OBEX and programming techniques for infrared ports, Bluetooth

We live in a world where wireless devices gradually creep into the life of every family. From a beautiful mobile phone (mobile phone), multi-purpose personal support device (Pocket PC, Palm ...), to home entertainment devices such as speaker system, head

We live in a world where wireless devices gradually creep into the life of every family. From beautiful mobile phones, multi-purpose personal support devices (Pocket PC, Palm .) to home entertainment devices such as speaker systems, DVD players, televisions, all Both are connected together without a cord. If you've ever asked: " *Can you program to transmit image and sound data to your mobile phone via infrared or Bluetooth?* " Then this article will help you figure out how and how to solve the problem. on.

WHAT IS OBEX?

OBEX (OBject EXchange) is a data exchange protocol between infrared devices that was first introduced by IrDA (Infrared Data Association) in 1997. Initially, this protocol was limited to devices only. using infrared light environment, but very soon it was put on most Bluetooth devices by Bluetooth Special Interest Group (Bluetooth SIG).

1. Location OBEX in OSI model

Like other protocols, OBEX protocol is built on the OSI (Open Systems Interconnection) model which consists of two main components:

• ***OBEX session protocol (OBEX session protocol)*** : describes the packet structure in the session between two devices.

• ***OBEX application framework*** : set of OBEX services provided for terminal applications such as file transfer, photo printing .

OBEX Application
Oh

Application floor OBEX Framework OBEX Session Floor session Presentation stage Tiny TP RFCOMM
IrLMP transport floor L2CAP IrLAP Link network floor
Manager Floor links IrPHY Baseband IrDA Bluetooth OSI physical floor

Figure 1: OBEX protocol in OSI model

2. Packet structure in OBEX session protocol

OBEX protocol is mainly used in "push" (Push) or "pull" (Pull) applications, allowing clients to "push" data to servers (servers) or "pull" data. Data from server down. To do this, packets exchanged between the client and server must strictly follow the proposed structure. Here are some of the structures used during file transfer between the client and server (please refer to IrOBEX1.3 documentation on http://www.hitekgroup.net website).

2.1 Required packets

All required packets have the following structure:

Byte 0

Byte 1, 2 Bytes 3 to n
opcode

packet length

Headers
Opcode : The code for each request (Table 1). The highest bit is called the Final bit.

Packet length : Length of the packet

Header : The first information has the following structure:

Byte 0 Byte 1, 2 Byte 3 to n header identifier length (optional) value
Table 1 : Command code required **Command code Type**

Describe

0x80 CONNECT Setting up the session 0x81 DISCONNECT Stop the session 0x02 (0x82) PUT Sending data to the server 0x03 (0x83) GET Retrieving data from server 0xFF ABORT Cancel the session
Table 2 : First information **Identifier Name Description** 0x01 NAME File name (Unicode code) 0xC3 LENGTH File size in bytes 0x48 BODY Data segment of file 0x49 END OF BODY The last piece of data of the file

2.2 Package reply

Like the required packet, the reply packet has the following structure:

Byte 0 Byte 1, 2 Phiên b?n 3 ?? n mã mã ?áp ?ng response d? li?u ?áp ?ng length

Some response opcode are common:

Table 3 : Answer **code Answer code Description** 0x10 (0x90) Resume request or reply 0x20 (0xA0) Confirm request or answer 0x40 (0xC0) Request error 0x41 (0xC1) Error due to no rights 0x43 (0xC3) Transaction session cancel 0x44 (0xC4) No file found

3. Let's solve it together

The file exchange process between client and server is divided into 3 phases:

• *Set up session* : CONNECT

• *Receive / send file* : GET / PUT

• *Stop the session* : DISCONNECT

3.1 Setting up the session (CONNECT)

CONNECT packet has the following structure:

Byte 0 0 Byte 1, 2 Byte 3 Byte 4 Byte 5, 6 Byte 7 to n packet length 0x80 OBEX version number flags maximum OBEX packet length optional headers

OBEX version number : The OBEX protocol version includes major number stored in 4 high bits, the minor number stored at 4 low bits. The current version is 1.0, so this value is 0x10.

Flags : This value is always 0x00 in the current version.

Maximum OBEX packet length : The maximum value of the packet in the OBEX protocol that the device can receive or send. This value on the client and server may vary. Therefore, when setting up the session, the client needs to send this value to the server to check if the maximum packet size that the server can receive or send is?

Optional headers : The first information is optional according to the purpose of each session. In the example below, this value can be ignored.

Request from client bytes Meaning Required code 0x80 CONNECT 0x0007 Packet length = 7 bytes 0x10 OBEX version 1.0 0x00 Flags for current version 0x2000 Maximum size of packet is 8K
Answer from server Answer code 0xA0

SUCCESS

0x0007 Packet length = 7 bytes 0x10 OBEX version 1.0 0x00 Flags for the current version 0x0200 The maximum size of the packet that the server can receive or send is 512 bytes
3.2 Send file (PUT)

Unlike CONNECT packets, PUT packets have some additional early information:

NAME : File name information

LENGTH : Information about file size

BODY : File data segment

END OF BODY : The last piece of data for the file

Below is an example of sending a hello.gif file that is 721 bytes in size from the client (PC) to the server (mobile

phone). Because the file size is larger than the largest packet size the server can receive (with Sony Ericsson T610 being 512 bytes), the client will split the file into two packets to send. Package 1 has a PUT request code of 0x02 (no Final bit set). Pack 2 has a PUT request code of 0x82 (Final bit setting).

Request from client bytes Meaning Code requires 0x02 PUT, Final bit is not set to indicate to the server that the client also sends the next request 0x01E2 Packet length = 482 bytes 0x01 Initial information identifier NAME (file name) 0x0017 Length of information head NAME = 20 + 3 = 23 bytes hello.gif File name (unicode) has NULL end character (20 bytes) 0xC3 Initial information identifier LENGTH (file size) 0x000002D1 File size = 721 bytes 0x48 Identifier first message BODY (file data) 0x01C3 Length of first BODY information = 448 + 3 = 451 bytes 0x . File data size of 448 bytes **Reply from server** Code answer 0x90 CONTINUE, continue to receive love request from client 0x0003 packet length = 3 bytes **Request from client Request** code 0x82 PUT, Final bit is set to show server that this is a packet final 0x0117 Packet length = 279 bytes 0x49 Identifier first information END OF BODY 0x0114 Length of the first information END OF BODY = 276 bytes 0x . File data segment size 721-448 = 273 bytes **Pay words from server** Reply code 0xA0 SUCCESS 0x0003 packet length = 3 bytes

3.3 Receive file (GET)

Unlike PUT packets, GET packets only have the first information NAME. The following example describes the process of receiving a hello.gif file that is 721 bytes in size from the server. First, the client (PC) will send a GET request to the server with the first information NAME is the file name. Because the file size requires larger than 512 bytes (with Sony Ericsson T610), the first answer packet has a reply code of 0x90 (CONTINUE) along with a data section of the file. When receiving the answer code CONTINUE, the client knows that this is not the final piece of the file, so continue sending the request (no first information required from NAME) until the answer code is 0xA0 (SUCCESS) .

Request from client bytes Meaning Required code 0x83 GET, Final bit set 0x001A Packet length = 26 bytes 0x01 Initial information identifier NAME (file name) 0x0017 Length of first information NAME = 20 + 3 = 23 bytes hello.gif File name (unicode) has the ending character NULL (20 bytes) **Reply from server** Code answer 0x90 CONTINUE, data on server 0x01C6 Packet length = 454 bytes 0x48 Identify first information BODY (file data) 0x01C3 Length of first BODY information = 448 + 3 = 451 bytes 0x . 448 bytes of file data size **Request from client Request** code 0x83 PUT, continue to request file 0x0003 Packet length = 3 bytes **reply from server** code 0xA0 SUCCESS replies, last data segment length 0x0117 = 279 byte packet header information 0x49 Identification eND oF A BODY 0x0114 length of the header information 276 END OF BODY = 0x . The data bytes in size 273 bytes file

3.4 Termination (DISCONNECT)

To end the session, the client needs to send the DISCONNECT packet to the server.

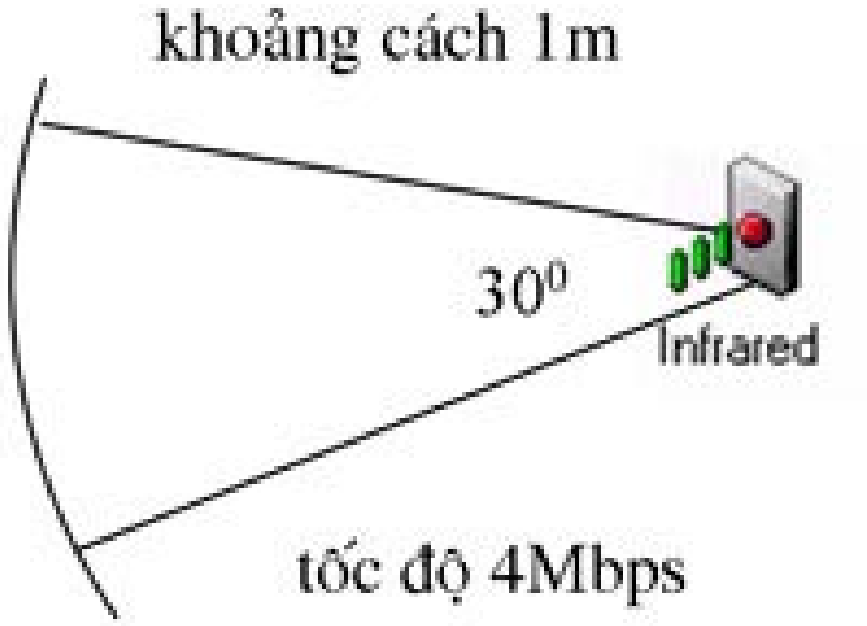
Request from client bytes Meaning Required code 0x81 DISCONNECT 0x0003 Packet length = 3 bytes **Reply from server Reply** code 0xA0 SUCCESS 0x0003 Packet length = 3 bytes

OBEX in mobile phones

Today, mobile phones are not only a means of communication but also an entertainment device with many functions such as listening to music, taking photos, playing games. This means that users always need to update good music, favorite games or save memorable moments on their phones. Below I would like to introduce the

two most common types of connections on most newer mobile phones today are infrared and Bluetooth; at the same time, instructing how to program data exchange between PC and mobile phone using OBEX protocol through these two types of connections.

1. Connect via infrared with C #



The IrDA protocol was first introduced by the IrDA association in 1994 with the aim of enhancing wireless connectivity between devices via infrared light. With an operating range of up to 1 m, an opening angle of between 15 and 30 degrees, the speed can reach 4Mbps, the infrared port is quickly included in most wireless devices such as cell phones, PDAs .

Previously, programming with infrared was a barrier for those unfamiliar with Windows' API (Application Programming Interface), and today, with the version of .NET 2.0, Microsoft has included the This framework is an IrDA library class, allowing programmers to write code more easily and quickly. Like the TCP / IP protocol, the client can establish an IrDA connection to the server by specifying the server's address (similar to the IP address) and the service name on the server (similar to TCP Port).

Each mobile phone has a unique address corresponding to the infrared port on it. The following code allows defining this address:

```
using System.Net.Sockets;

void Form1_Load (object sender, EventArgs e)

{

/ * Initializing the client * /
```

```
IrDAClient irClient = new IrDAClient ();

/ * Search up to 2 devices * /

IrDADeviceInfo [] irDevices = irClient.DiscoverDevices (2);

/ * Print the message when no device is found * /

if (irDevices.Length == 0) {

Console.WriteLine ("No infrared device found");

}

else {

/ * Print the name and address of each device found * /

for (int i = 0; i

Console.WriteLine ("Device Name: {0}", irDevices [i] .DeviceName);

Console.WriteLine ("Device ID: {0}", irDevices [i] .DeviceID);

}

}

}
```

Built-in IrDA services in mobile phones can vary from manufacturer to manufacturer. However, most mobile phones currently offer two main services: IrDA: IrCOMM and IrDA: OBEX. In this article, I only mention IrDA: OBEX service, which allows PC and mobile phones to exchange data via OBEX protocol. Connecting to this service is done through the following code:


```
using System.Net;
```

```
using System.Net.Sockets;

void Form1_Load (object sender, EventArgs e)

{

.

/ * EndPoint setting * /

IrDAEndPoint irEndPoint = new IrDAEndPoint (irDevices [0] .DeviceID, "IrDA: OBEX");

/ * Initialize socket * /

Socket irSocket = new Socket (AddressFamily.Irda, SocketType.Stream, ProtocolType.Unspecified);

/ * Connect to mobile phone via OBEX service * /

irSocket.Connect (irEndPoint);

}
```

Thus, from now on we can exchange data between PC and cellphone via irSocket by pushing / pulling appropriate OBEX packets as mentioned above.

2. Connect via Bluetooth with VC ++

In 1994, the world's leading telecommunications equipment provider, Ericsson, successfully researched wireless technology to allow mobile phones to connect accessories such as headsets and microphones via radio waves.

Four years later, the Bluetooth SIG was established (including Ericsson, Intel, IBM, Nokia and Toshiba) officially released the specification for the 1.0A version of Bluetooth technology in 1999. Bluetooth is also available. The IEEE 802.15.1 name works at 2.4 GHz, the coverage range is up to 100 m (Class 1), the speed can reach 3Mpbs for the 2.0 + EDR (Enhanced Data Rate) version.

Since Windows XP SP1, Microsoft has included its operating system with a Microsoft Bluetooth Stack programming model that allows connecting to Bluetooth devices via Bluetooth socket. One problem is that not all Bluetooth chipsets support Microsoft Bluetooth Stack, but most have a built-in driver as well as a separate SDK for Software Development Kit. We can find Bluetooth chipsets that support Microsoft Bluetooth Stack on SONY VAIO laptops, while IBM series supports Widcomm Bluetooth Stack (the SDK costs up to 1400 USD).

To program with Bluetooth socket we must install the SDK for Windows XP SP2 (http://www.microsoft.com/msdownload/platformsdk/sdkupdate/XPSP2FULLInstall.htm). This SDK will provide some header files as well as the necessary libraries during programming. Since .NET 2.0 version does not yet support Bluetooth, coding must be completely done on the basis of Windows' built-in API library.

The approach to Bluetooth devices is exactly the same as infrared devices, but only, instead of using the built-in .NET functions, you have to move on your own. Below is a code written in VC ++ 2005 that allows searching Bluetooth devices, and printing the name and address of each device found:

```
#include

#include

#include

#pragma comment (lib, "ws2_32.lib")

#pragma comment (lib, "irprops.lib")

/ * compile with: / clr * /

using namespace System;

void main ()

{

WORD wVersionRequested = 0x202;

WSADATA m_data;

/ * Initializing Windows Socket * /

if (WSAStartup (wVersionRequested, & m_data) == 0) {

/ * Set search parameters * /
```

```
WSAQUERYSET querySet;

memset (& querySet, 0, sizeof (querySet));

querySet.dwSize = sizeof (querySet);

/ * Setting the search range to be Bluetooth devices * /

querySet.dwNameSpace = NS_BTH;

HANDLE hLookup;

/ * Set up return information * /

DWORD flags = LUP_RETURN_NAME | LUP_CONTAINERS | LUP_RETURN_ADDR | LUP_FLUSHCACHE | LUP_RETURN_BLOB;

/ * Search up to 10 devices * /

int maxDevices = 10;

/ * Start the search process * /

int result = WSALookupServiceBegin (& querySet, flags, & hLookup);

while (count

BYTE buffer [1000];
```

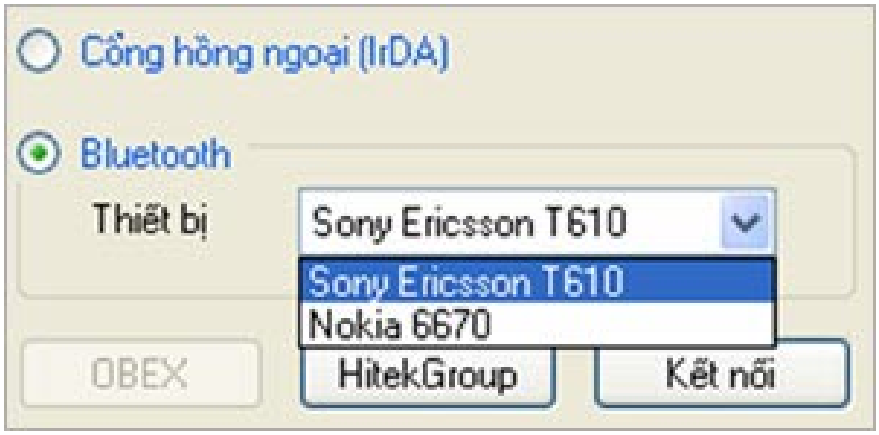


Figure 2: Select Bluetooth connection

```
DWORD bufferLength = sizeof (buffer);
```

```
WSAQUERYSET * pResults = (WSAQUERYSET *) & buffer;

result = WSALookupServiceNext (hLookup, flags, & bufferLength, pResults);

/ * Print the name and address of each device found * /

if (result == 0) {

CSADDR_INFO * pCSAddr = (CSADDR_INFO *) pResults-> lpcsaBuffer;

SOCKADDR_BTH * bts = (SOCKADDR_BTH *) pCSAddr-> RemoteAddr.lpSockaddr;

Console :: WriteLine (pResults-> lpszServiceInstanceName);

Console :: WriteLine ("Device ID: {0}", bts-> btAddr);

count ++;

}

}

/ * End the search process * /

result = WSALookupServiceEnd (hLookup);

WSACleanup ();

}

}
```

On mobile phones today, Bluetooth services are increasingly rich to fully meet the needs of users' wireless connections such as file transfer, photo printing, headphones . These services all have a unique identifier. (UUID) and defined in the bthdef.h header file. In which OBEX Object Push service has the same role as IRDA: OBEX of infrared device. Below is the code that allows connecting to this service:

```
#include

void main ()

{

.

/ * Initializing Windows Socket * /
```

```
if (WSAStartup (wVersionRequested, & m_data) == 0) {

/ * Initialize Bluetooth socket * /

SOCKET s = socket (AF_BTH, SOCK_STREAM, BTHPROTO_RFCOMM);

SOCKADDR_BTH sine;

sin.addressFamily = AF_BTH;

/ * Device address found above * /

sin.btAddr = bts-> btAddr;

/ * OBEX Object Push service identifier * /

sin.serviceClassId = OBEXObjectPushServiceClass_UUID;

sin.port = 0;

/ * Connect to OBEX Object Push * /

int result = connect (s, (SOCKADDR *) & sin, sizeof (sin));

WSACleanup ();

}

}
```

So by using the send () and recv () functions together with the OBEX protocol, we can easily send and receive files between PC and mobile phone.

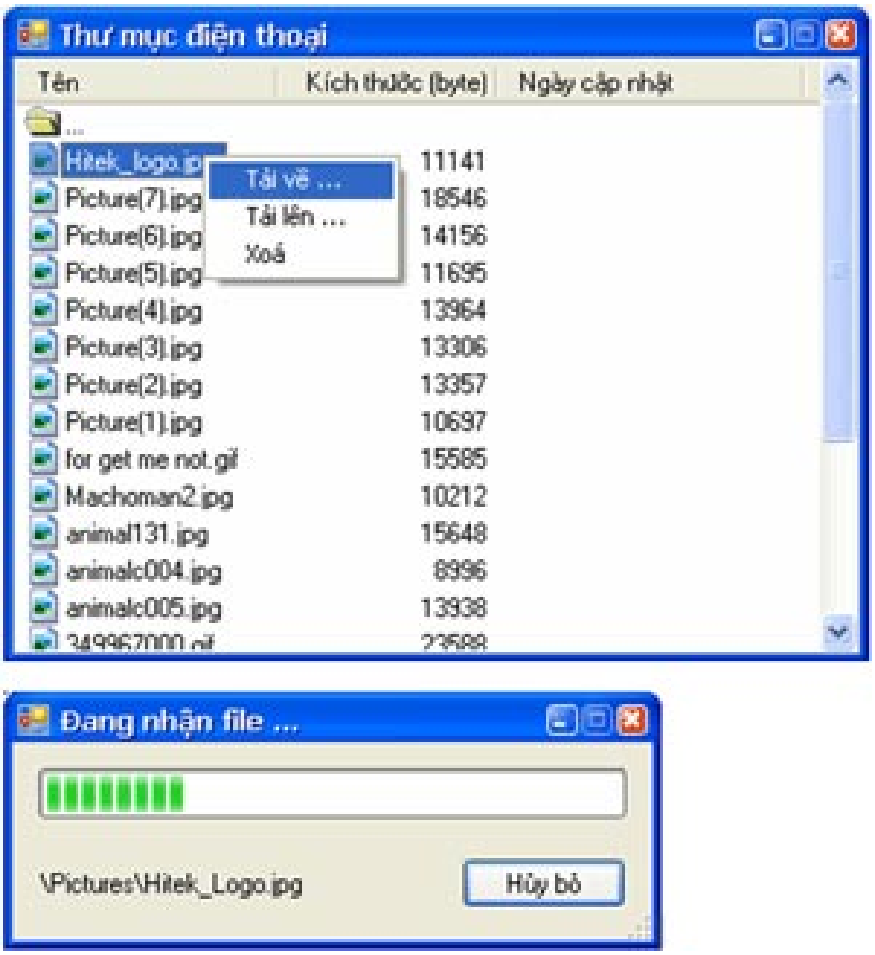


Figure 3: Files in the phone directory

EPILOGUE

Above, I have just introduced you to the two most popular ways to connect wirelessly today to serve the needs of data exchange between PC and mobile phone. For the convenience of developing .NET-based applications, I have written a Bluetooth library class that makes it easy to connect to Bluetooth devices similar to Microsoft's IrDA class library (available for download at www.hitekgroup website). .net). The demo program was written specifically for the Sony Ericsson T610 mobile phone, so it can work inefficiently on other models. Hopefully, through this article, you can build your own file management application suitable for your phone.

Demo materials and programs can be downloaded at www.hitekgroup.net or http://hitekgroup.cabspace.com/.

Nguyen Duc Thang
thangnd@hitekgroup.net