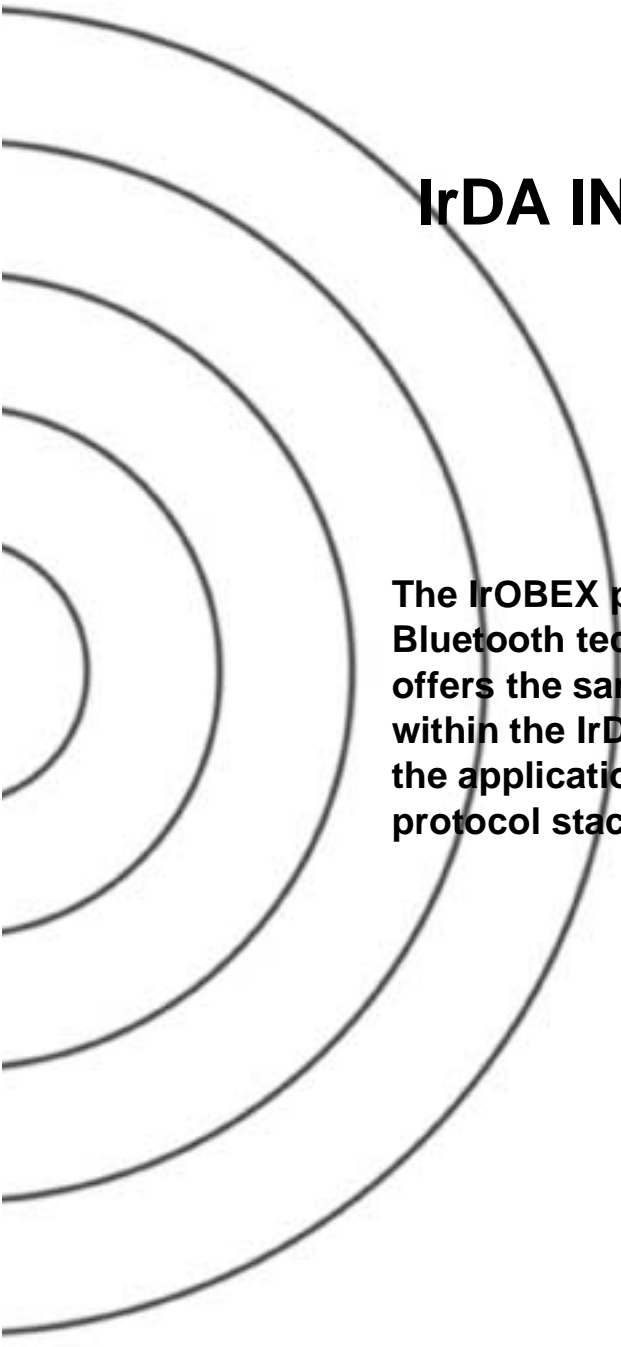


Part F:2

IrDA INTEROPERABILITY



The IrOBEX protocol is utilized by the Bluetooth technology. In Bluetooth, OBEX offers the same features for applications as within the IrDA protocol hierarchy, enabling the applications to work over the Bluetooth protocol stack as well as the IrDA stack.





CONTENTS

1	Introduction	430
1.1	OBEX and Bluetooth Architecture.....	431
1.2	Bluetooth OBEX-Related Specifications	431
1.3	Other IrOBEX Implementations.....	432
2	OBEX Object and Protocol	433
2.1	Object.....	433
2.2	Session Protocol	433
2.2.1	Connect Operation	434
2.2.2	Disconnect Operation.....	435
2.2.3	Put Operation	435
2.2.4	Get Operation	436
2.2.5	Other Operations.....	436
3	OBEX over RFCOMM	437
3.1	OBEX Server Start-up on RFCOMM.....	437
3.2	Receiving OBEX Packets from Serial Port.....	437
3.3	Connection Establishment	438
3.4	Disconnection	438
3.5	Pushing and Pulling OBEX Packets over RFCOMM	438
4	OBEX over TCP/IP	439
4.1	OBEX Server Start-up on TCP/IP	439
4.2	Connection Establishment	439
4.3	Disconnection	440
4.4	Pushing and Pulling OBEX Packets over TCP	440
5	Bluetooth Application Profiles using OBEX	441
5.1	Synchronization	441
5.2	File Transfer	441
5.3	Object Push	442
6	References	443
7	List of Acronyms and Abbreviations.....	444

1 INTRODUCTION

The goal of this document is to enable the development of application programs that function well over both short-range RF and IR media. Each media type has its advantages and disadvantages but the goal is for applications to work over both. Rather than fragment the application domain, this document defines the intersection point where Bluetooth and IrDA applications may converge. That intersection point is IrOBEX [1].

IrOBEX is a session protocol defined by IrDA. This protocol is now also utilized by the Bluetooth technology, making it possible for applications to use either the Bluetooth radio technology or the IrDA IR technology. However, even though both IrDA and Bluetooth are designed for short-range wireless communications, they have some fundamental differences relating to the lower-layer protocols. IrOBEX will therefore be mapped over the lower layer protocols which are adopted by Bluetooth.

This document defines how IrOBEX (OBEX for short) is mapped over RFCOMM [2] and TCP/IP [3]. Originally, OBEX (Object Exchange Protocol) was developed to exchange data objects over an infrared link and was placed within the IrDA protocol hierarchy. However, it can appear above other transport layers, now RFCOMM and TCP/IP. At this moment, it is worth mentioning that the OBEX over TCP/IP implementation is an optional feature for Bluetooth devices supporting the OBEX protocol.

The IrOBEX specification [1] provides a model for representing objects and a session protocol, which structures the dialogue between two devices. The IrOBEX protocol follows a client/server **request-response** paradigm for the conversation format.

Bluetooth uses only the connection-oriented OBEX even though IrDA has specified the connectionless OBEX also. The reasons for the connection-oriented approach are:

- OBEX is mapped over the connection-oriented protocols in the Bluetooth architecture.
- Most of application profiles using OBEX and Bluetooth needs a connection-oriented OBEX to provide the functionality described for the features included in these profiles.
- The connectionless OBEX with the connection-oriented one would raise the interoperability problems, which are not desirable.

1.1 OBEX AND BLUETOOTH ARCHITECTURE

Figure 1.1 depicts part of the hierarchy of the Bluetooth architecture and shows the placement of the OBEX protocol and the application profiles using it (See also [Section 5 on page 439](#)). The protocols can also communicate with the service discovery DB even though the figure does not show it.

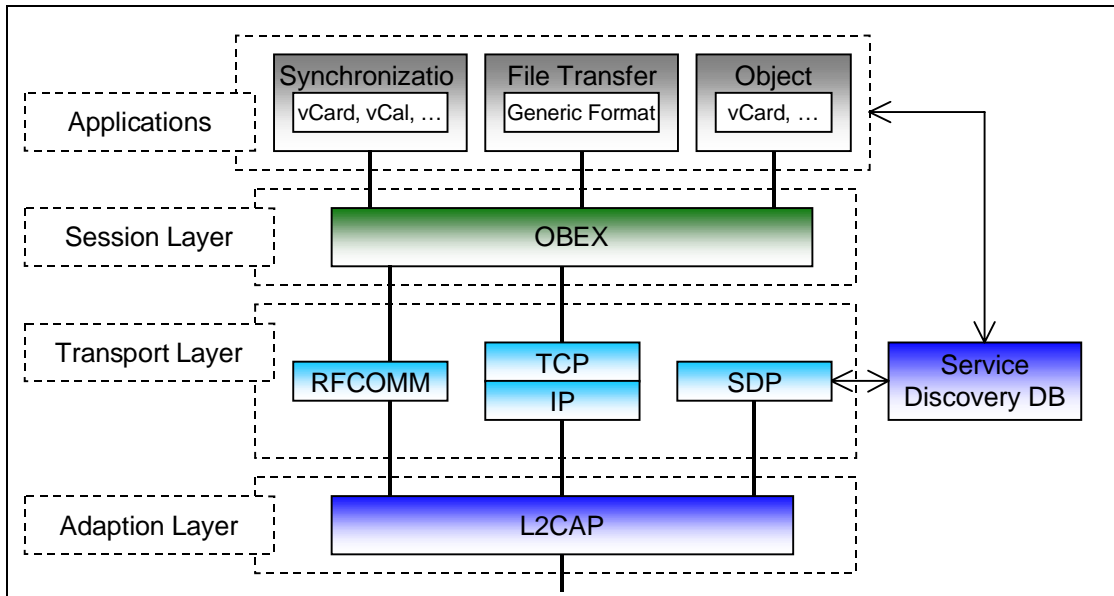


Figure 1.1: Part of Bluetooth Protocol Hierarchy

In the Bluetooth system, the purpose of the OBEX protocol is to enable the exchange of data objects. The typical example could be an object push of business cards to someone else. A more complex example is synchronizing calendars on multiple devices using OBEX. Also, the File Transfer applications can be implemented using OBEX. For the Object Push and Synchronization applications, content formats can be the vCard [4], vCalendar [5], vMessage [6], and vNotes [6] formats. The vCard, vCalendar, vMessage, and vNotes describe the formats for the electronic business card, the electronic calendar-ing and scheduling, the electronic message and mails, and the electronic notes, respectively.

1.2 BLUETOOTH OBEX-RELATED SPECIFICATIONS

Bluetooth Specification includes five separate specifications related to OBEX and applications using it:

1. Bluetooth IrDA Interoperability Specification (This specification)
 - Defines how the applications can function over both Bluetooth and IrDA
 - Specifies how OBEX is mapped over RFCOMM and TCP
 - Defines the application profiles using OBEX over Bluetooth



2. Bluetooth Generic Object Exchange Profile Specification [7]

- Generic interoperability specification for the application profiles using OBEX
- Defines the interoperability requirements of the lower protocol layers (e.g. Baseband and LMP) for the application profiles

3. Bluetooth Synchronization Profile Specification [8]

- Application Profile for the Synchronization applications
- Defines the interoperability requirements for the applications within the Synchronization application profile
- Does not define the requirements for the Baseband, LMP, L2CAP, or RFCOMM.

4. Bluetooth File Transfer Profile Specification [9]

- Application Profile for the File Transfer applications
- Defines the interoperability requirements for the applications within the File Transfer application profile.
- Does not define the requirements for the Baseband, LMP, L2CAP, or RFCOMM.

5. Bluetooth Object Push Profile Specification [10]

- Application Profile for the Object Push applications
- Defines the interoperability requirements for the applications within the Object Push application profile.
- Does not define the requirements for the Baseband, LMP, L2CAP, or RFCOMM.

1.3 OTHER IROBEX IMPLEMENTATIONS

Over IR, OBEX has also been implemented over IrCOMM and TinyTP. The Bluetooth technology does not define these protocols as transport protocols for OBEX, but they can be supported by independent software vendors if desired.

2 OBEX OBJECT AND PROTOCOL

This section is dedicated to the model of OBEX objects and the OBEX session protocol. The section is intended to be read with the IrOBEX specification [1].

2.1 OBJECT

The OBEX object model (Section 2 in [1]) describes how OBEX objects are presented. The OBEX protocol can transfer an object by using the **Put-** and **Get-**operations (See Section 2.2.3 and 2.2.4). One object can be exchanged in one or more **Put-**requests or **Get-**responses.

The model handles both information about the object (e.g. type) and object itself. It is composed of headers, which consist of a header ID and value (See Section 2.1 in [1]). The header ID describes what the header contains and how it is formatted, and the header value consists of one or more bytes in the format and meaning specified by Header ID. The specified headers are **Count**, **Name**, **Type**, **Length**, **Time**, **Description**, **Target**, **HTTP**, **Body**, **End of Body**, **Who**, **Connection ID**, **Application Parameters**, **Authenticate Challenge**, **Authenticate Response**, **Object Class**, and User-Defined Headers. These are explained in detail by Section 2.2 in the IrOBEX specification.

2.2 SESSION PROTOCOL

The OBEX operations are formed by **response-request** pairs. Requests are issued by the client and responses by the server. After sending a request, the client waits for a response from the server before issuing a new request. Each request packet consists of a one-byte opcode (See Section 3.3 in [1]), a two-byte length indicator, and required or optional data depending on the operation. Each response packet consists of a one-byte response code (See Section 3.2.1 in [1]), a two-byte length indicator, and required or optional data depending on the operation.

In the following subsections, the OBEX operations are explained in general.



2.2.1 Connect Operation

An OBEX session is started, when an application asks the first time to transmit an OBEX object. An OBEX client starts the establishment of an OBEX connection. The session is started by sending a **Connect**-request (See Section 3.3.1 in [1]). The request format is:

Byte 0	Bytes 1 and 2	Byte 3	Byte 4	Bytes 5 and 6	Byte 7 to n
0x80 (opcode)	Connect request packet length	OBEX version number	Flags	Maximum OBEX packet length	Optional headers

Note. The Big Endian format is used to define the byte ordering for the PDUs (requests and responses) in this specification as well as in the IrOBEX specification; i.e. the most significant byte (MSB) is always on left and the least significant byte (LSB) on right.

At the remote host, the **Connect**-request is received by the OBEX server, if it exists. The server accepts the connection by sending the successful response to the client. Sending any other response (i.e. a non-successful response) back to the client indicates a failure to make a connection. The response format is:

Byte 0	Bytes 1 and 2	Byte 3	Byte 4	Bytes 5 and 6	Byte 7 to n
Response code	Connect response packet length	OBEX version number	Flags	Maximum OBEX packet length	Optional headers

The response codes are list in the Section 3.2.1 in the IrOBEX specification. The bytes 5 and 6 define the maximum OBEX packet length, which can be received by the server. This value may differ from the length, which can be received by the client. These **Connect**-request and response packets must each fit in a single packet.

Once a connection is established it remains 'alive', and is only disconnected by requests/responses or by failures (i.e. the connection is not automatically disconnected after each OBEX object has completely transmitted).

2.2.2 Disconnect Operation

The disconnection of an OBEX session occurs when an application, which is needed for an OBEX connection, is closed or the application wants to change the host to which the requests are issued. The client issues the **Disconnect**-request (See Section 3.3.2 in [1]) to the server. The request format is:

Byte 0	Bytes 1 and 2	Byte 3
0x81	Packet length	Optional headers

The request cannot be refused by the server. Thus, it has to send the response, and the response format is:

Byte 0	Bytes 1 and 2	Byte 3
0xA0	Response packet length	Optional response headers

2.2.3 Put Operation

When the connection has been established between the client and server the client is able to push OBEX objects to the server. The **Put**-request is used to push an OBEX object (See Section 3.3.3 in [1]). The request has the following format.

Byte 0	Bytes 1 and 2	Byte 3
0x02 (0x82 when Final bit set)	Packet length	Sequence of headers

A **Put**-request consists of one or more request packets, depending on how large the transferred object is, and how large the packet size is. A response packet from the server is required for every **Put**-request packet. Thus, one response is not permitted for several request packets, although they consist of one OBEX object. The response format is:

Byte 0	Bytes 1 and 2	Byte 3
Response code	Response packet length	Optional response headers



2.2.4 Get Operation

When the connection has been established between the client and server, the client is also able to pull OBEX objects from the server. The **Get**-request is used to pull an OBEX object (See Section 3.3.4 in [1]). The request has the following format.

Byte 0	Bytes 1 and 2	Byte 3
0x03 (0x83 when Final bit set)	Packet length	Sequence of headers starting with Name

The object is returned as a sequence of headers, and the client has to send a request packet for every response packet. The response format is:

Byte 0	Bytes 1 and 2	Byte 3
Response code	Response packet length	Optional response headers

2.2.5 Other Operations

Other OBEX operations consist of a **SetPath**-, and an **Abort**-operation. These are carefully explained in the Sections 3.3.5-6 in the IrOBEX specification. It is important to note that the client can send an **Abort**-request after each response – even in the middle of a request/response sequence. Thus, the whole OBEX object does not have to be received before sending an **Abort**-request. In addition to these operations, the IrOBEX specification facilitates user-defined operations, but their use may not necessarily be adopted in Bluetooth.

3 OBEX OVER RFCOMM

This section specifies how OBEX is mapped over RFCOMM, which is the multiplexing and transport protocol based on ETSI TS 07.10 [11] and which also provides a support for serial cable emulation. The Bluetooth devices supporting the OBEX protocol must satisfy the following requirements.

1. The device supporting OBEX must be able to function as either a client, a server, or both
2. All servers running simultaneously on a device must use separate RFCOMM server channels
3. Applications (service/server) using OBEX must be able to register the proper information into the service discovery database. This information for different application profiles is specified in the profile specifications

3.1 OBEX SERVER START-UP ON RFCOMM

When a client sends a connecting request, a server is assumed to be ready to receive requests. However, before the server is ready to receive (i.e. is running) certain prerequisites must be fulfilled before the server can enter the listening mode:

1. The server must open an RFCOMM server channel
2. The server must register its capabilities into the service discovery database

After this, other hosts are able to find the server if needed, and the server listens for get requests from clients.

3.2 RECEIVING OBEX PACKETS FROM SERIAL PORT

As discussed earlier, one object can be exchanged over one or more **Put**-requests or **Get**-responses (i.e. the object is received in one or several packets). However, if OBEX is running directly over the serial port, it does not receive packets from RFCOMM. Instead, a byte stream is received by OBEX from a serial port emulated by RFCOMM.

To detect packets in the byte stream, OBEX has to look for opcodes or response codes (See [Chapter 2.2](#)) depending on whether a packet is a request or a response. The opcodes and response code can be thought of as the start flags of packets. In OBEX packets, there is no 'end flag' that would indicate the end of a packet. However, after the opcode or response code, the length of a packet is received in the next two bytes. Thus, the whole length of a packet is known, and the boundary of two packets can be determined.



All data that is not recognized must be dumped. This could cause a synchronization problem but, considering the nature of the OBEX protocol, this is not a problem over RFCOMM, which provides reliable transport over Bluetooth.

3.3 CONNECTION ESTABLISHMENT

A client initiates the establishment of a connection. However, the following sequence of tasks must occur before the client is able to send the first request for data:

1. By using the SD protocol described in the SDP specification [12], the client must discover the proper information (e.g. RFCOMM channel) associated with the server on which the connection can be established
2. The client uses the discovered RFCOMM channel to establish the RFCOMM connection
3. The client sends the **Connect**-request to the server, to establish an OBEX session. The session is established correctly if the client receives a successful response from the server

3.4 DISCONNECTION

The disconnection of an OBEX session over RFCOMM is straightforward. The disconnection is done by using the **Disconnect**-request (See [Section 2.2.2](#)). When the client has received the response, the next operation is to close the RFCOMM channel assigned to the OBEX client.

3.5 PUSHING AND PULLING OBEX PACKETS OVER RFCOMM

Data is pushed in OBEX packets over RFCOMM by using **Put**-requests (See [Section 2.2.3](#)). After each request, a response is required before the next request with the data can be pushed.

Pulling data from a remote host happens by sending a **Get**-request (See [Section 2.2.4](#)). The data arrives in OBEX response packets. After each response, a new request has to be sent, to pull more data.

4 OBEX OVER TCP/IP

This section specifies how OBEX is mapped over the TCP/IP creating reliable connection-oriented services for OBEX. This specification does not define how TCP/IP is mapped over Bluetooth.

The Bluetooth devices, which support the OBEX protocol over TCP/IP, must satisfy the following requirements:

1. The device supporting OBEX must be able to function as either a client, or a server, or both
2. For the server, the TCP port number 650 is assigned by IANA. If an assigned number is not desirable, the port number can be a value above 1023. However, the use of the TCP port number (650) defined by IANA is highly recommended. The 0-1023 range is reserved by IANA (See [13])
3. The client must use a port number (on the client side), which is not within the 0-1023 range
4. Applications (service/server) using OBEX must be able to register the proper information into the service discovery database. This information for different application profiles is specified in the profile specifications

4.1 OBEX SERVER START-UP ON TCP/IP

When a client sends a **Put-** or **Get-**request, a server is assumed to be ready to receive requests. However, when the server is ready (i.e. is running), certain prerequisites must be fulfilled before the server can enter the listening mode:

1. The server must initialize a TCP port with the value 650 or value above 1023
2. The server registers its capabilities into the service discovery database

After this, other devices are able to find the server if needed, and the server listens for get requests from clients.

4.2 CONNECTION ESTABLISHMENT

A client initiates a connection. However, the following sequence of tasks must occur before a connection can be established:

1. By using, the SD protocol described in the SDP specification [12], the client discovers the proper information (e.g. TCP port number) associated with the server, to enable the connection can be established
2. The client initializes a socket associated to a TCP port number above 1023, and establishes a TCP connection with the host of the server
3. The client sends the **Connect-**request to the server, to establish an OBEX session. The session is established correctly if the client receives a successful response from the server.



4.3 DISCONNECTION

The disconnection of an OBEX session over TCP is straightforward. The disconnection is done by using the **Disconnect**-request (See [Section 2.2.2](#)). When the client has received the response, the next operation is to close the TCP port dedicated for this session.

4.4 PUSHING AND PULLING OBEX PACKETS OVER TCP

See [Section 3.5](#).

5 BLUETOOTH APPLICATION PROFILES USING OBEX

Bluetooth SIG (Special Interest Group) has defined three separate application profiles using OBEX. These profiles are briefly introduced in this section.

5.1 SYNCHRONIZATION

Basically, the synchronization means comparing two object stores, determining their inequalities, and then unifying these two object stores. The Bluetooth devices supporting the synchronization may be desktop PCs, notebooks, PDAs, cellular phones, or smart phones.

The Bluetooth Synchronization profile uses the servers and clients compliant to the IrMC synchronization specified by IrDA (See Section 5 in [6]).

The Bluetooth Synchronization servers and clients must support the level 4 synchronization functionality specified in the IrMC specification.

The actual logic of the synchronization engines which process the synchronization algorithm at the client device is implementation-specific. It is therefore left to the participating software vendors, and is not considered in the Bluetooth specifications.

The synchronization is not limited to one type of application. The Bluetooth synchronization (i.e. the IrMC synchronization) enables four different application classes:

1. Phone Book – provides a means for a user to manage contact records
2. Calendar – enables a user to manage calendar items, and can also be used for ‘to-do’ or task lists
3. Messaging – lets a user manage messages (e.g. e-mails)
4. Notes – provides a means for a user to manage small notes

The interoperability requirements for the Bluetooth Synchronization profile are defined in the Synchronization Profile [8] and Generic Object Exchange Profile [7] specifications.

5.2 FILE TRANSFER

At the minimum, the File Transfer profile is intended for sending and retrieving generic files to and from the Bluetooth device. The File Transfer service also facilitates the browsing of the remote Bluetooth device’s folder.

The interoperability requirements for the Bluetooth File Transfer profile are defined in the File Transfer Profile [9] and Generic Object Exchange Profile [7] specifications.



5.3 OBJECT PUSH

The Object Push profile is the special case of the File Transfer Profile for beaming objects and optionally pulling the default objects. At a minimum, it offers the capability to exchange business cards, but is not limited to this service.

The interoperability requirements for the Object Push profile are defined in the Object Push Profile [10] and Generic Object Exchange Profile [7] specifications.

6 REFERENCES

- [1] Infrared Data Association, IrDA Object Exchange Protocol (IrOBEX), Version 1.2, April 1999
- [2] Bluetooth RFCOMM with TS 07.10, on [page 393](#)
- [3] Internet Engineering Task Force, IETF Directory List of RFCs (<http://www.ietf.org/rfc/>), May 1999.
- [4] The Internet Mail Consortium, vCard - The Electronic Business Card Exchange Format, Version 2.1, September 1996.
- [5] The Internet Mail Consortium, vCalendar - The Electronic Calendaring and Scheduling Exchange Format, Version 1.0, September 1996.
- [6] Infrared Data Association, IrMC (Ir Mobile Communications) Specification, Version 1.1, February 1999.
- [7] Bluetooth Generic Object Exchange Profile, see Volume 2.
- [8] Bluetooth Synchronization Profile, see Volume 2.
- [9] Bluetooth File Transfer Profile, see Volume 2.
- [10] Bluetooth Object Push Profile, see Volume 2.
- [11] ETSI, TS 07.10, Version 6.3.0
- [12] Bluetooth Service Discovery Protocol, see Volume 2.
- [13] Internet Assigned Numbers Authority, IANA Protocol/Number Assignments Directory (<http://www.iana.org/numbers.html>), May 1999.



7 LIST OF ACRONYMS AND ABBREVIATIONS

Abbreviation or Acronym	Meaning
GEOP	Generic Object Exchange Profile
IrDA	Infrared Data Association
IrMC	Ir Mobile Communications
L2CAP	Logical Link Control and Adaptation Protocol
LSB	Least Significant Byte
MSB	Most Significant Byte
OBEX	Object exchange protocol
PDU	Protocol Data Unit
RFCOMM	Serial cable emulation protocol based on ETSI TS 07.10
SD	Service Discovery
SDP	Service Discovery Protocol
SDDB	Service Discovery Database
TCP/IP	Transport Control Protocol/Internet Protocol