Palm, Inc.

Palm OS® Bluetooth Support

Table of Contents

1 Introduction

Bluetooth is a new and exciting technology that has fired the imagination of the entire technical community. The standard is still evolving, and volume shipments of products are not anticipated until Q1/Q2 of 2001. In technical terms, Bluetooth is a new low-cost short-range Frequency Hopping Spread Spectrum (FHSS) radio technology in the 2.4 GHz ISM band. This document describes the integration of Bluetooth into Palm products.

Bluetooth is an industry standard being developed by the Bluetooth Special Interest Group (SIG), with includes the following founding (promoter) companies: 3Com, Ericsson, IBM, Intel, Lucent, Microsoft, Motorola, Nokia and Toshiba. Currently, there are over 2000 additional adopter/associate member companies. Palm™, Inc. is an associate member of the SIG. Additional information about Bluetooth can be found at http://www.bluetooth.com.

The original Bluetooth specification, version 1.0B, was released December 1999. As of this writing, it is anticipated that version 1.1 will be released by the SIG by the end of the 2000, Version 1.1 addresses issues in the 1.0B specification by including errata. A final draft version of the 1.1 specification was published in October of 2000. Palm intends to release a product compliant with version 1.1 of the SIG specification.

This document provides information with respect to work in progress on Bluetooth enabled products within Palm. Please note that the information contained herein is subject to change prior to product release. This document will be updated as new information is received. The latest version of this document can be found in the Developer area of the Provider Pavilion on http://www.palmos.com.

2 Palm Objectives

Bluetooth is all about connectivity and interoperability between Bluetooth-enabled devices. To ensure that devices can talk to each other, the SIG has created *profiles*, or usage scenarios. These profiles provide a basis for the development of a common core set of functions that the first Bluetooth-enabled devices will support. The SIG requires that all devices must be qualified using a standardized set of test procedures to ensure interoperability and conformance to the Bluetooth specification. Palm intends to provide a compelling, intuitive, and easy to use implementation of Bluetooth, optimized for the Palm OS™.

Palm's ultimate goal is to integrate Bluetooth radio technology into Palm handheld devices as soon as the technology matures enough for it to do so. Until then, Palm will produce Bluetooth-enabled peripheral devices.

Palm OS® support for Bluetooth provides Bluetooth connectivity to Palm OS features, such as HotSync® operations and PPP network access. Additionally, a comprehensive Application Programmer Interface (API) allows development of Bluetooth-enabled applications. Key elements of the software system design include:

- Provide a consistent UI and API across all Palm OS devices.
- Release a Bluetooth product that conforms to version 1.1 of the Bluetooth Special Interest Group (SIG) specification.
- Develop and integrate an API to allow developers to write Bluetooth-enabled applications
- Provide a software architecture that allows for the integration of new radio hardware easily through the Host Controller Interface (HCI), a hardware abstraction layer defined in the Bluetooth specification.
- Provide a User Interface (UI) for device discovery and connection.
- Provide a UI for Personal Identification Number (PIN) entry.
- Provide appropriate modifications to the Palm Connection Panel to support Bluetooth.
- Allow emulation of serial ports using the Virtual Serial Port driver.
- Support desktop-based HotSync operations with a select number of Bluetooth PC Cards.
- Support HotSync operations over the Internet Protocol (IP) using cell phones and access points.
- Maintain compatibility with Wireless Internet Kit and other popular third party Internet applications.

3 Functional Features

3.1 BLUETOOTH FEATURES DEFINED

Within the Bluetooth SIG specification, certain features are mandatory and others are optional. This section provides an overview of the Bluetooth specification, and defines which of the optional features that Palm has elected to support.

3.1.1 Bluetooth Protocol Stack

Bluetooth allows for developing interactive services and applications over interoperable radio modules and data communication protocols. The ultimate objective of the Bluetooth protocol stack is to allow applications to interoperate with each other.

Figure 1 shows the complete Bluetooth protocol stack as identified in the specification. Interoperable applications supporting Bluetooth usage models are built on top of this protocol stack. Not all applications make use of all protocols shown in Figure 1. Instead, applications run over one or more vertical slices from this protocol stack.

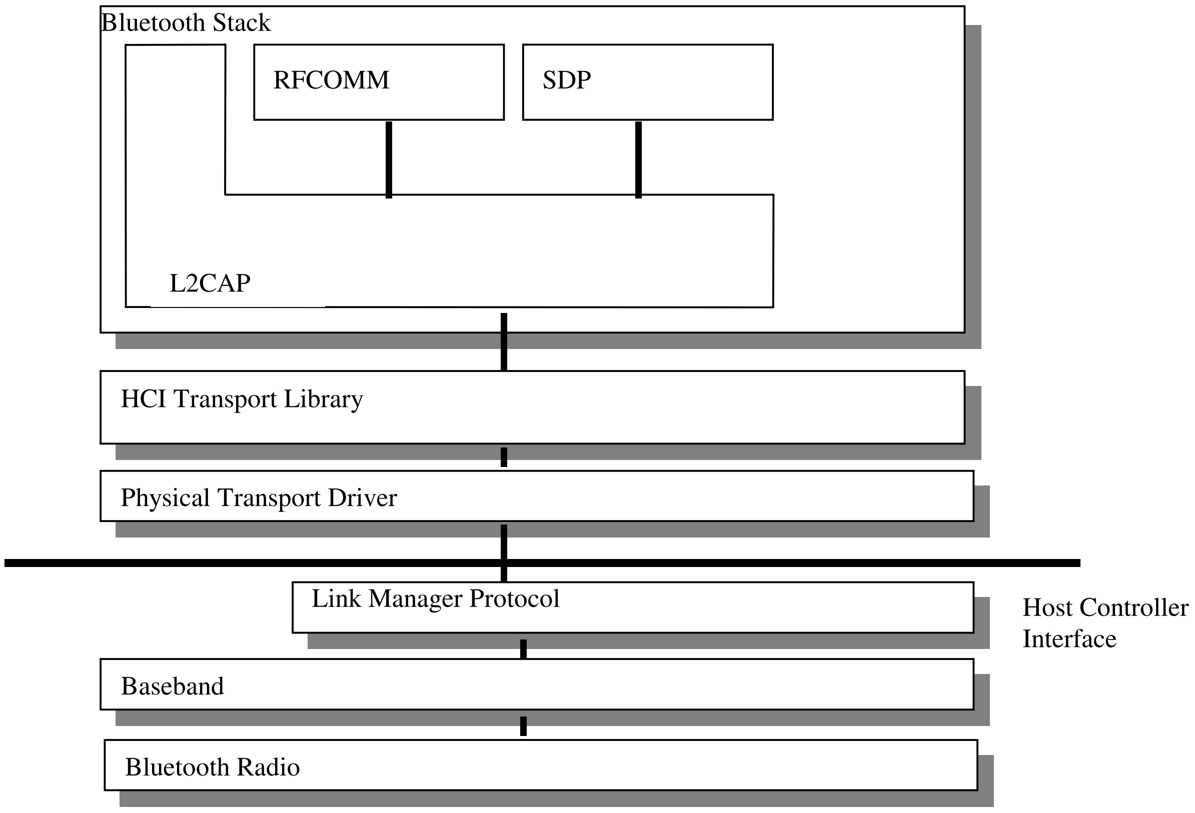

Figure 1 – Bluetooth Stack

The Bluetooth protocol stack can be divided into four layers according to their purpose, as shown in the following table:

Protocol Layer	Protocols in the Stack
Bluetooth Core Protocols	Baseband, LMP, L2CAP, SDP
Cable Replacement Protocol	RFCOMM
Telephony Control Protocols	TCS Binary, AT-commands
Adopted Protocols	PPP, UDP/TCP/IP, OBEX, WAP, vCard, vCal, IrMC , WAE

Table 1 – Bluetooth Supported Protocols and Layers

In addition to the above protocol layers, the specification also defines a Host Controller Interface (HCI), which provides a command interface to the baseband controller and link manager and also provides access to hardware status and control registers.

3.1.1.1 Bluetooth Core Protocols

3.1.1.1.1 Baseband

The baseband layer and the link control layer enable the physical Radio Frequency (RF) link between Bluetooth units. When two or more units are linked in this manner, it is called a *piconet*. The Bluetooth RF system is a Frequency Hopping Spread Spectrum (FHSS) system in which packets are transmitted in defined time slots on defined frequencies. The Baseband Layer uses inquiry (remote device discovery) and paging (establishing connections to remote devices) procedures to synchronize the transmission hopping frequency and the clock of different Bluetooth devices. Two kinds of physical links can be formed with their corresponding baseband packets: Synchronous Connection-Oriented (SCO) and Asynchronous Connectionless (ACL). These can be transmitted in a multiplexing manner on the same RF link. ACL packets are used for data only, while the SCO packet can contain audio only or a combination of audio and data. **Table 2** defines which physical links are supported.

Link Type	Description	Palm Feature
Synchronous Connection-Oriented (SCO) link	The SCO link is a symmetric, point-to-point link between the master and a specific slave. The SCO link reserves slots and can therefore be considered as a circuit-switched connection between the master and the slave. The SCO link typically supports time-bounded information like voice.	No
Asynchronous Connection-Less (ACL) link	In the slots not reserved for SCO links, the master can exchange packets with any slave on a per-slot basis. The ACL link provides a	Yes

Link Type	Description	Palm Feature
	packet-switched connection between the master and all active slaves participating in the piconet. Both asynchronous and isochronous services are supported.	

Table 2 – Physical Link Types Supported

3.1.1.1.2 Audio

Audio data can be transferred between one or more Bluetooth devices, making various usage models possible. Audio data in SCO packets is routed directly to and from the baseband layer; it does not go through the Link Control and Adaptation Protocol (L2CAP). The audio model is relatively simple within Bluetooth. Any two Bluetooth devices can send and receive audio data between each other just by opening an audio link. SCO data links are not supported.

3.1.1.1.3 Link Manager Protocol

The Link Manager Protocol is responsible for setting up the link between Bluetooth devices. This includes security aspects like authentication and encryption by generating, exchanging, and checking of link and encryption keys. It also includes the control and negotiation of baseband packet sizes. Furthermore, it controls the power modes and duty cycles of the Bluetooth radio device, and the connection states of a Bluetooth unit in a piconet.

3.1.1.1.4 Logical Link Control and Adaptation Protocol

The L2CAP protocol adapts upper layer protocols over the baseband layer. L2CAP provides both connection-oriented and connectionless data services to the upper layer protocols with protocol multiplexing capability, segmentation and reassembly operation, and group abstractions. L2CAP permits higher level protocols and applications to transmit and receive L2CAP data packets up to 64 KB in length. L2CAP is defined only for ACL links and not supported for SCO links, as specified by the Bluetooth Specification.

3.1.1.1.5 Service Discovery Protocol

Service Discovery is a crucial part of the Bluetooth framework. Using the Service Discovery Protocol (SDP), specific information about a remote device, such as available services and the characteristics of these services can be queried.

3.1.1.2 Cable Replacement Protocol

3.1.1.2.1 RFCOMM

RFCOMM is a serial line emulation protocol and is based on the ETSI 07.10 specification. This “cable replacement” protocol emulates RS-232 control and data signals over the Bluetooth baseband layer, providing both of these transport capabilities for upper level services (e.g. OBEX) that typically use serial line as a transport mechanism.

3.1.1.3 Telephony Control Protocol

3.1.1.3.1 Telephony Control – Binary

Telephony Control protocol - Binary (TCS Binary or TCS BIN), a bit-oriented protocol, defines the call control signaling for the establishment of speech and data calls between Bluetooth devices. In addition, it defines mobility management procedures for handling groups of Bluetooth TCS devices. TCS Binary is specified in the Bluetooth Telephony Control protocol Specification Binary, which is based on the ITU-T Recommendation Q.931, applying the symmetrical provisions as stated in Annex D of Q.931

3.1.1.3.2 Telephony Control – AT Commands

The Bluetooth SIG defines a set of AT commands by which a mobile phone and modem can be controlled. The AT commands that Bluetooth uses are based on ITU-T Recommendation V.250 and ETS 300 916 (GSM 07.07). In addition, the commands used for FAX services are also specified.

3.1.1.3.3 Palm OS Telephony Control Support

Table 3 defines which telephony control protocols are supported.

Telephony Control	**Description**	**Palm Feature**
TCS – Binary	Telephony Control Protocol Specification Binary (TCS Binary), a bit-oriented protocol, defines the call control signaling for the establishment of speech and data calls between Bluetooth devices.	No
AT Commands	AT commands are based on ITU-T recommendation V.250 and ETS 300 916.	Yes

Table 3– Telephony Control

3.1.1.4 Adopted Protocols

3.1.1.4.1 PPP

In the Bluetooth specification, PPP runs over RFCOMM to accomplish point-to-point connections. PPP is the IETF Point-to-Point Protocol and PPP-Networking is the means of taking IP packets to/from the PPP layer and placing them onto the LAN.

3.1.1.4.2 TCP/UDP/IP

The TCP, UDP, and IP protocol standards are used for communication over the Internet and are considered the most widely used protocol family in the world. The implementation of these standards in Bluetooth devices allows for communication with any other device connected to the Internet. TCP/IP/PPP is used for the all Internet Bridge usage scenarios in Bluetooth 1.1 and for OBEX in future versions.

3.1.1.4.3 OBEX Protocol

IrOBEX (shortly OBEX) is a session protocol developed by the Infrared Data Association (IrDA) to exchange objects in a simple and spontaneous manner. OBEX, which provides the same basic functionality as HTTP but in a much lighter fashion, uses a client-server model and is independent of the transport mechanism and transport API, provided the transport mechanism is reliable. Along with the protocol itself, which is the "grammar" for OBEX conversations between devices, OBEX also provides a model for representing objects and operations. In addition, the OBEX protocol defines a folder-listing object, which is used to browse the contents of folders on remote device. In the first phase of Bluetooth, RFCOMM is used as sole transport layer for OBEX.

3.1.1.4.4 Content Formats

vCard and vCalendar are open specifications developed by the Versit Consortium and are now controlled by the Internet Mail Consortium. These specifications define the format of electronic business cards and scheduling information, respectively. vCard and vCalendar do not define any transport mechanism, but only the format under which data is transported.

Other content formats transferred by OBEX in Bluetooth are vMessage and vNote. These content formats are also open standards. They are used to exchange messages and notes, respectively. These standards are defined in the IrMC specification, which also defines a format for the log files that are needed when synchronizing data between devices.

3.1.2 Bluetooth Profiles

The Bluetooth Specification defines a set of usage models for Bluetooth radios. Profiles define the protocols and protocol features supporting a particular usage model. In addition to these profiles, there are four general profiles that are widely utilized by these usage model oriented profiles. These are the:

- Generic Access Profile (GAP)
- Serial Port Profile
- Service Discovery Application Profile (SDAP)
- Generic Object Exchange Profile (GOEP)

Figure 2 below illustrates the profiles defined for Bluetooth. Profiles shaded in red are not supported. Note that the profile definitions are hierarchical:

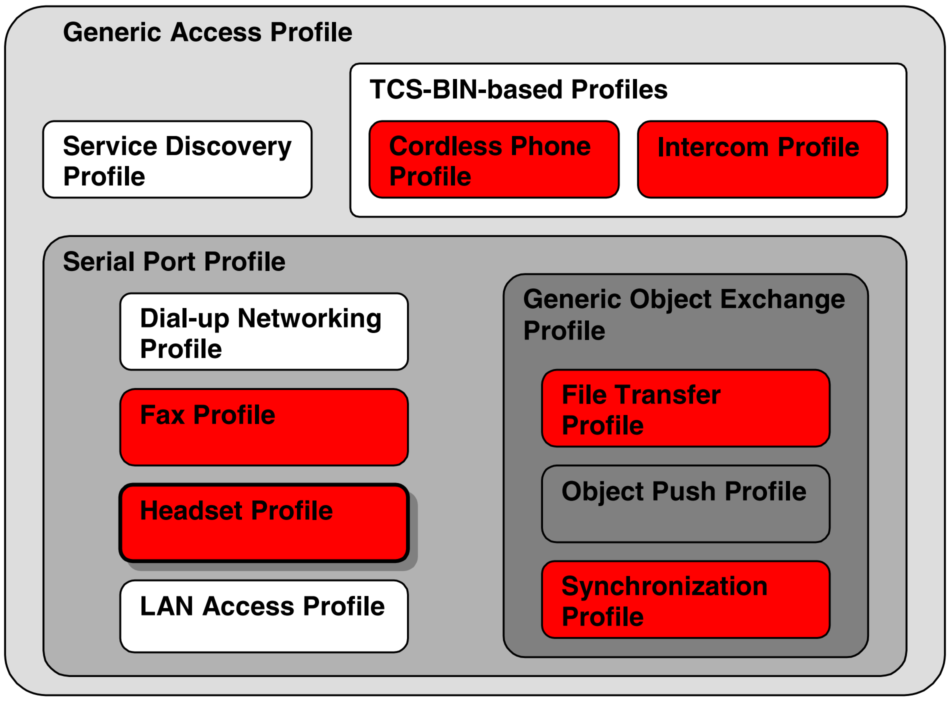

Figure 2 – Bluetooth Profiles

The table below outlines each of the defined Bluetooth profiles and which are supported. Palm does not support the Bluetooth Synchronization profile, but implements HotSync operations over Bluetooth using the Serial Port profile.

Bluetooth Profiles	**Description**	**Palm Feature**
Generic Access	The main purpose of the general access profile is to describe the use of the lower layers of the Bluetooth protocol stack (LC and LMP) and to describe security related alternatives, and higher layers (L2CAP, RFCOMM and OBEX).	Yes
Service Discovery Application	The service discovery profile defines the protocols and procedures that should be used by a service discovery application on a device to locate services in other Bluetooth-enabled devices using the Bluetooth Service Discovery Protocol (SDP).	Yes
Cordless Telephony	The cordless telephony profile defines the protocols and procedures that should be used by devices implementing the use case called "3-in-1 phone."	No
Intercom	The intercom profile defines the protocols and procedures that should be used by devices implementing the intercom part of the usage model called "3-in-1 phone." More popularly, this is often referred to as the "walkie-talkie" usage of Bluetooth.	No
Serial Port	The serial port profile defines the protocols and procedures that should be used by devices using Bluetooth for RS-232 (or similar) serial cable emulation. The scenario covered by this profile deals with legacy applications using Bluetooth as a cable replacement through a virtual serial port abstraction (which in itself is operating system-dependent).	Yes
Headset	This headset profile defines the protocols and procedures that should be used by devices implementing the usage model called "ultimate headset." The most common examples of such devices are headsets, personal computers, and cellular phones.	No
Dial-up Networking (DUN)	The dial-up networking profile defines the protocols and procedures that should be used by devices implementing the usage model called "Internet Bridge" (see the Bluetooth SIG Marketing Requirements Document). The most common examples of such devices are modems and cellular	Yes

Bluetooth Profiles	Description	Palm Feature
	phones. The following scenarios are covered by this profile: • Usage of a cellular phone or modem by a computer as a wireless modem for connecting to a dial-up internet access server, or for using other dial-up services. • Usage of a cellular phone or modem by a computer to receive data calls.	
Fax	The fax profile defines the protocols and procedures that shall be used by devices implementing the fax part of the usage model called "Data Access Points, Wide Area Networks."	No
LAN Access Point (LAP)	The LAN access point profile defines LAN access using PPP over RFCOMM.	Yes
Generic Object Exchange Profile (GOEP)	The generic object exchange profile defines the protocols and procedures that should be used by the applications providing the usage models that need object exchange capabilities. The usage model can be, for example, Synchronization, File Transfer, or Object Push model. The most common devices using these usage models can be notebook PCs, PDAs, smart phones, and mobile phones.	Yes
Object Push	The object push profile defines the requirements for the protocols and procedures that shall be used by the applications providing the object push usage model. This profile makes use of the generic object exchange profile to define the interoperability requirements for the protocols needed by applications. The most common devices using these usage models can be notebook PCs, PDAs, and mobile phones.	Yes
File Transfer	The file transfer profile defines the requirements for the protocols and procedures that shall be used by the applications providing the file transfer usage model. This profile uses the generic object exchange profile as a base profile to define the interoperability requirements for the protocols needed by the applications. The most common devices using these usage	No

Bluetooth Profiles	Description	Palm Feature
	models can be (but are not limited to) PCs, notebooks, and PDAs.	
Synchronization	The synchronization profile defines the requirements for the protocols and procedures that should be used by the applications providing the Synchronization usage model. This profile makes use of the generic object exchange profile to define the interoperability requirements for the protocols needed by applications. The most common devices using these usage models might be notebook PCs, PDAs, and mobile phones.	No

Table 4– Bluetooth Profile Support Requirements

3.1.3 Bluetooth Security

To provide usage protection and information confidentiality, the system has to provide security measures both at the application layer and at the link layer in a manner that is appropriate for a peer environment. This means that in each Bluetooth unit, the authentication and encryption routines must be implemented in the same way. Four different entities are used for maintaining security at the link layer:

- A public address (Bluetooth device address, BD_ADDR), unique for each device.
- Two secret keys (authentication and encryption).
- A random number unique for each new transaction.

The Bluetooth addresses are publicly known and can be obtained manually through UI interactions or automatically through an inquiry routine performed by a Bluetooth unit.

The secret keys are derived during initialization and are never disclosed. Normally, the encryption key is derived from the authentication key during the authentication process. The authentication algorithm always uses a 128-bit key. For the encryption algorithm, the key size may vary between 1 and 16 octets (8 - 128 bits).

The size of the encryption key is configurable for two reasons. The first has to do with the many different requirements imposed on cryptographic algorithms in different countries. The second reason is to facilitate a future upgrade path for the security without the need of a costly redesign of the algorithms and encryption hardware; increasing the effective key size is the simplest way to combat increased computing power at the opponent side.

The encryption key is entirely different from the authentication key (even though the latter is used when creating the former). The lifetime of the encryption key does not necessarily correspond to the lifetime of the authentication key. A new encryption key is generated each time encryption is activated. It is anticipated that the authentication key will be more static in nature than the encryption key. Once the authentication key is established, the particular application running on the Bluetooth device decides when, or if, to change it. The authentication key will often be referred to as the link key.

Palm will handle the generation, utilization and storage of authentication and encryption keys at the OS level.

3.1.3.1 Encryption

User information can be protected by encryption of the packet payload; the access code and the packet header are never encrypted. The encryption of the payloads is carried out with a stream cipher called E0 that is resynchronized for every payload.

3.1.3.2 Authentication

Authentication is the process of verifying who is at the other end of the link. In Bluetooth, the authentication procedure performs authentication for devices based on the stored link key or by pairing (in which the user enters a PIN). The Palm OS platform will handle authentication requests, including the user query for a PIN, at the OS level.

3.1.3.3 Authorization

Authorization is the process of deciding if device X is allowed to have access to service Y. Trusted devices (authenticated and indicated as “trusted”) are allowed access to services. Untrusted or unknown devices may require authorization based on user interaction before access to services is granted. This does not principally exclude that the authorization might be given by an application automatically. Authorization always includes authentication. The Palm OS platform does not support this feature; access concerns beyond authentication are left to the individual application, as in a standard networking environment (see following section for details).

3.1.3.4 Supported Security Modes

The generic access profile specifies three security modes for a device:

- Security mode 1 (non-secure): A device will not initiate any security procedure.

- Security mode 2 (service-level enforced security): A device does not initiate security procedures before channel establishment at L2CAP level. This mode allows different and flexible access policies for applications, especially running applications with different security requirements in parallel.
- Security modes 3 (link level enforced security): A device initiates security procedures before the link set-up at the LMP level is completed.

Palm OS will support security modes 1 and 2.

3.1.3.5 Secure Link Establishment

Secure connections require authentication and encryption. Authentication may or may not require the user to enter a PIN number. The two possible scenarios are:

- Remote device is unknown
- Remote device is known and paired

3.1.3.5.1 Unknown Device

The user will be required to enter a PIN number to establish the connection. PIN numbers are not stored in the Palm OS. The user must enter the PIN number manually whena connection is desired. The same PIN number must be entered on the devices at both ends of the connection. The PIN and a random number are used to create an initialization key (K init). Authentication then needs to be done, whereby the calculation of the authentication response is based on K init instead of the link key.

After a successful authentication, the link key is created. The link key created in the pairing procedure will either be a combination key or one of the unit's unit keys. This common link key is then stored in each device, and the two devices are then paired, such that no PIN entry will be required to establish a secure connection to the device in the future. The remote device might not allow pairing, in which case a PIN would be required to be entered each time a connection is established. At this point, the remote device is considered to be paired, and future secure links will be established using the "Paired Device" method specified bellow. A method for "un-pairing" devices will be provided to the user.

3.1.3.5.2 Paired Device

Paired devices share a common link key that was created and exchanged during the pairing procedure when the two devices communicated for the first time. Once two devices have paired, PIN entry is not required to establish a secure connection.

In paired devices, the link key is used in an authentication procedure based on a challenge-response scheme. The verifier (initiator) sends a random number (the challenge) to the claimant. The claimant calculates a response, which is a function of the challenge, the claimant's Bluetooth address (BD_ADDR) and a secret key (link key) and sends it to the verifier. The verifier checks the response. If the response is correct, the connection is established, otherwise the connection fails.

3.1.4 Device Discovery

In a Bluetooth system, ad-hoc networks are established between Bluetooth devices. The specification provides a method to discover Bluetooth units that are in range, the inquiry procedure. Once a device has been discovered, a connection can be established to it. In addition to an inquiry, a discovery also includes for the retrieval of friendly names from remote devices, since this information is not included in the inquiry response. The following table defines the supported Bluetooth inquiry modes:

Inquiry Mode	**Description**	**Palm Feature**
General	Provides the initiator with the Bluetooth device address, clock, device class, and used page scan mode of general discoverable devices. Also, devices in limited discoverable mode will be discovered using general inquiry. The general inquiry should be used by devices that need to discover devices that are made discoverable continuously or for no specific condition.	Yes
Limited	Provides the initiator with the Bluetooth device address, clock, device class, and used page scan mode of limited discoverable devices. The latter devices are devices that are in range with regard to the initiator, and may be set to scan for inquiry messages with the Limited Inquiry Access Code, in addition to scanning for inquiry messages with the General Inquiry Access Code. The limited inquiry should be used by devices that need to discover devices that are made discoverable only for a limited period of time, during temporary conditions or for a specific event.	No

Table 5– Inquiry Modes

3.1.5 Piconet Support

A *piconet* is formed when a Bluetooth unit creates connections to one or more other Bluetooth units. Bluetooth provides a point-to-point connection (only two Bluetooth units involved), or a point-to-multipoint connection, see Figure 3. In the point-to multipoint connection, the channel is shared among several Bluetooth units.

One Bluetooth unit acts as the master of the piconet, while the other units act as slaves. Up to seven slaves can be active in the piconet. In addition, many more slaves can remain locked to the master in a so-called parked state. These parked slaves cannot be active on the channel, but they remain synchronized to the master. The master controls channel access for both active and parked slaves.

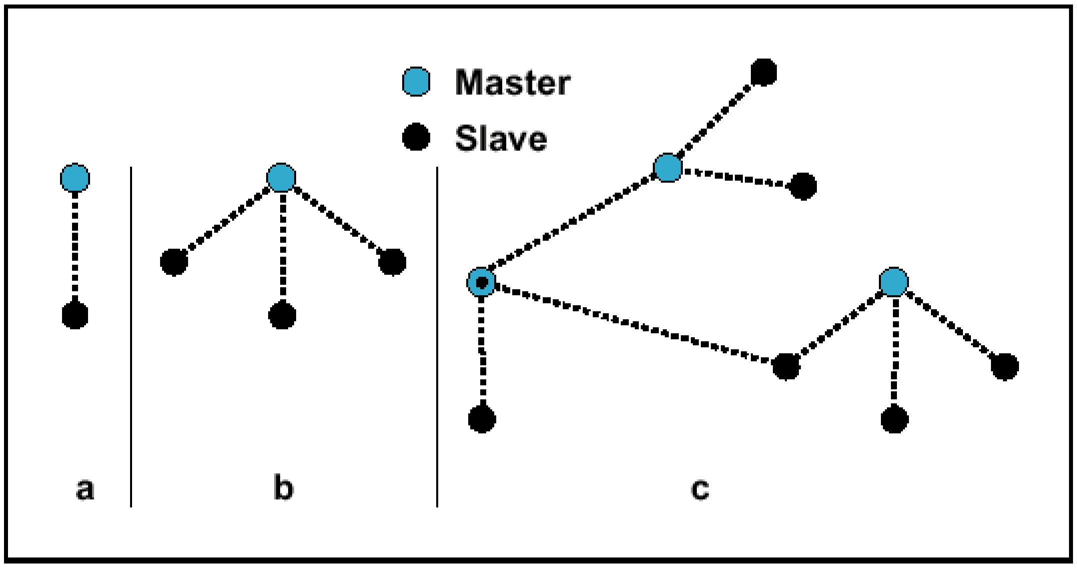

Figure 3 – Piconets (a,b) and Scatternet (C)

When creating piconets, one issue is how to handle existing master/slave ACL connections as new links are added to the piconet. The Bluetooth specification recommends that existing ACL connections be placed in hold or park mode to free up bandwidth while performing page/inquiry and page/inquiry scanning. The other option is to leave current connections active while new connections are established.

Palm will place existing connections in hold mode while new links are established. There are two main scenarios in which a piconet can be created, and Palm OS will support both:

- Master performs inquiry, sees a number of devices, and proceeds to connect to each of them. This case has the link policy problems (link policy race condition or link policy not set). A variant is that the master later performs inquiry to find additional slaves. There are no profiles in the current specification that use this scenario. Palm envisions this scenario to be useful for a game server were the master establishes the connection to each Palm device that wants to participate in a game.

- Master sits in page scan mode, and when a device connects to it, a master/slave switch is performed. Policy setting is not needed if the switch is performed as part of the connect operation. The LAN access profile uses this approach for multipoint LAN access devices. Palm OS will handle the master/slave negotiation automattically.

For the first scenario, hold times for each connection will be determined based on a list of devices that the user has selected to participate in the piconet. Palm OS will perform the following for each device on the list:

- Establish an ACL connection to the device

- Place the device in hold mode for a period of time that is a function of the total number of devices that are to participate in the piconet.
- Delay for a set period of time to allow the slave to enter hold mode.

After all connections have been established, each of the slave hold timers should expire, and the piconet should be operational.

3.1.6 Bluetooth Virtual Serial Port

The Palm OS implementation of the serial port profile will be a Bluetooth Virtual Serial Driver. The Bluetooth virtual serial driver:

- Opens a background thread for the Bluetooth stack.
- Supports only one current active serial channel (point-to-point connection) at a time.
- Uses a blocking open call, thus requiring the use of a progress manager.
- Is opened explicitly as either a client or a server.
- Is utilized, as a client, by the following Palm OS components:
    - PPP
    - HotSync
    - Telephony
- If opened as a server, advertises a list of services (UUIDs) for remote clients to query.
- If opened as a client, creates the necessary baseband and RFCOMM connections, based upon information passed in by the opener.

An RFCOMM-based virtual serial port is far less symmetrical than a physical serial port. In a traditional serial port, there is no need to establish the underlying transport. However in the establishment of a Bluetooth serial port, there are roles for a client and a server device on three different stack levels (ACL, L2CAP, and RFCOMM) as well as responsibilities for registering with and querying SDP.

Users of the Bluetooth virtual serial driver employ a new call, SrmExtOpen, when opening the port. This call uses a custom info block to explicitly define whether it is opening the port as a client or server. A Bluetooth port cannot be opened by the normal SrmOpen function because it does not contain the required info block. Applications should first verify that the serial driver is in fact a Bluetooth serial driver before passing this info block, as other drivers may wish to use the block for other purposes.

The SrmExtOpen information block should contain:

- Client/Server Flag - If the flag is set as server, the virtual driver will register as a listener on a RFCOMM Channel, and create an entry in the SDP. If the flag is client, the virtual will attempt to connect to a remote device.

Clients:

- UUID or RFCOMM channel - The UUID or channel number of the service to connect to, depending upon how the Connection method flag is set.
- Device Address - the Bluetooth device address (BD_ADDR) of the remote device
- Connection method flag – Connect using either the service UUID or the RFCOMM channel number.

Servers:

- UUID -The UUID to register for this service.
- Friendly name - The optional, user-readable name of the service to advertise.

3.1.6.1 Palm OS Usage of the Virtual Serial Driver

The Palm OS components HotSync, PPP, and Telephony Manager can only act as clients. The virtual serial driver will block on the SrmExtOpen call while setting up the underlying transport. Setting up the underlying transport involves three steps – the selection of a target device, connection to the target, and the establishment of the protocol session (SDP, L2CAP, RFCOMM).

The Connection Panel is where the device discovery and connection is performed. The Connection Panel uses known UUIDs and service names for each of the services. For example, PPP will set the remote service name to "Dial-up Networking" if connecting to a modem or a phone. If it is connecting to a PC, however, it will set the UUID to "LAN Access Point using PPP." If the Connection Panel does not provide connection information in the SrmExtOpen info block, a device discovery needs to be performed.

3.1.6.2 Third Party Application Usage of the Virtual Driver

Most third party applications will probably act as clients only; however, in the case of Palm-to-Palm applications, they may need to act as both clients and servers. In the client-only model, the BD_ADDR and UUID of a remote Bluetooth device advertising a desired service should be passed to SrmExtOpen.

In cases where applications can act as either client or server, the virtual driver should initially be configured as a server to advertise its services to the other remote device. At this point, both devices are acting as servers, advertising their services. The role of server describes the device that will be waiting for the establishment of the virtual serial connection.

On one of the devices, a user-initiated action, such as pressing a connect button, will cause that device to reopen the virtual driver as a client. This device can then discover the remote device and advertised service (advertised through a predefined, agreed upon UUID) so that a channel can be opened between the two devices.

3.1.7 Bluetooth HotSync Support

Palm will not support the Bluetooth synchronization profile for HotSync operations. The synchronization profile requires IRMCSync level communications and is not supported in the Palm OS. Palm will support HotSync through the serial port profile. The virtual serial port can then be used for PC HotSync operations, or network-based IP HotSync operations over PPP.

3.1.7.1 PC-Based HotSync Operations

Bluetooth desktop HotSync operations will initially be supported only on the Windows platform, with plans to expand support to additional platforms as Bluetooth technology becomes available on those platforms. Palm will initially use the stacks and APIs provided by individual card manufacturers, but will ultimately standardize on the Bluetooth stack provided by Microsoft when it is made available. There will be a transitional period where vendor-specific stacks will be required, but ultimately, it is anticipated that all vendors will adopt the Microsoft stack, once released.

3.1.7.2 Access Point HotSync Operations

Network HotSync operations using an access point will use PPP.

3.1.7.3 Cell Phone HotSync Operations

Network HotSync operations using a cell phone will use PPP.

3.1.8 Telephony Support

The Palm OS will enable Telephony functionality, allowing the user to dial and control voice calls on a Bluetooth-enabled phone as if it were connected through a serial cable. Since telephony operation is not standardized by a profile in the Bluetooth Specification, universal interoperability may not be possible.

3.1.9 Exchange Library Support

Palm OS will provide a Bluetooth Exchange Library, allowing third party applications to take advantage of Bluetooth support using the standard Exchange Manager APIs. The Bluetooth Exchange Library will conform to the object push and object exchange profiles.

3.1.10 Radio Power Management

Palm has always considered the extended battery life of its handhelds to be a key competitive advantage. Palm intends to continue to preserve battery life by taking advantage of the Bluetooth power efficiency modes (hold, park, and sniff) as described below, and by internal power management functionality built into the Bluetooth radio chipset. Bluetooth radio technology is designed to minimise battery consumption. Battery life is influenced by several factors:

- End user radio utilization profile.
- Default system configuration settings:
  - The radio shall be both discoverable and connectable when the host is active.
  - The radio shall be connectable when the host is suspended.
- Use of defined Bluetooth power efficiency modes:
  - SNIFF(low power): In the sniff mode, the duty cycle of the slave's listen activity can be reduced. If a slave participates on an ACL link, it has to listen in every ACL slot to the master traffic. With the sniff mode, the time slots where the master can start transmission to a specific slave are reduced.
  - HOLD: During the connection state, the ACL link to a slave can be put in a hold mode. This means that the slave temporarily does not support ACL packets on the channel any more. With the hold mode, capacity can be freed to do other things like scanning, paging, inquiring, or attending another piconet. The unit in hold mode can also enter a low-power sleep mode. During the hold mode, the slave unit keeps its active member address (AM_ADDR).
  - PARK: When a slave does not need to participate on the piconet channel but still wants to remain synchronized to the channel, it can enter the park mode, which is a low-power mode with very little activity in the slave. In the park mode, the slave gives up its active member address AM_ADDR. Instead, it receives two new addresses to be used in the park mode:
    - PM_ADDR: 8-bit Parked Member Address
    - AR_ADDR: 8-bit Access Request Address
  - STANDBY (Lowest power): This is the state when the Module has no Bluetooth connection and it is in between scans. The standby state is the default state in the Bluetooth unit. In this state, the Bluetooth unit is in a low-power mode.
- Slaves not addressed in the first slot can go to sleep for the remaining slots the packet occupies.

Applications will not explicitly put the radio into the sniff, park, or standby modes. Instead, power management will be under the control of the OS. When participating in a piconet, the OS will honor requests from the other members of the piconet to enter any of the above defined power savings modes.

4 Bluetooth Hardware Subsystem

Palm will initially ship Bluetooth enabled peripheral devices. Palm OS software support for Bluetooth is abstracted from and independent of the particular hardware implementation. The generalized hardware subsystem is comprised of a Radio/Modem, referred to as RM, Link Controller, referred to as LC, and the Baseband Controller, referred to as BC.

Bluetooth is a short-range radio link that operates in the unlicensed ISM band at 2.4 GHz. A frequency hop transceiver is applied to combat interference and fading. A shaped, binary FM modulation is applied to minimize transceiver complexity. The symbol rate is 1 Ms/s. A slotted channel is applied with a nominal slot length of 625 ms. For full duplex transmission, a Time-Division Duplex (TDD) scheme is used. On the channel, information is exchanged through packets. Each packet is transmitted on a different hop frequency. A packet nominally covers a single slot, but can be extended to cover up to five slots.

The Bluetooth protocol uses a combination of circuit and packet switching. Slots can be reserved for synchronous packets. Bluetooth can support an asynchronous data channel, up to three simultaneous synchronous voice channels, or a channel that simultaneously supports asynchronous data and synchronous voice. Voice channels are not supported by Palm. The asynchronous channel can support maximal 723.2 kb/s asymmetric (and still up to 57.6 kb/s in the return direction), or 433.9 kb/s symmetric.

4.1 Bluetooth Radio Transmitter

The Bluetooth specification defines three RF transmitter power levels. The following table specifies these power classes. Palm devices only support the lowest power output level.

Power Class	**Maximum Output Power (Pmax)**	**Nominal Output Power**	**Minimum Output Power**	**Power Control**	**Palm Feature**
1	100 mW (20 dBm)	N/A	1 mW (0 dBm)	Pmin<+4 dBm to Pmax Optional: Pmin to Pmax	No
2	2.5 mW (4 dBm)	1 mW (0 dBm)	0.25 mW (-6 dBm)	Optional: Pmin to Pmax	No
3	1 mW (0 dBm)	N/A	N/A	Optional: Pmin to Pmax	Yes

Table 6 – Supported Power Class

4.2 HOST CONTROL INTERFACE (HCI) TRANSPORT LAYER

The HCI Transport layer provides an abstraction of the physical interface used to communicate between the handheld and the Bluetooth radio module. In the Palm OS, this layer will be run time replaceable, allowing for the easy substitution of new physical transports by licensees and hardware vendors. The Bluetooth specification currently defines three transport protocols for the Host Control Interface, with a fourth (SD/IO) expected shortly. The table below defines which transports are supported natively:

Transport Layer	**Description**	**Palm Feature**
USB	Universal Serial Bus	No
RS-232		No
UART		Yes
SD Card I/O	Secure Data Card, currently not a defined transport layer in the Bluetooth specification.	Yes

Table 7– HCI Transport Layers

4.3 PALM BLUETOOTH REFERENCE HARDWARE

The reference development hardware platform for Bluetooth is the Palm V Bluetooth Hardcase, which will connect to a Palm V and Palm Vx device. The reference platform is compatible with POSE, allowing development of Bluetooth enabled applications in the standard development environment.

5 Functional Relationships

Figure 4 illustrates the relationships between the functional components.

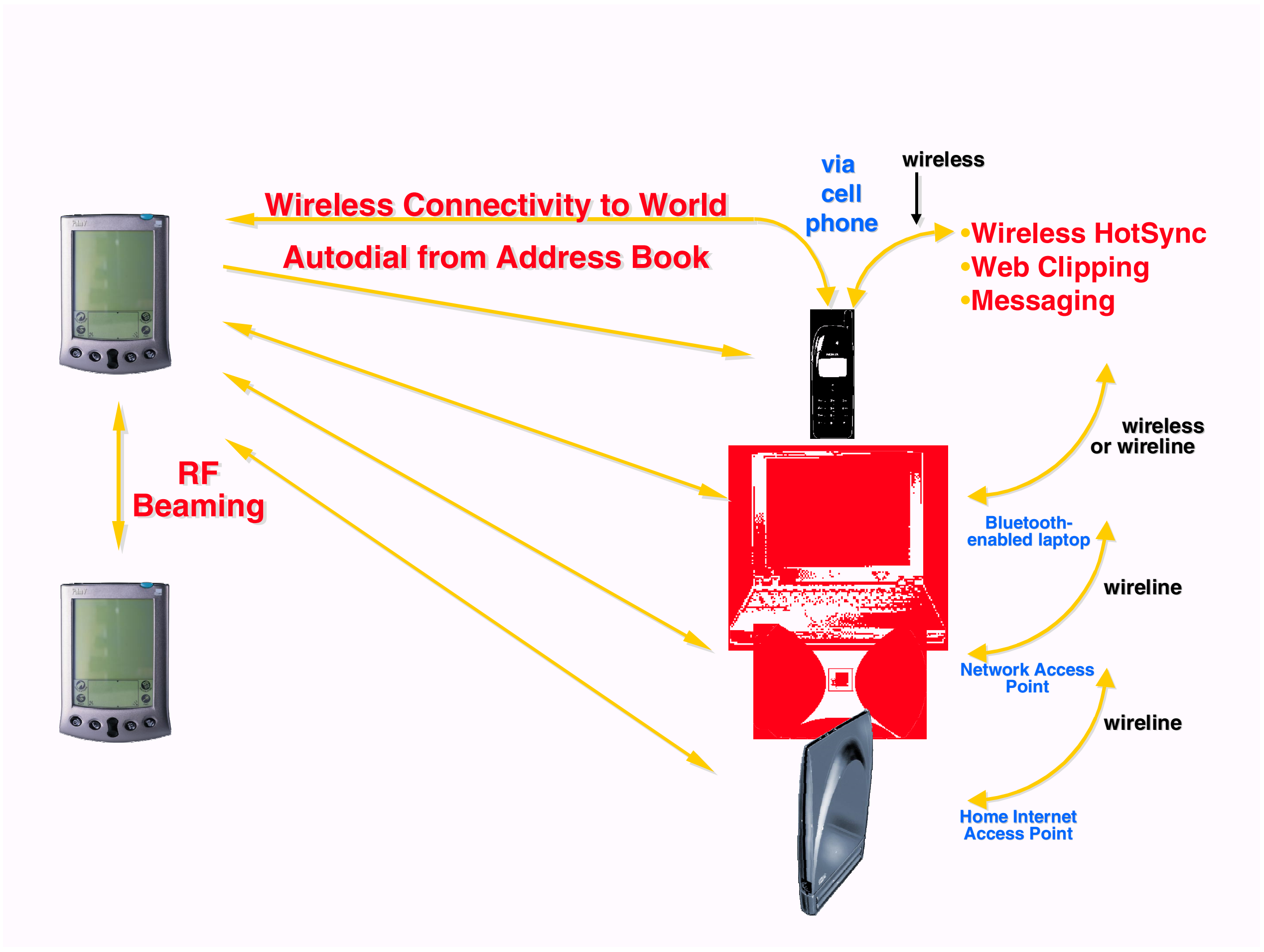

Figure 4 – Bluetooth Functional Relationships

In Figure 4, a Bluetooth-enabled Palm device is able to communicate with a variety of remote Bluetooth-enabled devices. Palm OS uses the profiles defined by the Bluetooth specification in order to support the following usage scenarios:

Feature	Palm Device Connects With:	Required Profiles
Email		
	Cell Phone	Generic Access Service Discovery Serial Port Dial-up Networking
	Access Point	Generic Access Service Discovery Serial Port LAN Access
	PC	Generic Access Service Discovery Serial Port LAN Access
Web Clipping		
	Cell Phone	Generic Access Service Discovery Serial Port Dial-up Networking
	Access Point	Generic Access Service Discovery Serial Port LAN Access
	PC	Generic Access Service Discovery Serial Port LAN Access
Mobile Handset Management		
	Cell Phone	Generic Access Service Discovery Serial Port Dial-up Networking
SMS		
	Cell Phone	Generic Access Service Discovery

Feature	Palm Device Connects With:	Required Profiles
		Serial Port Dial-up Networking
Beaming		
	Other Palm devices	Generic Access Service Discovery Object Push Profile
HotSync Operation		
	PC	Generic Access Service Discovery Serial Port Dial-up Networking
	Cell Phone	Generic Access Service Discovery Serial Port Dial-up Networking
	Access Point	Generic Access Service Discovery Serial Port
Game Server		
	Another Palm	Generic Access Service Discovery

Table 8 – Bluetooth Feature Relationships

Figure 5 illustrates the high level relationship between the Bluetooth stack and the Palm OS. The Bluetooth Developer API provides a level of abstraction between the developer and the Bluetooth stack API. BtTransIF provides a level of abstraction between the Bluetooth hardware and the stack. Details of the BtTransIF will be published in the HDK.

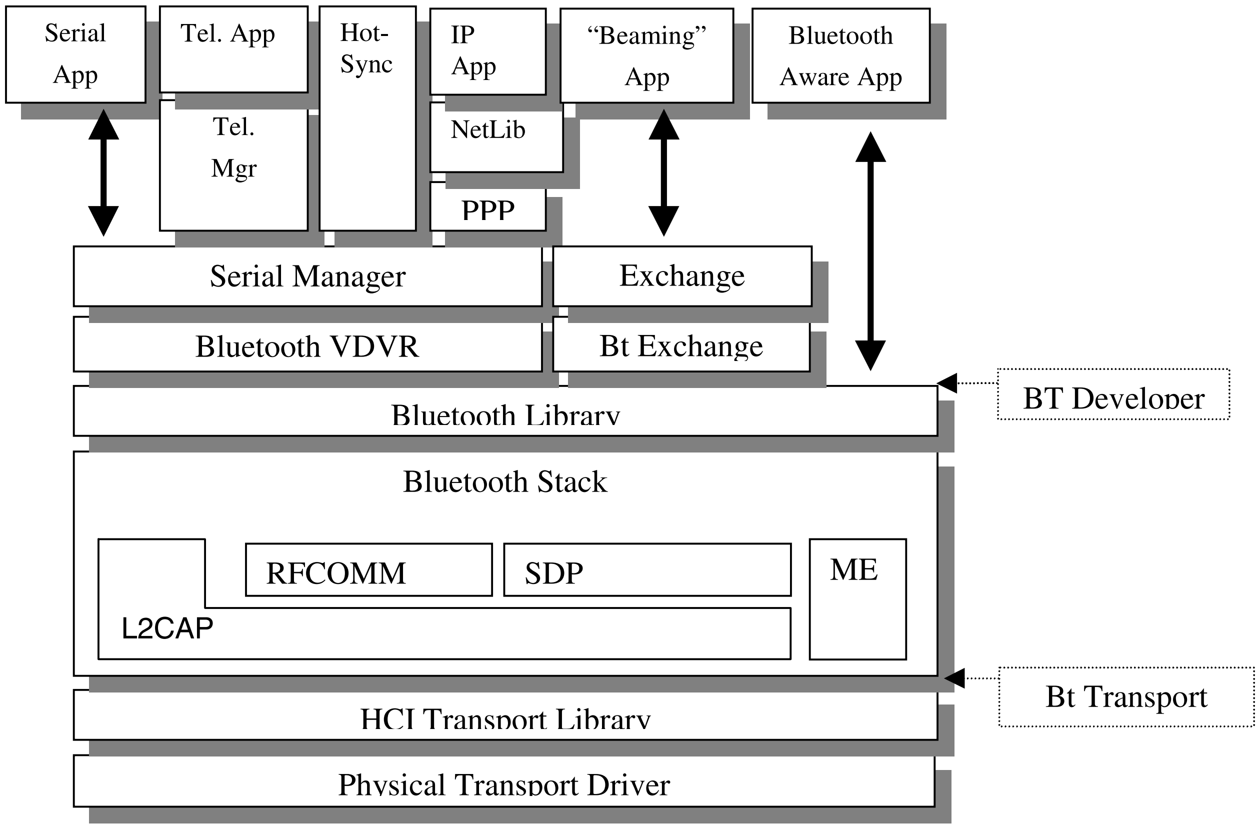

Figure 5 – Overall Bluetooth Protocol Stack Architecture

5.1 DEVELOPING BLUETOOTH ENABLED APPLICATIONS

The Palm OS will expose Bluetooth through multiple interfaces, allowing the application developer to choose the interface that is best suited for the task at hand. Bluetooth development will be supported through the Serial Manager via the Bluetooth Serial Driver for COM emulation, through the Exchange Manager via the Bluetooth Exchange Library for object transfer, as well as through the Bluetooth Developer API. NetLib, or IP, usage will also be supported, with Bluetooth acting as a transport to a network-enabled phone or LAN Access Point. Development of Bluetooth applications must be done on Palm OS v4.0 or higher.

6 Bluetooth Developer API Description

The Bluetooth Developer API allows developers to write Bluetooth enabled applications. The API is asynchronous, meaning that the API returns immediately instead of blocking to wait for a response from the remote device or local radio module. The operation completes in the background, and the application is notified through an event sent to and processed by a registered callback procedure. Events initiated from remote devices (such as receiving data) are also handled by registered callback procedures. Events are divided into two categories:

1. Management events for ACL links and global Bluetooth settings.
2. Socket events for communication through RFCOMM, L2CAP, and SDP.

6.1 LIBRARY FUNCTIONS

6.1.1 BtLibOpen

Purpose:

Opens and initializes the Bluetooth library.

Prototype:

Err BtLibOpen(UInt16 btLibRefNum)

Parameters:

->brLibRefNum	Reference number of the Bluetooth library

Result:

Returns one of the following values:

errNone	No error.

btLibErrAlreadyOpen

The library was already open, and the open count was simply incremented. This is not an error condition.

btLibErrOutOfMemory

There is not enough memory available to open the library.

Comments:

Applications must call this function before using the Bluetooth library. If the Bluetooth library was already open, BtLibOpen increments its open count. Otherwise, it opens the library, initializes it, and starts up the protocol stack component of the library.

6.1.2 BtLibClose

Purpose:

Closes the Bluetooth Library. Closes existing connections, save the current accessible mode, sets mode to connectable only.

Prototype:

Err BtLibClose(UInt16 btLibRefNum)

Parameters:

->brLibRefNum	Reference number of the Bluetooth Library

Returns one of the following values:

errNone

No error.

btLibErrNotOpen

Library was not open.

BtLibErrStillOpen

The library's open count was decremented, but the library was not closed because another process is using it. This is not an error condition.

Comments:

Applications must call this function when they no longer need the Bluetooth library. If the Bluetooth library open count is greater than 1 before this call is made, the count is decremented and btLibErrStillOpen is returned. If the open count was 1, the library is shut down.

6.1.3 BtLibSleep

Purpose:

Automatically called when the Palm OS goes to sleep. Closes existing connections. Change accessible mode to connectable only.

Prototype:

Err BtLibSleep(UInt16 btLibRefNum)

Parameters:

->brLibRefNum	Reference number of the Bluetooth library

Result:

Returns one of the following values:

errNone

No error.

6.1.4 BtLibWake

Purpose:

Automatically called when the Palm OS wakes up. Change accessible mode to connectable and discoverable.

Prototype:

Err BtLibWake(UInt16 btLibRefNum)

Parameters:

->brLibRefNum	Reference number of the Bluetooth library

Result:

Returns one of the following values:

0

No error.

6.2 MANAGEMENT ENTITY FUNCTIONS

The Management Entity (ME) APIs are used for discovery, ACL links, and global Bluetooth settings.

6.2.1 BtLibRegisterManagementNotification

Purpose:

Registers a callback function to process events generated by the Management Entity functions.

Prototype:

Err BtLibRegisterManagementNotification(UInt16 btLibRefNum, BtLibNotifyProcPtr callbackP, Uint32 refCon)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> callbackP	A pointer to an application-defined callback procedure. Cannot be NULL, must be defined.
-> refCon	Application-defined data to pass to the event handler.

Result:

Returns one of the following values:

btLibErrNoError

The callback was registered successfully.

btLibErrParamError

An invalid parameter was passed in.

Callback Events:

None

Comments:

This function registers a callback function that will process events asynchronously generated by the ME. For examples of the callback events that might be received, see the Callback Events sections of each of the function descriptions in this section.

6.2.2 BtLibUnRegisterManagementNotification

Purpose:

Unregisters previously registered ME callbacks. In general, applications should unregister the callback management before terminating.

Prototype:

Err BtLibUnRegisterManagementNotification(UInt16 btLibRefNum, BtLibNotifyProcPtr callbackP)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> callbackP	The callback procedure to unregister.

Result:

Returns one of the following values:

btLibErrNoError

The callback unregistered successfully.

btLibErrParamError

An invalid parameter was passed in.

Callback Event:

None

6.2.3 BtLibStartInquiry

Purpose:

This function starts a Bluetooth inquiry. This is an asynchronous call. An event is generated whenever a device is discovered.

Prototype:

Err BtLibStartInquiry(UInt16 btLibRefNum, UInt8 timeOut, UInt8 maxResp);

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> timeOut	Maximum number of seconds before the Inquiry is halted. The maximum is 60 Seconds. Because of the constants defined within the Bluetooth specification, the time value is be rounded up to nearest multiple of 1.28 seconds. Pass NULL to use the default value specified by the generic access profile (currently, ~10 seconds).
-> maxResp	The maximum number of responses. Responses may not be unique.

Result:
Returns one of the following values:

btLibErrPending
Success. The results will be returned through callback events.

btLibErrBusy
An inquiry is already in progress.

Callback Events:
The following events will be generated:

BtLibManagementEventInquiryResult
Occurs every time a device is discovered.

BtLibManagementEventInquiryComplete
Occurs when the inquiry is complete.

Comments:
The function performs a low-level Bluetooth inquiry, as opposed to a full device discovery. This function only returns the Bluetooth address and the class of the discovered device.

6.2.4 BtLibCancelInquiry

Purpose:
This function cancels a Bluetooth inquiry in process.

Prototype:
Err BtLibCancelInquiry(UInt16 btLibRefNum);

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.

Result:
Returns one of the following values:

btLibErrPending

Success. The results will be returned an event callback.

btLibErrNotInProgress

No inquiry is in progress to be cancelled.

Callback Events:

The following events will be generated:

BtLibManagementEventInquiryCanceled

Occurs to confirm that an inquiry has been cancelled.

Comments:

The function will cancel inquiries initiated using BtLibStartInquiry. It does not cancel Bluetooth discoveries initiated using either BtLibDiscoverSingleDevice or BtLibDiscoverMultipleDevices blocking calls. These can only be cancelled by the user using the Cancel key.

6.2.5 BtLibDiscoverSingleDevice

Purpose:

This blocking call performs a full discovery for an application, including name and feature retrieval and testing. This function takes over the UI and presents a choice box to the user, allowing the user to select one device from the list of devices that were discovered.

Prototype:

Err BtLibDiscoverSingleDevice(UInt16 btLibRefNum, Char* instructionTxt, BtLibClassOfDeviceType* deviceFilterList, UInt8 deviceFilterListLen, BtLibDeviceAddressType *selectedDeviceP, Boolean addressAsName, Boolean showLastList);

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> instructionTxt	The text displayed at the top of the selection box. Pass NULL to display the default text.
-> deviceFilterList	Array of BtLibClassOfDeviceTypes. This function checks each element in this list against the remote device's BtLibClassOfDeviceType value. Any match in the list is considered a success. Pass NULL to skip this test, so that

	all discovered devices are returned.
-> deviceFilterListLen	The number of elements in deviceFilterList.
<- selectedDeviceP	Pointer to an allocated BtLibDeviceAddressType. The device address that the user selected is returned here.
-> addressAsName	If true, show the remote device's Bluetooth addresses instead of friendly names. This option is used for debugging purposes.
-> showLastList	If true, causes all other parameters to be ignored and displays the same list as the previous call to BtLibDiscoverSingleDevice. Calling BtLibStartInquiry in beetween calls to BtLibDiscoverSingleDevice may cause only a partial list to be displayed.

Result:

Returns one of the following values:

btLibErrNoError

Success

Callback Events:

None

Comments:

6.2.6 BtLibDiscoverMultipleDevices

Purpose:

This blocking call performs a full discovery for an application, including name and feature retrieval and testing. This function takes over the UI and presents a choice box to the user, allowing the user to select multiple devices from the list of devices that were discovered.

Prototype:

Err BtLibDiscoverMultipleDevices(UInt16 btLibRefNum, Char* instructionTxt, Char* buttonTxt, BtLibClassOfDeviceType* deviceFilterList, UInt8 deviceFilterListLen, Uint8 *numDevicesSelected, BtLibDeviceAddressType *selectedDeviceP, Boolean addressAsName, Boolen considerAllSelected, Boolen showLastList);

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> instructionTxt	The text displayed at the top of the selection box. Pass NULL to display the default text.
-> buttonTxt	Text for the Done button. Pass NULL to display the default text.
-> deviceFilterList	Array of BtLibClassOfDeviceTypes. This function checks each element in this list against the remote device's BtLibClassOfDeviceType value. Any match in the list is considered a success. Pass NULL to skip this test, so that all discovered devices are returned.
->deviceFilterListLen	The number of elements in deviceFilterList.
<- numDevicesSelected	The number of devices selected. The application should call BtLibGetSelectedDevices to get the actual device list.
-> addressAsName	If true, show the remote device's Bluetooth addresses instead of friendly names. This option is used for debugging purposes.
-> considerAllSelected	If true, skips the user selection phase of the discovery and causes all eligible devices to be considered selected. This parameter isuseful for using the call's filtering capabilities in an unattended setting.
-> showLastList	If true, causes all other parameters to be ignored and displays the same list as the previous call to BtLibDiscoverMultipleDevice. Calling BtLibStartInquiry in beetween calls to BtLibDiscoverMultipleDevice may cause only a partial list to be displayed.

Result:

Returns one of the following values:

btLibErrNoError

Success

Callback Events:

None

Comments:

Use BtLibGetSelectedDevices to retrieve the list of devices that the user selected.

6.2.7 BtLibGetSelectedDevices

Purpose:

Gets the list of devices selected during the last BtLibDiscoverMultipleDevices.

Prototype:

Err BtLibGetSelectedDevices(UInt16 btLibRefNum, UInt8 numDevices, BtLibDeviceAddressType* selectedDeviceArray);

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
<- numDevices	The number of devices selected during the last BtLibDiscoverMultipleDevices.
<-selectedDeviceArray	Pointer to array of devices selected.

Result:

Returns one of the following values:

btLibErrNoError

Success

Callback Events:

None

6.2.8 BtLibRemoteDeviceGetName

Purpose:

Get the remote device's name.

Prototype:

Err BtLibRemoteDeviceGetName(UInt16 btLibRefNum, BtLibDeviceAddressType remoteDeviceP, BtLibFriendlyNameType* nameP, Boolean forceRemote);

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> remoteDeviceP	The address of the device whose name is desired.
<-> nameP	Memory and size of memory for the name to be stored. Actual length of name (including null terminator) is returned.
-> forceRemote	Retrieve the name from the remote device rather than from the local cache.

Result:

Returns one of the following values:

btLibErrPending

The results will be returned through an event.

btLibErrNoError

The name structure was successfully retrieved from the cache.

Callback Events:

BtLibManagementEventNameResult

This event with a status equal to 0 signals that the friendly name was retrieved from the cache successfully.

Comments:

The Bluetooth library keeps a small cache of device names. This routine first checks the cache for a name. If the name is in the cache, the value is returned immediately in the nameP parameter. If the name is not in the cache or if the forceRemote flag is true, the function queries the remote device for its name, forming a temporary ACL connection if one is not already in place. In this case, the function returns a btLibErrPending error, and the name is returned by generating a BtLibManagementEventNameResult event.

6.2.9 BtLibLinkCreate

Purpose:

Create a Bluetooth Asynchronous Connectionless Link (ACL).

Prototype:

Err BtLibLinkCreate(UInt16 btLibRefNum, BtLibDeviceAddressType remoteDeviceArray, UInt8 remoteDeviceCount)

Parameters:

-> btLibRefNum	The reference number for the bluetooth library.
-> remoteDeviceArray	An array of remote device addresses to connect with.
-> remoteDeviceCount	Size of the remoteDevices array (maximum 7).

Result:

Returns one of the following values:

btLibErrPending

Success. The results will be returned through a callback event.

btLibErrorSlave

The device is already a slave in a pre-existing connection and cannot create a new connection.

btLibTooMany

The maximum allowed number of ACL Links has been reached.

Callback Events:

BtLibManagementEventACLConnectComplete

This event with a status equal to 0 signals that the ACL link is up.

Comments:

Attempts to create an ACL link to each of the devices in the remoteDevices array, forming a piconet. When the connection is established, the BtLibManagementEventACLConnectComplete event is generated.

6.2.10 BtLibLinkDisconnect

Purpose:

Disconnects an existing ACL Link. Result returned through event.

Prototype:

Err BtLibLinkDisconnect(UInt16 btLibRefNum, BtLibDeviceAddressType remoteDeviceP)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> remoteDeviceP	The address of the remote device.

Result:

Returns one of the following values:

btLibErrPending

Success. The results will be returned through a callback event.

btLibErrNoConnection

No link exists.

Callback Events:

BtLibManagementEventACLDisconnect

This event signals that the link has disconnected.

6.2.11 BtLibLinkSetState

Purpose:

Set the state of an ACL link.

Prototype:

Err BtLibLinkSetState(UInt16 btLibRefNum, BtLibDeviceAddressType remoteDeviceP, BtLibLinkPrefsEnum pref, void* linkState, UInt16 linkStateSize)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> remoteDeviceP	The address of the remote device (identifies ACL link).
-> pref	Link preference to set (see LINK PREFERENCES section).
-> linkState	Value corresponding to preference (see LINK PREFERENCES section).
-> linkStateSize	Size in bytes of linkState.

Result:

Returns one of the following values:

btLibErrPending

Success.The results will be returned through a callback event.

Callback Events:

BtLibManagementEventAuthenticated

Signals that the link has been authenticated.

BtLibManagementEventEncryptionChange

Signals an encryption change.

BtLibManagementEventRoleChange

Signals a role change from master to slave.

Comments:

Applications use this function to set the state of an ACL link. Which event is generated in response to this function depends on how the state was changed. This function is not used to request information about the ACL state; to request information, use BtLibLinkGetState.

6.2.12 BtLibLinkGetState

Purpose:

Get the state of an ACL link.

Prototype:

Err BtLibLinkGetState(UInt16 btLibRefNum, BtLibDeviceAddressTypePtr remoteDeviceP, BtLibLinkPrefsEnum pref, void* linkState, UInt16 linkStateSize)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> remoteDeviceP	The address of the remote device (identifies ACL link).
-> pref	Link preference to retrieve (see LINK PREFERENCES section).
<- linkState	Value corresponding to preference (see LINK PREFERENCES section).
<- linkStateSize	Size in bytes of linkState.

Result:

Returns one of the following values:

btLibErrNoError

Success. The linkState variable has been filled in.

Callback Events:

None

6.2.13 BtLibSetGeneralPreference

Purpose:

Set the general management preferences.

Prototype:

Err BtLibGeneralPreferenceSet(UInt16 btLibRefNum, BtLibGeneralPrefEnum pref, void* prefValue, UInt16 prefValueSize)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> pref	Link preference to set (see LINK PREFERENCES section).
-> prefValue	Value corresponding the preference (see GENERAL PREFERENCES section).
-> linkStateSize	Size in bytes of prefValue.

Result:

Return one of the following values:

btLibErrNoError

Success

btLibErrPending

The results will be returned through a callback event.

Callback Events:

BtLibManagementEventAccesibleChange

Signals that accessibility has been changed

BtLibManagementEventLocalName

Signals that the name has been changed.

Comments:

6.2.14 BtLibGetGeneralPreference

Purpose:

Get the general management preferences

Prototype:

Err BtLibGeneralPreferenceGet(UInt16 btLibRefNum, BtLibGeneralPrefEnum pref, void* prefValue, UInt16 prefValueSize)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.

-> pref	Link preference to get (see LINK PREFERENCES section).
-> prefValue	Memory to store prefValue (see GENERAL PREFERENCES section).
-> prefValueSize	Size in bytes of prefValue.

Result:

Return one of the following values:

btLibErrNoError

Success. The preference is returned in prefValue.

Callback Events:

None

6.2.15 Link Preferences:

The following table lists the link preferences and corresponding value types for BtLibLinkSetState and BtLibLinkGetState functions.

Pref	**LinkState**
BtLibLinkPref_Authenticated	Boolean
BtLibLinkPref_Encrypted	Boolean
BtLibLinkPref_LinkRole	BtLibConnectionRoleEnum
BtLibLinkPref_RSSI	Int8: Signal strength, read-only, range – 128 <= n <= 127, units dB

Table 9 – Link Preferences

6.2.16 General Preferences

The following table lists general preferences and corresponding value types for BtLibGeneralPreferenceSet and BtLibGeneralPreferenceGet functions:

Pref	**PrefValue**
BtLibPref_LocalName	BtLibFriendlyNameType
BtLibPref_ConnectedAccessible	BtLibAccessiblityInfoType (Allow inquiry and page scan while in a connection, has data bandwidth penality.

Pref	**PrefValue**
	The default is not to allow inquiry and page scan).
BtLibPref_UnConnectedAccessible	BtLibAccessiblityInfoType (In general, only the OS should set BtLibPref_UnConnectedAccessible)
BtLibPref_BtEnabled	Boolean (In general, only the OS should set BtLibPref_BtEnabled)
BtLibPref_BecomeMasterOfInbound	Boolean (Force the master/slave switch on all inbound connections if true. The default is false).

Table 10 – General Preferences

6.3 SOCKETS

The sockets API is used to manage RFCOMM, L2CAP, and SDP communications.

6.3.1 BtLibSocketCreate

Purpose:
Create a socket with foreground notification.

Prototype:
Err BtLibSocketCreate(UInt16 btLibRefNum, BtLibSocketRef* socketRefP, BtLibNotifyProcPtr callbackP, UInt32 refCon, BtLibProtocolEnum socketProtocol)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
<- socketRefP	The returned socket value
-> callbackP	The callback procedure used to respond to socket eventss. Cannot be NULL, must be defined.
-> Refcon	Caller-defined data to pass to the callback procedure.
-> socketProtocol	The protocol (L2CAP, RFComm, or SDP) to associate with this socket.

Result:
Returns one of the following values:

btLibErrNoError
Success.

btLibErrOutOfMemory
Not enough memory to create the socket.

btLibTooMany
The maximum number of sockets allocated for the system has already been reached.

Callback Events:
None

Comments:
Before terminating, applications should destroy all sockets that they have created using BtLibSocketClose.

6.3.2 BtLibSocketClose

Purpose:
Closes a socket, frees associated resources, and kills all connections.

Prototype:
Err BtLibSocketClose(UInt16 btLibRefNum, BtLibSocketRef socket)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> socket	The socket to close.

Result:
Returns one of the following values:

btLibErrNoError
Success.

Callback Events:
None

Comments:

6.3.3 BtLibSocketListen

Purpose:

Set up an L2CAP or RFCOMM socket as a listener.

Prototype:

Err BtLibSocketListen(UInt16 btLibRefNum, BtLibSocketRef socket,
BtLibSocketListenInfoType *listenInfo)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> socket	A listener socket.
-> listenInfo	Protocol-specific listening information.

Result:

Returns one of the following values:

btLibErrNoError

Socket listening for incoming connections.

Callback Events:

BtLibSocketEventOpenRequest

This event is sent when a remote device initiates a connection on this socket. Respond to this event with a call to BtLibSocketConnectionRespond on the listener socket to accept or reject the connection.

Comments:

6.3.4 BtLibSocketConnect

Purpose:

Create an outbound L2CAP or RFCOMM connection.

Prototype:

Err BtLibSocketConnect(UInt16 btLibRefNum, BtLibSocketRef socket,
BtLibSocketConnectInfoType* connectInfo);

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> socket	A socket reference number.
-> listenInfo	Bluetooth device address and protocol-specific connection information.

Result:

Return one of the following values:

btLibErrPending

Success. The results will be returned through an event.

btLibErrNoAclLink

An ACL link for the remote device does not exist

Callback Events:

BtLibSocketEventConnectedOutbound

This event with a status of zero signals success of the connection. A non-zero status gives a reason for failure.

BtLibSocketEventDisconnected

If the connection fails or if connection establishment is successful then this event is broadcast when the channel disconnects.

BtLibSocketEventData

If connection establishment is successful, then this event is broadcast if the remote device sends data.

6.3.5   BtLibSocketConnectionRespond

Purpose:

Accept or reject an in-bound connection on a given listener socket. Called in response to BtLibSocketEventConnectRequest event delivered to a listerner socket.   A new connection socket is returned in the btLibSocketEventConnectedInbound event to the listerner socket after BtLibSocketConnectionRespond is called.

Prototype:

Err BtLibSocketConnectionRespond(UInt16 btLibRefNum, BtLibSocketRef socket, Boolean accept)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> socket	A listener socket reference number.
-> accept	true means accept connection. false means reject connection.

Result:

Returns one of the following values:

btLibErrNoError

Success (if connection is rejected then no notification of the completion of the rejection is necessary)

btLibErrPending

Success. The results will be returned through a callback event.

btLibErrNotListening

The socket passed in is not listening.

btLibErrNoAclLink

An ACL link for the remote device does not exist

Callback Events:

BtLibSocketEventConnectedInbound

The connection was made. Contains the reference for the new connection socket (which will use the same callback as the listener)

BtLibSocketEventDisconnected

The connection failed.

6.3.6 BtLibSocketSend

Purpose:

Sends data over a connected L2CAP or RFCOMM socketCompletion returned through notification.

Prototype:

Err BtLibSocketSend(UInt16 btLibRefNum, BtLibSocketRef socket, UInt8 *data, UInt32 dataLen)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> socket	A socket reference number.
-> data	Pointer to data to send.
-> dataLen	Length of data to send. This value must be less than the MTU for the socket.

Result:

Returns one of the following values:

btLibErrPending
Success. The results will be returned through an event.

btLibErrBusy
A send is already in process.

btLibErrNoAclLink
An ACL link for the remote device does not exist

Callback Events:

BtLibSocketEventSendComplete
This event, with a status of 0, signals that the data has been successfully transmitted.

6.3.7 BtLibSocketGetInfo

Purpose:

Retrieves information for a currently open socket.

Prototype:

Err BtLibSocketGetInfo(UInt16 btLibRefNum, BtLibSocketRef socket, BtLibSocketInfoEnum infoType, void * valueP, UInt32 valueSize);

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> socket	A socket reference number
-> infoType	Type of information to retrieve (see SOCKET INFO section).
<- valueP	Memory in which to store result (see SOCKET INFO section).
-> valueSize	Size in bytes of valueP.

Result:

Returns one of the following values:

btLibErrNoError

Success. Results are placed in valueP.

Callback Events:

None

6.3.8 Socket Information

The following table lists information types and corresponding value types for BtLibSocketGetInfo function:

InfoType	**valueP**
BtLibSocketInfo_Protocol	BtLibProtocolEnum*
BtLibSocketInfo_RemoteDeviceAddress	BtLibDeviceAddressType*
BtLibSocketInfo_SendPending	Boolean*
BtLibSocketInfo_ChannelMtu	UInt32
BtLibSocketInfo_L2CapPsm	BtLibL2CapPsmType*
BtLibSocketInfo_L2CapChannel	BtLibL2CapChannelIDType*
BtLibSocketInfo_RfCommChannelId	BtLibRfCommChannelType*
BtLibSocketInfo_SdpServiceRecordHandle	BtLibSdpRemoteServiceRecordHandle*

Table 11 – Socket Information

6.4 SERVICE DISCOVERY PROTOCOL (SDP)

The Service Discovery Protocol (SDP) API is used create and advertise service records to remote devices and to discover services available on remote devices. Only one outstanding query at a time is allowed per socket.

6.4.1 BtLibSdpGetServiceRecordsByServiceClass

Purpose:

Get the service record handles for service classes advertised on a remote device.

Prototype:

Err BtLibSdpGetServiceRecordsByServiceClass(UInt16 btLibRefNum, BtLibSocketRef socket, BtLibDeviceAddressType* rDev, BtLibSdpUUIDType* uuidList, UInt16 uuidListLen, BtLibSdpRemoteServiceRecordHandle* serviceRecordList, UInt32 numSrvRec);

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> socket	An SDP socket.
-> rDev	The remote device to query.
-> uuidList	List of UUIDs identifying service classes that the remote service should support.
-> uuidListLen	Number of elements in the uuidList. The maximum value is 12
<- serviceRecordList	Array to store results of query.
<-> numSrvRec	Number of service records that serviceRecordList can store. This value is sent to the SDP server so it can limit the number of responses. This value is set upon receiving BTLibSockettEventSdpQueryResponse to the actual number of record handles received.

Result:

Returns one of the following values:

btLibErrPending

Success. The results will be returned through an event.

Callback Events:

BtLibSocketEventSdpServiceRecordHandle

This event with a status of btLibErrNoError signals that the serviceRecordList and numSrvRec has been filled in with the SDP response results. Otherwise, the eventData is not valid because the SDP operation did not complete successfully.

Comments:

Before using this function, use BtLibSdpServiceRecordMapRemote to associate a BtLibSdbRemoteServiceRecordHandle with a remote device. The information about the remote device is returned in this handle.

6.4.2 BtLibSdpServiceRecordMapRemote

Purpose:

Associates an SDP record with a socket and a remote device so that BtLibSdpGetAttributeInServiceRecord can be called to get remote attributes.

Prototype:

Err BtLibSdpServiceRecordMapRemote (UInt16 btLibRefNum, BtLibSocketRef socket, BtLibDeviceAddressType* rDev, BtLibSdpRemoteServiceRecordHandle remoteHandle, BtLibSdpRecordHandle recordH)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> socket	An SDP socket.
-> rDev	The device to query.
-> remoteHandle	Remote service record handle.
-> recordH	An empty SDP record.

Result:

Returns one of the following values:

btLibErrNoError

Indicates that the mapping was successful.

Callback Events:

None

6.4.3 BtLibSdpServiceRecordCreate

Purpose:
Create an SDP Record from existing data or from no data at all.

Prototype:
Err BtLibSdpServiceRecordCreate (UInt16 btLibRefNum, char* sdpData, UInt16 sdpDataSize, BtLibSdpRecordHandle* recordH)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> sdpData	Raw SDP data to initialize the new record (or NULL).
-> sdpDataSize	Size in bytes of sdpData.
<- recordH	Handle to the newly created service record.

Result:
Returns one of the following values:

btLibErrNoError
Success.

Callback Events:
None

6.4.4 BtLibSdpServiceRecordDestroy

Purpose:
Destroy an SDP record (frees memory).

Prototype:
Err BtLibSdpServiceRecordDestroy (UInt16 btLibRefNum, BtLibSdpRecordHandle recordH)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> recordH	SDP record to be destroyed.

Result:
Returns one of the following values:

btLibErrNoError
Success.

Callback Events:
None

6.4.5 BtLibSdpServiceRecordStartAdvertising

Purpose:
Advertise a service record so that the service can be discovered by remote devices.

Prototype:
Err BtLibSdpServiceRecordStartAdvertising (UInt16 btLibRefNum, BtLibSdpRecordHandle recordH)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> recordH	The SDP record to be advertised.

Result:
Returns one of the following values:

btLibErrNoError
Success

btLibErrRemoteRecord
A remote record was passed in recordH. The record must be local.

btLibErrAdvertised
The record is already advertised.

Callback Events:
None

6.4.6 BtLibSdpServiceRecordStopAdvertising

Purpose:

Stop advertising an SDP record.

Prototype:

Err BtLibSdpServiceRecordStopAdvertising (UInt16 btLibRefNum, BtLibSdpRecordHandle recordH)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> recordH	The SDP record to stop advertising.

Result:

Returns one of the following values:

btLibErrNoError

Indicates that the SDP record is no longer advertised.

btLibErrRemoteRecord

A remote record was passed in recordH. The record must be local.

Callback Events:

None

6.4.7 BtLibSdpServiceRecordSetAttributesForSocket

Purpose:

Sets up a basic SDP record for L2CAP and RFCOMM listener sockets.

Prototype:

Err BtLibSdpServiceRecordSetAttributesForSocket(UInt16 btLibRefNum, BtLibSocketRef socket, BtLibSdpUUIDType* serviceUUIDList, UInt8 uuidListLen, Char* serviceName, UInt16 serviceNameLen, BtLibSdpRecordHandle recordH);

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.

-> socket	The socket reference number.
-> serviceUUIDList	List of UUIDs for the service record.
-> uuidListLen	Length of serviceUUIDList. A maximum of 12 entries is allowed.
-> serviceName	Name of the service (English only).
-> serviceNameLen	Size in bytes of serviceName.
-> recordH	The service record to be set up.

Result:
Return one of the following values:

btLibErrNoError
Success.

btLibErrRemoteRecord
A remote record was passed in recordH. The record must be local.

btLibErrAdvertised
An advertised record was passed in recordH. The record must not be advertised.

Callback Events:
None

6.4.8 BtLibSdpServiceRecordSetAttribute

Purpose:
Sets a specific attribute type's value for a given record. This function only works on SDP records that are local and not currently advertised.

Prototype:
Err BtLibSdpServiceRecordSetAttribute (UInt16 btLibRefNum, BtLibSdpRecordHandle recordH, BtLibSdpAttributeEnum attributeID, BtLibSdpAttributeDataType *attributeValues, UInt16 listNumber, UInt16 listEntry);

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> recordH	An SDP record.

-> attributeID	The attribute to set.
-> attributeValues	The value for the attribute.
-> listNumber	Identifies which list to use (usually 0). Ignored for non ProtocolDesriptorListEntry attributes.
-> listEntry	The item to get in the list. Ignored for non-list attributes.

Result:

Returns one of the following values:

btLibErrNoError

Success.

btLibErrRemoteRecord

A remote record was passed in recordH. The record must be local.

btLibErrAdvertised

An advertised record was passed in recordH. The record must not be advertised.

Callback Events:

None

6.4.9 BtLibSdpServiceRecordGetAttribute

Purpose:

Get a specific attribute's value for a given record.

Prototype:

Err BtLibSdpServiceRecordGetAttribute (UInt16 btLibRefNum, BtLibSdpRecordHandle recordH, BtLibSdpAttributeIDType attributeID, BtLibSdpAtributeType* attributeValues, UInt16 listNumber, UInt16 listEntry);

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> recordH	The SDP record
-> attributeID	The attribute to retrieve.
<- attributeValues	The attribute’s value upon return.
-> listNumber	Identifies which list to use (usually 0). Ignored for non ProtocolDescriptorListEntry attributes.

-> listEntry	The item to get in the list. Ignored for non-list attributes.

Result:

Returns one of the following values:

btLibErrNoError

Success.

btLibErrPending

The requested data is for a remote record. The result will be returned through an event.

btLibErrNoAclLink

The ACL link for the remote device does not exist

Callback Events:

BtLibSocketEventSdpGetAttribute

This event with a status of btLibErrNoError signals that attributeValues has been filled in with the SDP response results Otherwise the eventData is not valid because the SDP operation did not complete successfully.

6.4.10 BtLibSdpServiceRecordGetLengthOfStringOrURL

Purpose:

Get the size of a string or URL type of attribute for a given record.

Prototype:

Err BtLibSdpServiceRecordGetLengthOfStringOrURL(UInt16 btLibRefNum, BtLibSdpRecordHandle recordH, BtLibSdpAttributeIDType attributeID, UInt16* size)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> recordH	The SDP record.
-> attributeID	The attribute for which to retrieve the size.
<- size	The returned size of the attribute.

Result:

Returns one of the following values:

btLibErrNoError
Success.

btLibErrPending
The requested data is for a remote record. The result will be returned through an event.

Callback Events:

BtLibSocketEventSdpGetStringLen
This event with a status of btLibErrNoError signals that the size variable has been filled in with the SDP response results. Otherwise the eventData is not valid because the SDP operation did not complete successfully.

Comments:

6.4.11 BtLibSdpServiceRecordGetNumListEntries

Purpose:

Get the number of entries in an attribute list, valid for:

- ServiceClassIdListEntry
- ProtocolDescriptorListEntry
- BrowseGroupListEntry
- LanguageBaseAttributeIDListEntry
- ProfileDescriptorListEntry

Prototype:

Err BtLibSdpServiceRecordGetNumListEntries(UInt16 btLibRefNum, BtLibSdpRecordHandle recordH, BtLibSdpAttributeIDType attributeID, UInt16 listNumber, UInt16 *numEntries)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> recordH	The SDP record.
-> attributeID	Attribute for which to retrieve the number of list entries.
-> listNumber	Identifies which list to use (usually 0). Ignored for non-ProtocolDescriptorListEntry attributes.
<- numEntries	The number of entries in the list upon return.

Result:

Returns one of the following values:

btLibErrNoError

Success

btLibErrPending

The requested data is for a remote record.   The result will be returned through an event.

btLibErrNoAclLink

The ACL link for the remote device does not exist.

Callback Events:

BtLibSocketEventSdpGetNumListEntries

This event with a status of btLibErrNoError signals that the numEntries variable has been filled in with the SDP response results. Otherwise the eventData is not valid because the SDP operation did not complete successfully.

6.4.12 BtLibSdpServiceRecordGetNumLists

Purpose:

Get the number of lists since there can be more than one protocol descriptor list in an SDP record. Valid for ProfileDescriptorListEntry.

Prototype:

Err BtLibSdpServiceRecordGetNumLists(UInt16 btLibRefNum, BtLibSdpRecordHandle recordH, BtLibSdpAttributeIDType attributeID, UInt16 *numLists)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> recordH	An SDP record.
-> attributeID	The attribute for which to retrieve the number of lists.
<- numLists	The number of lists upon return.

Result:

Returns one of the following values:

btLibErrNoError

Success.

btLibErrPending

The requested data is for a remote record. The result will be returned through an event.

Callback Events:

BtLibSocketEventSdpGetNumLists

This event with a status of btLibErrNoError signals that the numLists variable has been filled in with the SDP response results. Otherwise the eventData is not valid because the SDP operation did not complete successfully.

6.4.13 BtLibSdpServiceRecordSetRawAttribute

Purpose:

Sets the value for any attribute for a given record. This function only works on SDP records that are local and not currently advertised.

Prototype:

Err BtLibSdpServiceRecordSetRawAttribute (UInt16 btLibRefNum, BtLibSdpRecordHandle recordH, BtLibSdpRawAttributeIDType attributeID, const UInt8* value, UInt16 valSize)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> recordH	The SDP record.
-> attributeID	The attribute to set.
-> value	Raw SDP attribute data (as it would appear in raw SDP protocol data).
-> valSize	The size in bytes of value.

Result:

Returns one of the following values:

btLibErrNoError

Success.

btLibErrRemoteRecord

The record passed in is associated with a remote service. Only local records are allowed.

btLibErrAdvertised

The record passed in is for an advertised served. Only unadvertised records are allowed.

Callback Events:

None

6.4.14 BtLibSdpServiceRecordGetRawAttribute

Purpose:

Retrieves the value of any attribute for a given record.

Prototype:

Err BtLibSdpServiceRecordGetRawAttribute (UInt16 btLibRefNum, BtLibSdpRecordHandle recordH, BtLibSdpRawAttributeIDType attributeID, Uint8* value, UInt16* valSize)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth library.
-> recordH	An SDP record.
-> attributeID	The attribute to retrieve.
<- value	Raw SDP attribute data retrieved (as it would appear in raw SDP protocol data).
<-> valSize	The size of value buffer upon entry. Upon return, constains the number of bytes in value.

Result:

Returns one of the following values:

btLibErrNoError

Success

btLibErrPending

The requested data is for a remote record. The result will be returned through an event.

btLibErrNoAclLink

The ACL link for the remote device does not exist.

Callback Events:

BtLibSocketEventSdpGetRawAttribute

This event with a status of btLibErrNoError signals that the value and valSize variables have been filled in with the SDP response results. Otherwise, the eventData is not valid because the SDP operation did not complete successfully.

Comments:

6.4.15 BtLibSdpServiceRecordGetSizeOfRawAttribute

Purpose:
Returns the size of any attribute type for a given record.

Prototype:
Err BtLibSdpServiceRecordGetSizeOfRawAttribute (UInt16 btLibRefNum, BtLibSdpRecordHandle recordH, BtLibSdpRawAttributeIDType attributeID, UInt16* size)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> recordH	The SDP record.
-> attributeID	The attribute for which to retrieve the size
<- size	The size of the attribute upon return.

Result:
Returns one of the following values:

btLibErrNoError
Success

btLibErrPending
The requested data is for a remote record. The result will be returned through an event.

btLibErrNoAclLink
The ACL link for the remote device does not exist.

Callback Events:
BtLibSocketEventSdpGetRawAttributeSize
This event with a status of btLibErrNoError signals that the size variable has been filled in with the SDP response results. Otherwise the eventData is not valid because the SDP operation did not complete successfully.

6.5 BLUETOOTH BYTE ORDERING ROUTINES

6.5.1 BtLibHTo/NS/NL/HS/HL

Purpose:

Convert the endianess of integers back and forth between network byte ordering (big endian) and host byte ordering.

- Convert host Int16 to network Int16.
- Convert host long to network long.
- Convert network Int16 to host Int16
- Convert network long to host long

Convert host byte orders

Prototypes:

Int16 BtLibHToNS (Int16 value)

Int32 BtLibHToNL (Int32 value)

Int16 BtLibNToHS (Int16 value)

Int32 BtLibNToHL (Int32 value)

Convert SDP byte orders (SDP is big endian)

Prototypes:

Int32 Sdp32ToHost32 (Int32 value)

Int16 Sdp16ToHost16 (Int16 value)

Int32 Host32ToSdp32 (Int32 value)

Int16 Host16ToSdp16 (Int16 value)

Convert RFCOMM byte orders (RFCOMM is big endian)

Prototypes:

Int32 RFCOMM32ToHost32 (Int32 value)

Int16 RFCOMM16toHost16 (Int16 value)

Int32 Host32ToRFCOMM32 (Int32 value)

Int16 Host16ToRFCOMM16 (Int16 value)

6.6 BLUETOOTH ADDRESS CONVERSION

6.6.1 BtLibAddrBtdToA

Purpose:

Convert 48-bit BtLibAddressType to ASCII colon-seperated form.

Prototype:

Err BtLibAddrBtdToA (UInt16 libRefNum, BtLibDeviceAddressType *btDevP, Char *spaceP, UInt16 spaceSize);

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> btDevP	The address of a Bluetooth device.
<- spaceP	Contains the ASCII formatted Bluetooth devices address upon return.
<-> spaceSize	Size of spaceP in bytes upon entry. Upon return, contains the actual number of bytes used in spaceP excluding the null terminator.

Result:

Return one of the following values:

btLibErrNoError

Success.

Callback Events:

None

6.6.2 BtLibAddrBtdToA

Purpose:

Convert a colon-separated ASCII string Bluetooth device address into a 48-bit BtLibDeviceAddressType.

Prototype:

Err BtLibAddrAToBtd (UInt16 libRefNum, const Char *a, BtLibDeviceAddressType *btDevP)

Parameters:

-> btLibRefNum	The reference number for the Bluetooth Library.
-> a	String containing ASCII colon-seperated Bluetooth device address.
<- btDevP	Contains the converted Bluetooth device address upon return.

Result:

Returns one of the following values:

btLibErrNoError

Success

Callback Events:

None