning and supported-features information from the applicable SDP record on the Responder.

SDP has no piconet role dependency. Service discovery may be initiated by either a master or a slave device at any time.

Service discovery applications as recommended in this whitepaper do not require the use of authentication and/or encryption. Any security restrictions for SDP transactions are dictated by the security restrictions already in place (if any) on the Bluetooth link. If a specific service/profile requires security then mode 2 is suggested for connections that do not support Secure Simple Pairing, as this establishes security when the profile connection is established.

3 User Interface Aspects

3.1 PAIRING

No requirements regarding pairing are recommended by this whitepaper. Pairing may, or may not, be performed as part of the service discovery process. Whenever an Initiator performs service discovery against not-previously-connected Responder(s), the SrvDscApp may choose to allow pairing prior to connection, or to by-pass any devices that may require pairing first. As previously mentioned, in devices that do not support secure simple pairing, security mode 3 is strongly discouraged, but not prohibited.

The SrvDscApp may have some control over when the Initiator is able to enter the inquiry and/or page states. A Responder, with services to advertise via SDP, shall be able to enter the inquiry scan and/or page scan states. For more information about the inquiry and page related states see Section 8.

Since the SrvDscApp may also perform service inquiries against already connected Responders, it is not necessary for an Initiator to be the master of a connection with a Responder. Similarly, a Responder may not always be the slave of a connection with an Initiator.

4 Application Layer

4.1 THE SERVICE DISCOVERY APPLICATION

In this subsection, the operational framework of the SrvDscApp is presented. Figure 4.1 shows two possibilities for a SrvDscApp.

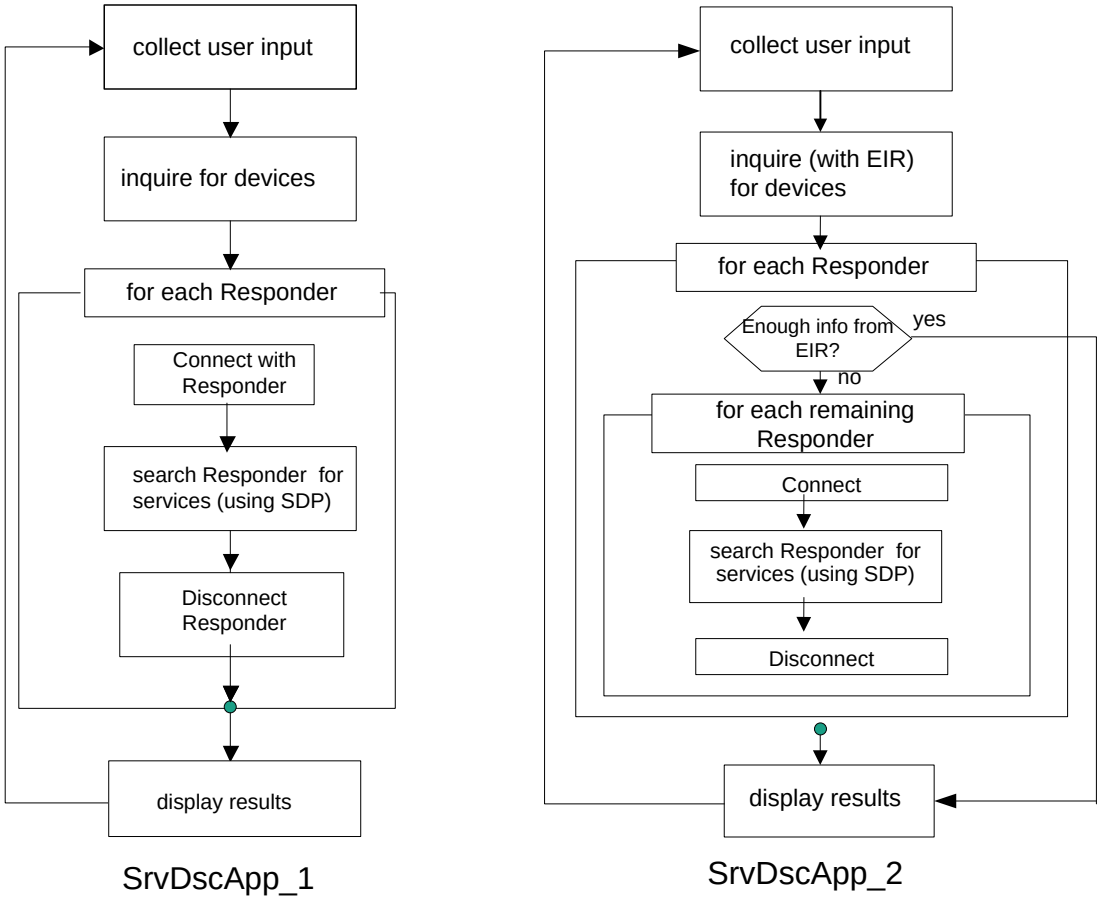

Figure 4.1 : Two possible SrvDscApps

Two SrvDscApp possibilities are shown in Figure 4.1. The first describes an application without EIR, the second, with EIR. These diagrams illustrate a typical implementation, but are not meant to suggest that these are the only possible implementations.

In both cases, the user initiates the action and possibly supplies filtering criteria that can be used to limit the number of devices displayed once the search has completed. The user could, for example, request to see only devices that can stream high quality audio (A2DP).

The application then uses a device inquiry to locate devices that are in-range and discoverable. In case 1, there is no EIR information available, so the application connects to each discovered device and uses SDP to search for services. The search can be for a specific service, or set of services, or a general inventory of everything that the device supports.

In case 2, when both devices support EIR, some time may be saved as the application code can examine the EIR response to determine if the user criteria are met. Having EIR does not eliminate the need for an SDP search to obtain further information about a specific profile, such as its RFCOMM port number, or supported features bits. However, if the goal is to display a list of suitable devices for the user, EIR can reduce the number of connections and SDP searches that are needed for that task.

When the SrvDscApp pages devices to conduct SDP (typically the devices for which the EIR has not provided enough service-related information) then it should arrange for the Initiator to attempt to use either the "no security" option of "JustWorks" pairing when connecting to a v2.1+EDR or later device, or no pairing if connecting to an earlier device. Note that for v2.1+EDR devices, the "no security" configuration is allowed only for the SDP channel as described in GAP[8]. For the Responders that trigger user-involved pairing (legacy pairing, Passkey Entry, or Numeric Comparison) the SrvDscApp may defer SDP queries until after performing inquiries for further Responders. This allows the SrvDscApp to continuing searching for devices that support the requested service, without undue involvement from the user.

To perform its task, SrvDscApp must know the BD_ADDR of the devices in the vicinity of theInitiator, no matter whether these devices have been located via a Bluetooth inquiry process or are already connected to the Initiator. Thus an Initiator will need to maintain the list of devices in the vicinity of the Initiator and make this list available to the SrvDscApp.

4.2 DISCOVERY WHEN EIR IS SUPPORTED

EIR (available in V2.1 + EDR and later core specifications) provides a possible mechanism for service-related information to be sent in response to inquiry. The minimum required information is the user name for the Responder. Other information is optional but may be adequate for the automated selection or user selection of the devices to further interrogate for service information.

It is recommended that service-related information be included in EIR responses. The information can be used by the Initiator to reduce the number of devices to page and perform full SDP – this implies that association of devices can be performed more quickly with EIR. For example if a user wishes to find a device that can render stereo audio the Initiator should focus on devices that have sent EIR responses indicating the A2DP Snk service class. If the user of the Initiator chooses to use that Responder the local A2DP profile will comply with the A2DP profile requirements, thereby performing full SDP searches on the A2DP Snk service record. The benefit of EIR here is that the user finds a suitable stereo rendering device more quickly than if the Initiator paged all Responders to perform SDP (including those devices that do not have the A2DP service)

4.3 SERVICE PRIMITIVES ABSTRACTIONS

This section briefly describes the functionality of a SrvDscApp and is presented in the form of service primitive abstractions that provide a formal framework for describing user expectations. It is assumed that the underlying Bluetooth stack can meet the objectives of these service primitive abstractions directly or indirectly.[2] The exact syntax and semantics of the service primitive abstractions (or simply "service primitives") may be platform-dependent (e.g. an operating system, a hardware platform, like a PDA, a notebook computer, a cellular phone, etc.) and are beyond the scope of this whitepaper. However, the functionality of these primitives is recommended for the SrvDscApp to accomplish its task.

Table 4.1 contains a minimum set of enabling service primitives to support a SrvDscApp. Low-level primitives like **openSearch**(.) or **closeSearch**(.) are not shown and are assumed to be part of the implementation of the primitives shown whenever necessary. Different implementations of the Bluetooth stack will, at a minimum, need to enable the functions provided by these service primitives. For example, the **serviceSearch**(.)primitive

2 These service primitive abstractions do not represent programming interfaces, even though they may be related to them. The word 'directly' indicates the possibility that the described function may be implemented with a single call to the underlying Bluetooth stack. The word 'indirectly' indicates the possibility that the described function may be implemented by combining the results from multiple calls to the underlying Bluetooth stack.

permits multiple identical operations to be handled concurrently. A stack implementation that requires an application to accomplish this function by iterating through the multiple identical operations one-at-a-time will be considered as enabling the function of this service primitive.[3] The service primitives shown next relate only to service primitives whose invocation result or relate to an over-the-air data exchange using SDP and/or EIR. Additional service primitives can be envisioned relating to purely local operations like *service registration*, but these primitives are outside the scope of this whitepaper

Service primitive abstraction	Resulted action
serviceBrowse (LIST(*Responder)* LIST(*ResponderRelation*) LIST(*browseGroup*) *getResponderName stopRule*)	Search for services (service browsing) that belong to the list of browseGroup services in the devices in the list of Responders; the search may be further qualified with a list of ResponderRelation parameters, whereby a user specifies the trust and connection relation of the devices to be searched; e.g. search only the devices that are in the Rem Dev list for which there is a trust relation already established; when the getResponderName parameter is set to “yes,” the names of the devices supporting the requested services are also returned; the search continues until the stopping rule stopRule is satisfied
serviceSearch (LIST(*Responder)* LIST(*ResponderRelation*) LIST(*searchPattern*, *attributeList*) *getResponderName stopRule*)	determines whether the devices listed in the list of Responders support services in the requested list of services; each service in the list must have a service search pattern that is a superset of the *searchPattern*; for each search pattern the values of the attributes contained in the corresponding *attributeList* are also retrieved; the search may be further qualified with a list of *ResponderRelation* parameters, whereby a user specifies the trust and connection relation of the devices to be searched (e.g. search only the devices that are in the *Responder* list for which there is a trust relation already established); when the *getResponderName* parameter is set to “yes,” the names of the devices supporting the requested services are also returned; the search continues until the stopping rule *stopRule* is satisfied
enumerateResponder (*stopRule*)	a search for Responders in the vicinity of a Initiator; the search continues until the stopping rule *stopRule* is satisfied
terminatePrimitive (primitiveHandle returnResults)	a termination of the actions executed as a result of invoking the services primitive identified by the *primitiveHandle*;* optionally, this service primitive may return any partially accumulated results related to the terminated service primitive

Table 4.1: Service primitives in support of SrvDscApp

* It is assumed that each invocation of a service primitive can be identified by a primitiveHandle, the realization of which is implementation-dependent.

The *stopRule* parameter is used to provide graceful termination of a service search. It could represent the number of search items found, or the duration of search, or both. A Bluetooth stack implementation may not

3 Even though the service primitives presented in this whitepaper are assumed to act upon a local device for accessing physically remote devices, they are general enough to apply in cases where the ‘remote device’ characterization is only a logical concept; i.e. inquired service records and service providers are located within the same device that invokes these primitives. Such a system is outside the scope of this whitepaper.

expose this parameter, in which case it should provide guarantees that all searches terminate within a reasonable amount of time (eg, 120 seconds).

The **enumerateResponder**(.) service primitive is directly related to the inquiry mode of operation for the baseband. It also relates to the collection of Responder that an Initiator is currently connected with. This service is exported to the SrvDscApp via the BT_module_Ctrl; see Figure 2.1. The interface between the BT_module_Ctrl and the baseband is for activating Bluetooth inquiries and collecting the results of these inquiries. The interface between the BT_module_Ctrl and L2CAP is for keeping track of the Responders that are currently connected to the Initiator.

The result of the **enumerateResponder**(.) service primitive can be used with the **serviceSearch**(.) to search for desired services in the devices found. Once again, based on the implementation of the Bluetooth stack, this service primitive may not be provided explicitly, but its service may be provided within other service primitives; e.g. the **serviceSearch**(.).

Missing primitive parameters should be interpreted (whenever appropriate) as a general service search on the remaining parameters. For example, if the LIST(*Responder*) parameter is missing from the **serviceSearch**(.), it means that the search will need to be performed against any device found in the vicinity of a Initiator. In this case, the first two service primitives may be combined to a single one.

The above service primitives return the requested information (which has been obtained with SDP and/or EIR implementation-specific) via implementation-specific methods in the Initiator.

4.4 MESSAGE SEQUENCE CHARTS (MSCS)

The SrvDscApp has three distinct Bluetooth procedures: Device inquiry, name discovery, and service discovery. Note that each one of these procedures does not preclude any other; e.g. to connect to a Responder, an Initiator may have to first discover it, and it may also ask for its name. The first two procedures (i.e., device inquiry and name discovery) are provided in GAP contained in Core System Package [Host volume] and the Message Sequence Charts contained in Part F of the Core System Package [Controller volume]. Note that GAP also defines a procedure (device discovery) that combines the device inquiry and name discovery procedures.

Figure 4.2 summarizes the key message exchange ‘phases’ encountered during the execution of a SrvDscApp. Not all procedures are required to be used each time, and not all devices need to go through these procedures each time. For example, if authentication is not required, the authentication phase in the figure will not be executed. If the SrvDscApp needs to inquire for services on a connected Responder, inquiries and pages are not necessary. In the figure, the conditions under which particular phases are executed or not are also provided.

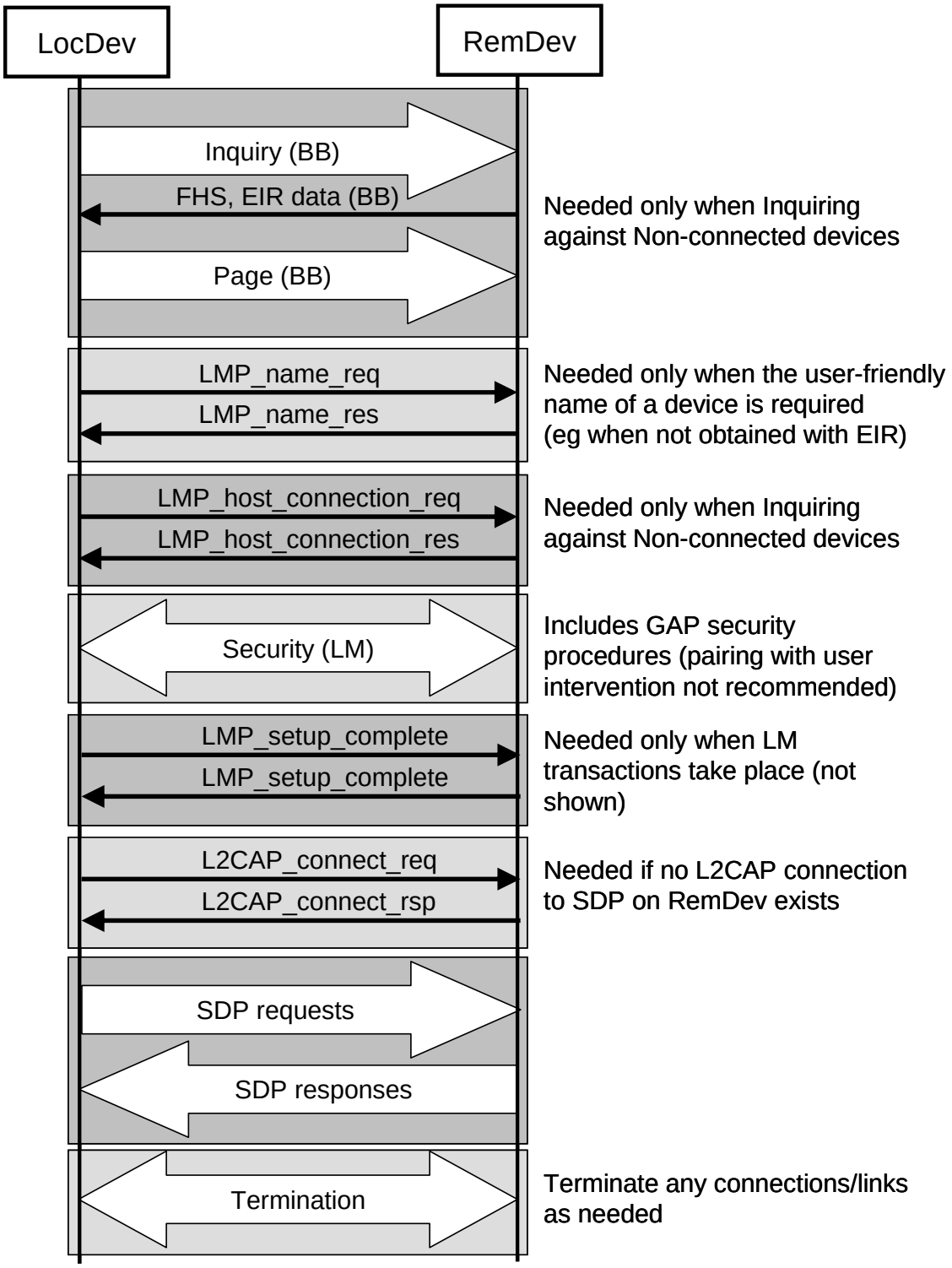

Figure 4.2: Bluetooth processes in support of a SrvDscApp

5 Service Discovery Protocol

The service discovery application uses SDP as a means of informing the user of an Initiator about the services that are available from a Responder. Bluetooth-aware applications running in a local device can also use the procedures described in this and the following sections to retrieve the information required to access a service in a Responder.

SDP transactions can be performed when the Initiator is either a master or slave, so a role switch is never required in order to perform SDP.

Table 5.1 shows the SDP feature requirements in an Initiator and in a Responder.

	SDP feature	Support in Initiator	Support in Responder
1.	SDP client	M	O
2.	SDP server	O	M

Table 5.1: SDP feature requirements

Table 5.2 shows the SDP PDUs can be exchanged between devices providing SrvDscApps.

SDP PDUs	Ability to Send		Ability to Receive	
	Initiator	Responder	Initiator	Responder
SDP_ErrorResponse	C1	M	M	C1
SDP_ServiceSearchRequest	C2	C1	C1	M
SDP_ServiceSearchResponse	C1	M	M	C1
SDP_ServiceAttributeRequest	M	C1	C1	M
SDP_ServiceAttributeResponse	C2	M	M	C1
SDP_ServiceSearchAttributeRequest	C3	C1	C1	M
SDP_ServiceSearchAttributeResponse	C1	M	M	C1
Comments: [C1]: With regard to this white paper, these PDU transmissions will not occur. Nevertheless, since a device could act as an Initiator on some occasions and as a Responder on others, these PDU transmissions may still take place between these devices. [C2],[C3]: At least one of C2 or C3 is needed to implement an Initiator SrvDscApp				

Table 5.2: SDP PDUs as used by a SrvDscApp

5.1 AN SDP PDU EXCHANGE EXAMPLE

Figure 5.1 shows two examples of SDP PDU exchanges. In particular, it shows PDU exchange sequences for the inquiry and retrieval of any information pertinent to a particular Bluetooth profile.

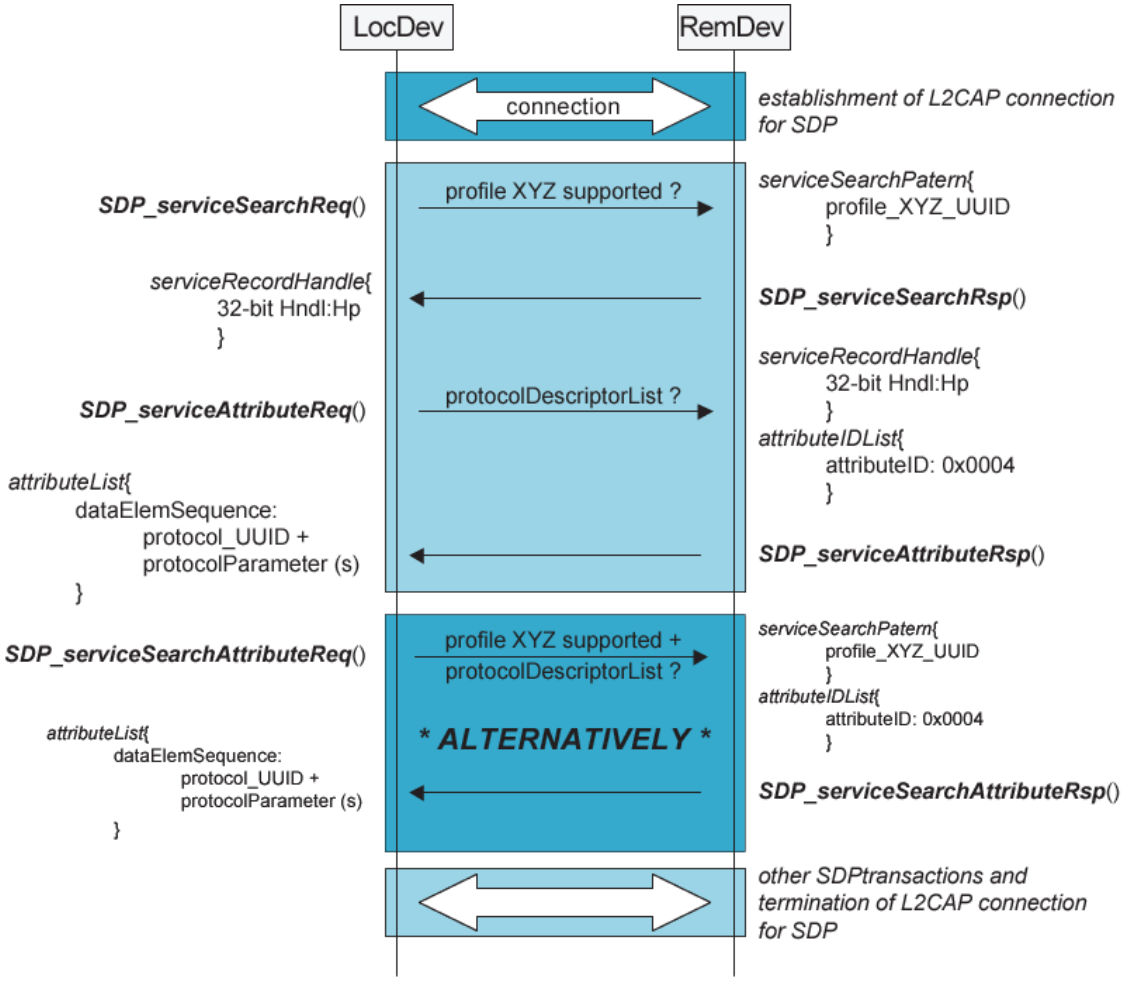

Figure 5.1: SDP PDU exchange examples for retrieving protocolDescriptorLists

For each PDU sent, the figure shows which device sends it (shown on the starting side of an arrow) and any relative information that this PDU carries (shown on the ending side of an arrow). Note that the Initiator sends request PDUs, while the Responder sends back response PDUs.

Two alternatives are shown utilizing different SDP PDUs to ultimately retrieve the same information – the *protocolDescriptorList* attribute from devices that support a specific Bluetooth profile. With the first alternative, the desired information is derived in two steps.

- The Initiator sends an SDP_serviceSearchReq PDU which contains a service search pattern composed of a list of UUIDs to search; (see Bluetooth Assigned Numbers). The desired profile (profile 'XYZ') is identified by its UUID, denoted in the figure as 'profile_XYZ_UUID.' In its response PDU, the SDP server returns one or more 32-bit service record handles whose corresponding service records contain the 'profile_XYZ_UUID' UUID. In the figure, only one such handle is shown, denoted as 'prHndl'.
- The Initiator then enters prHndl in an SDP_serviceAttributeReq PDU together with one or more attribute IDs. In this example, the attribute of interest is the protocolDescriptorList, whose SDP attribute ID is 0x0004. The SDP server then, in its response, returns the requested protocol list.

In the event that no service record containing the desired service search pattern is found in the SDP server, the *SDP_serviceSearchResp* PDU will contain an empty *serviceRecordHandleList* and a *totalServiceRecordCount* parameter set to its minimum value; see the SDP section in the Bluetooth Core specification.

If the desired attributes do not exist in the SDP server, the *SDP_serviceAttributeResp* PDU, as required by the SDP, will contain an empty *attributeList* and an *attributeListByteCount* parameter set to its minimum value, see the SDP section in the Bluetooth Core specification.

With the second alternative, the desired attributes are retrieved in one step:

- The Initiator sends an *SDP_serviceSearchAttributeReq* PDU where the desired profile is included (service search pattern: profile_XYZ_UUID) and the desired attribute(s) is provided (attribute ID: 0x0004). In its response the SDP server will provide the requested attribute(s) from the service record(s) that matches the service search pattern.

In case no service record containing the desired service search pattern and/or the desired attribute(s) is found in the SDP server, the *SDP_serviceSearchAttributeResp* PDU will contain an empty *attributeLists* and an *attributeListsByteCount* parameter set to its minimum value, see the SDP section in the Bluetooth Core specification.

While, in the example in Figure 5.1, only a few service attributes are shown retrieved by the SDP client, additional information could and should be requested; reading the versioning information from an SDP record is particularly recommended. In cases where service information is to be cached for future use, an SDP client should also request any pertinent information that can aid in assessing whether cached information has become stale. The service attributes *serviceDatabaseState*, *serviceRecordState*, and*serviceInfoTimeToLive* have been defined for this purpose in SDP: see the SDP section in the Bluetooth Core specification

6 L2CAP

6.1 CHANNEL TYPES

Only connection-oriented channels are used for SDP transactions.

6.2 SIGNALING

For the purpose of retrieving device and service related information, the Initiator starts the inquiry process and L2CAP connection requests.

SDP sessions are always started by the SDP client. If an L2CAP connection does not already exist between the two devices, it is the client side that will create it. The Responder (SDP server) processes incoming requests for connection establishment.

Under normal operation, an SDP server does not terminate the L2CAP connection for SDP. However, exceptional cases, such as when a Responder is shut down during the execution on an SDP transaction, cannot be excluded. In such a case, prior to the final power-off, the Responder may terminate all its active L2CAP connections by sending connection termination PDUs. In any case, a Responder shall process an incoming request for connection termination.

A PSM value of 0x0001 in the Connection Request packet is used, to create an L2CAP connection with the SDP server.

6.3 CONFIGURATION OPTIONS

This section describes the usage of configuration options in L2CAP.

6.3.1 MAXIMUM TRANSMISSION UNIT (MTU)

For efficient use of the communication resources, the MTU should be as large as possible, while respecting any physical constraints imposed by the devices involved. This includes honoring any active QoS contracts with other devices and/or applications. It is expected that during the lifetime of an L2CAP connection for SDP transactions (also referred to as the ‘SDP session’, see Section 6.4) between two devices, any one of these devices may become engaged in an L2CAP connection with another device and/or application. If this new connection has ‘non-default’ QoS requirements, the MTU for the ongoing SDP session can be re-negotiated to accommodate the QoS constraints of the new L2CAP connection.

6.3.2 FLUSH TIME-OUT

It is recommended that SDP transactions should be carried over an L2CAP reliable channel: the flush time-out value (see L2CAP in the Bluetooth Core Specification [5]) should be set to its default value 0xFFFF. However, the flush timeout may be constrained by other L2CAP channels running over the same ACL link. The Packet-Boundary flag feature introduced in Bluetooth Core v2.1 + EDR and later may be used to prevent packets containing SDP from being automatically flushed.

6.3.3 QUALITY OF SERVICE

It is assumed that SDP traffic will be treated as best-effort service traffic and thus no special QoS requests need to be utilized.

6.3.4 FLOW AND ERROR CONTROL

The use of this option is recommended to provide the segmentation and reassembly feature: this allows SDP transactions to be efficiently interleaved with profile traffic. It is recommended to negotiate Enhanced Retransmission mode for the L2CAP channel carrying SDP.

6.4 SDP TRANSACTIONS AND L2CAP CONNECTION LIFETIME

While, in general, SDP transactions comprise a sequence of service request- and-response PDU exchanges, SDP itself constitutes a connectionless datagram service in that no SDP-level connections are formed prior to any SDP PDU exchange. The SDP layer delegates the creation of connections on its behalf to the L2CAP layer. It is thus the responsibility of the SDP layer to request the L2CAP layer to terminate these connections on its behalf as well.

Since SDP servers are considered stateless, terminating an L2CAP connection after a SDP PDU is sent (as a true connectionless service may imply) will be detrimental to the SDP transaction. Moreover, significant performance penalty will have to be paid if a new L2CAP connection is to be created for each SDP PDU transmission. Thus, L2CAP connections for SDP transactions should be kept in place for more than the transmission of a single SDP PDU.

An SDP session represents the time interval that the SDP client and server have the same L2CAP connection continuously present. A minimal SDP transaction consists of a single SDP request PDU sent by an SDP client and the corresponding SDP response PDU from the SDP server. With respect to the overall discovery process, under normal operational conditions the minimum duration of an SDP session should be the duration of a minimal SDP transaction.

An SDP session may last less than the minimum required in the event of unrecoverable (processing or link) errors in layers below SDP in the Initiator or Responder, or in the SDP layer or service records database in the Responder. An SDP session may also be interrupted by user intervention, which may terminate the SDP session prior to the completion of an SDP transaction.

The above minimum duration of an SDP session guarantees smooth execution of the SDP transactions. For improved performance, implementers may allow SDP sessions to last longer than the minimum duration of an SDP session. As a general implementation guideline, an SDP session should be maintained for as long as there is a need to interact with a specific device. Since the latter time is in general unpredictable, SDP implementations may maintain timers used to time periods of SDP transaction inactivity over a specific SDP session. Another benefit to keeping the SDP L2CAP channel open is that it keeps the underlying ACL connection open. Many stack implementations will close the ACL connection when the last L2CAP channel is closed. Since the goal of SDP is to locate a service and connect to it, this allows time for the Initiator to use the Responder's SDP information to initiate a profile connection with the desired service without the need to re-establish an ACL connection.

SDP implementations may also rely on explicit input received from a higher layer (probably initiated from the SrvDscApp itself) to open and close an SDP session with a particular device using low-level primitives; e.g. openSearch(.) and closeSearch(.). Finally, an implementation may permit users to interrupt an SDP session at any time, see the terminatePrimitive(.) service primitive in 4.2.

Normally, an SDP session is not terminated by the Responder. Yet such an event can occur, either by the Responder gracefully terminating the SDP session using the L2CAP Disconnect Request PDU, or abnormally terminating the SDP by stopping responses to SDP requests and/or L2CAP signaling commands. Such an event may be an indication of an exceptional condition that SDP client/server implementers should consider addressing for the smooth execution of this application. If a termination event is generated by a Responder, an SDP client should clear any stored information from this Responder. Such an exceptional event may mean that the SDP server has (or is about to) shut-down, in which case any service information retrieved from this server should be considered stale.

7 Generic Access Profile and EIR

For the next four subsections, it is assumed that an Initiator is performing service searches with unconnected Responders. It thus needs to inquire for and page these Responders. None of these subsections apply to an Initiator performing service searches with connected Responders.

7.1 DISCOVERY

Whenever instructed by the SrvDscApp, the Responder should enter the 'discoverable' state defined in the Generic Access Profile[1]. The time lag before entering this state and completion of the discovery may be constrained by existing connections. For example an active SCO connection severely limits the number of baseband slots available for inquiry and paging exchanges. When inquiry is invoked in a Initiator, the general inquiry procedure using a GIAC should be used as described in the GAP_profile:[3].

This whitepaper does not exclude performing inquiries according to the limited inquiry procedure described in GAP.

The user of the SrvDscApp, or the SrvDscApp itself, may set the criteria for the duration of an inquiry, see *stopRule* service primitive parameter in 4.2.

7.2 DISCOVERABLE MODE

To be discovered by an inquiry, a Responder should enter the inquiry scan state using the GIAC; see general discoverable mode in GAP. A DIAC can be used instead of a GIAC, though the use of DIACs are of limited benefit when doing service discovery..

The inquiry scan parameters are device-dependent and outside the scope of this whitepaper.

When EIR is supported, Responders should include as many 16-bit SDP service identifiers (UUIDs as possible in their EIR response.

7.3 LINK ESTABLISHMENT

Whenever the SrvDscApp needs to connect to a specific Responder for performing SDP, the Initiator follows the Link Establishment procedure in the Geriatric Access Profile The time lag before entering this state and the amount of time available for the link establishment may be constrained existing connections.

7.4 CONNECTABLE MODE

Devices that have responded to an inquiry will need to enter page scan state in order to allow the discovering device to connect, request device name, and perform SDP. Therefore, a device that is Discoverable should also be Connectable. As an example, if the Responder has sent an EIR response it should enter Connectable state because the Initiator will likely connect to perform full SDP searches using one or more of the service UUIDs included in the EIR response.

8 References

[1] Core specification, Vol 2, Part B: Baseband Specification v1.2 or later

[2] Bluetooth Assigned Numbers
https://www.bluetooth.org/Technical/AssignedNumbers/home.htm

[3] Core specification, Vol. 3, Part C: Generic Access Profile v1.2 or later

[4] Core specification, Vol 2, Part C: Link Manager Protocol

[5] Core specification, Vol 3, Part A : Logical Link Control and Adaptation Protocol in the Bluetooth Core Specification

[6] Core specification, Vol 2, Part F: Message Sequence Charts between Host–Host Controller/Link Manager

[7] Core specification, Vol 3, Part B: Service Discovery Protocol

[8] Core specification, Vol 2, Part B: Baseband Specification v2.1 or later