



**KAPITAŁ LUDZKI**  
NARODOWA STRATEGIA SPÓJNOŚCI

**UNIA EUROPEJSKA**  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Instrukcja współfinansowana przez Unię Europejską  
w ramach Europejskiego Funduszu Społecznego  
w projekcie

*„Innowacyjna dydaktyka bez ograniczeń  
– zintegrowany rozwój Politechniki Łódzkiej – zarządzanie Uczelnią,  
nowoczesna oferta edukacyjna i wzmacniania zdolności  
do zatrudniania osób niepełnosprawnych”*

Instrukcja jest dystrybuowana bezpłatnie.

## Instrukcja do laboratorium

---

Piotr Korbel

# Projektowanie i programowanie systemów bezprzewodowych

Korzystanie z interfejsów komunikacyjnych urządzeń przenośnych

Zadanie nr 14 – Studia podyplomowe „Bezprzewodowe systemy nadzoru i monitorowania”



**Politechnika Łódzka**  
Instytut Elektroniki

90-924 Łódź, ul. Żeromskiego 116,  
tel. 042 631 28 83  
[www.kapitalludzki.p.lodz.pl](http://www.kapitalludzki.p.lodz.pl)

## Korzystanie z interfejsów komunikacyjnych urządzeń przenośnych

### 1. Wprowadzenie

Celem zajęć laboratoryjnych jest zapoznanie słuchaczy ze sposobami obsługi interfejsów komunikacyjnych urządzenia przenośnego. W ramach realizacji ćwiczeń przeanalizowane oraz uruchomione zostaną programy wymieniające dane z wykorzystaniem protokołów IrDA oraz Bluetooth. Uruchomiona zostanie aplikacja umożliwiająca wysyłanie krótkich wiadomości tekstowych (SMS).

### 2. Korzystanie z portu podczerwieni IrDA

Wymagania – do realizacji ćwiczenia wymagane jest urządzenie przenośne z wbudowanym portem IrDA (np. ASUS MyPAL 696).

Szablon oprogramowania znajduje się w folderze **IrDAComm**. Po uruchomieniu pakietu Visual Studio należy otworzyć plik **IrDAComm.sln**, a następnie w zakładce Solution Explorer wskazać plik **Form1.cs** i wybrać opcje *View Code* oraz *View Designer* w celu wyświetlenia kodu źródłowego (Dodatek A) oraz projektu formularza głównego aplikacji (Rys. 1).



Rys. 1. Widok formularza głównego programu do komunikacji z wykorzystaniem protokołu IrDA

Za pomocą dostępnych kabli USB podłączyć urządzenie przenośne do stacji deweloperskiej oraz skompilować i uruchomić program (przycisk *F5*) wskazując jako docelową platformę urządzenie z systemem Windows Mobile 6 Professional (*Windows Mobile 6 Professional Device*).

### 3. Korzystanie z interfejsu Bluetooth

Wymagania – do realizacji ćwiczenia wymagane jest urządzenie przenośne z wbudowanym modulem Bluetooth (np. ASUS MyPAL 696, HP iPAQ 114 lub HP iPAQ 614c).

Szablon oprogramowania znajduje się w folderze **Bluetooth\_Chat**. Po uruchomieniu pakietu Visual Studio należy otworzyć plik **Bluetooth\_Chat.sln**, a następnie w zakładce Solution Explorer wskazać plik **Form1.cs** i wybrać opcje *View Code* oraz *View Designer* w celu wyświetlenia kodu źródłowego (Dodatek B) oraz projektu formularza głównego aplikacji (Rys. 2). Następnie wskazać plik klasy pomocniczej **BtThr.cs** i wybrać opcje *View Code* w celu wyświetlenia kodu źródłowego (Dodatek B).



Rys. 2. Widok formularza głównego programu do komunikacji z wykorzystaniem protokołu Bluetooth

Za pomocą dostępnych kabli USB podłączyć urządzenie przenośne do stacji deweloperskiej oraz skompilować i uruchomić program (przycisk *F5*) wskazując jako docelową platformę urządzenie z systemem Windows Mobile 6 Professional (*Windows Mobile 6 Professional Device*).

#### 4. Wysyłanie krótkich wiadomości tekstowych (SMS)

Wymagania – do realizacji ćwiczenia wymagane jest urządzenie przenośne z wbudowanym modulem GSM (np. HP iPAQ 614c).

Szablon oprogramowania znajduje się w folderze **SMS\_Msg**. Po uruchomieniu pakietu Visual Studio należy otworzyć plik **SMS\_Msg.sln**, a następnie w zakładce Solution Explorer wskazać plik **Form1.cs** i wybrać opcje *View Code* oraz *View Designer* w celu wyświetlenia kodu źródłowego (Dodatek A) oraz projektu formularza głównego aplikacji (Rys. 3).



Rys. 3. Widok formularza głównego programu do wysyłania wiadomości SMS

Za pomocą dostępnych kabli USB podłączyć urządzenie przenośne do stacji deweloperskiej oraz skompilować i uruchomić program (przycisk *F5*) wskazując jako docelową platformę urządzenie z systemem Windows Mobile 6 Professional (*Windows Mobile 6 Professional Device*).

Przetestować działanie programu wysyłając przykładową wiadomość. Wprowadzić modyfikacje do kodu źródłowego programu, które umożliwią automatyczne wysyłanie



wiadomości powiązane ze zdarzeniem wybranego typu (np. powodujące okresowe wysyłanie wiadomości z aktualną pozycją odczytaną z odbiornika GPS).





## Literatura – źródła internetowe

- [1] <http://msdn.microsoft.com/>
- [2] <http://www.devx.com/>
- [3] <http://www.codeplex.com/>





## Dodatek A

### *Kod programu do komunikacji pomiędzy urządzeniami za pomocą protokołu IrDA*

Form1.cs

```
using System;

using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.IO;

namespace IrDAComm
{
    public partial class Form1 : Form
    {
        private IrDAListener irListen;
        private IrDAClient irClient;
        private IrDAEndPoint irEndP;
        private IrDADeviceInfo[] irDevices;

        string fileSend;
        string fileReceive;
        string irServiceName;

        int buffersize;

        public Form1()
        {
            InitializeComponent();

            irClient = new IrDAClient();

            // Pliki do wysyłania / odbioru danych
            fileSend = ".\\My Documents\\send.txt";
            fileReceive = ".\\My Documents\\receive.txt";
        }
    }
}
```





```
// Nazwa usługi - jednakowa dla wszystkich
urządzeń!!!
irServiceName = "IrDAftp";

// Maksymalna długość bufora
bufferSize = 256;

this.MinimizeBox = false;

// Przyciski Send i Receive są zablokowane
// do czasu uzyskania połączenia
btnSend.Enabled = false;
btnReceive.Enabled = false;

}

//Odbieranie danych
private void btnReceive_Click(object sender, EventArgs
e)
{
    // Strumień zapisu odebranych danych do pliku
    Stream writeStream;
    try
    {
        writeStream = new FileStream(fileReceive,
            FileMode.OpenOrCreate);
    }
    catch (Exception)
    {
        MessageBox.Show("Couldn't open "
            + fileReceive + " for writing");
        return;
    }

    // Utworzenie połączenia z zaznaczonym urządzeniem
    try
    {
        int i = listBox1.SelectedIndex;
        irEndP = new
IrDAEndPoint(irDevices[i].DeviceID,
            irServiceName);
        irListen = new IrDAListener(irEndP);
        irListen.Start();
    }
    catch (SocketException exSoc)
    {
        MessageBox.Show("Couldn't listen on service "
            + irServiceName + ": ")
    }
}
```







```
        + exSoc.ErrorCode);
    }

    statusBar1.Text = "Listening for "
        + listBox1.SelectedItem.ToString();

    // Utworzenie połączenia dla wykrytej usługi
    // SERVICE_NAME - takie samo!!!
    IrDACLient irClient;
    try
    {
        irClient = irListen.AcceptIrDACLient();
    }
    catch (SocketException exp)
    {
        MessageBox.Show("Couldn't accept socket "
            + exp.ErrorCode);
        return;
    }

    // Połączenie w toku
    if (irListen.Pending() == true)
        statusBar1.Text = "Pending from "
            + irClient.RemoteMachineName;
    else
        statusBar1.Text = "Not pending from "
            + irClient.RemoteMachineName;

    // Odczyt strumienia danych
    Stream baseStream = irClient.GetStream();

    int numToRead;

    // Bufor
    byte[] buffer = new byte[bufferSize];

    // Odczyt danych z urządzenia
    numToRead = 4;
    while (numToRead > 0)
    {
        int numRead = baseStream.Read(buffer, 0,
numToRead);
        numToRead -= numRead;
    }

    // Odczyt długości strumienia danych
    numToRead = BitConverter.ToInt32(buffer, 0);
    statusBar1.Text = "Going to write "
```



```
+ numToRead + " bytes";

// Zapis strumienia do pliku
System.Text.ASCIIEncoding enc = new
System.Text.ASCIIEncoding();
while (numToRead > 0)
{
    int numRead = baseStream.Read(buffer, 0,
buffer.Length);
    numToRead -= numRead;
    writeStream.Write(buffer, 0, numRead);
    textBox1.Text +=
enc.GetString(buffer, 0, buffer.Length);
}

// Potwierdzenie transmisji dla użytkownika
statusBar1.Text = "File received";

baseStream.Close();
writeStream.Close();
irListen.Stop();
irClient.Close();
}

//Wykrywanie urządzeń w zasięgu
private void btnDiscover_Click(object sender,
EventArgs e)
{
    // Utworzenie listy dostępnych urządzeń (max. 3)
    irDevices = irClient.DiscoverDevices(2);

    // Okno dialogowe, gdy nie ma urządzeń w zasięgu
    if (irDevices.Length == 0)
    {
        MessageBox.Show("No remote infrared devices
found!");
        return;
    }

    // Lista dostępnych urządzeń
    // oraz ich właściwości
    string device;
    int ID;
    listBox1.Items.Clear();
    foreach (IrDADeviceInfo irDevice in irDevices)
    {
        ID = BitConverter.ToInt32(irDevice.DeviceID,
0);
```



```
        device = ID.ToString() + " " +  
irDevice.DeviceName + " "  
        + irDevice.CharacterSet + " " +  
irDevice.Hints;  
        listBox1.Items.Add(device);  
    }  
  
    listBox1.SelectedIndex = 0;  
    if (irDevices.Length > 0)  
        statusBar1.Text =  
            irDevices.Length.ToString() + " remote  
device(s)";  
  
    // Odblokowanie przycisków Send i Receive  
    btnSend.Enabled = true;  
    btnReceive.Enabled = true;  
  
}  
  
//Wysyłanie danych  
private void btnSend_Click(object sender, EventArgs e)  
{  
    // Otwarcie pliku i odczyt strumienia zapisanych  
danych  
    Stream fileStream;  
    try  
    {  
        fileStream = new FileStream(fileSend,  
FileMode.Open);  
    }  
    catch (Exception exFile)  
    {  
        MessageBox.Show("Cannot open " +  
exFile.ToString());  
        return;  
    }  
  
    // Utworzenie klienta IrDA i usługi o odpowiedniej  
nazwie  
    try  
    {  
        irClient = new IrDAClient(irServiceName);  
    }  
    catch (SocketException exS)  
    {  
        MessageBox.Show("Create socket error: " +  
exS.Message +
```





```
        " - Did you click Receive on the other
device?");
        return;
    }

    // Dostęp do strumienia danych klienta
    Stream baseStream = irClient.GetStream();

    // Rozmiar przesyłanego pliku, zapis do strumienia
    byte[] length =
BitConverter.GetBytes((int)fileStream.Length);
    baseStream.Write(length, 0, length.Length);

    // Bufor danych odczytywanych z pliku
    byte[] buffer = new byte[bufferSize];

    // Liczba wysyłanych bajtów
    int fileLength = (int)fileStream.Length;
    statusBar1.Text = "Sending " + fileLength + "
bytes";

    // File stream -> Base Stream
    while (fileLength > 0)
    {
        int numRead = fileStream.Read(buffer, 0,
buffer.Length);
        baseStream.Write(buffer, 0, numRead);
        fileLength -= numRead;
    }

    //Zamknięcie strumieni i klienta
    fileStream.Close();
    baseStream.Close();
    irClient.Close();

    //Potwierdzenie przesłania pliku
    statusBar1.Text = "File sent";
    }
}
}
```





## Dodatek B

### *Kod programu do komunikacji pomiędzy urządzeniami za pomocą protokołu Bluetooth*

Form1.cs

```
using System;

using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;
using System.Threading;

namespace Bluetooth_Chat
{
    #region Public Delegates

    // delegates used to call Form1 functions from worker
    thread
    public delegate void DelegateAddMessage(String s);
    public delegate void DelegateThreadFinished();

    #endregion

    public partial class Form1 : Form
    {
        #region Members

        // worker thread
        Thread m_WorkerThread;

        // events used to stop worker thread
        ManualResetEvent m_EventStopThread;
        ManualResetEvent m_EventThreadStopped;

        // Delegate instances used to call user interface
        functions
        // from worker thread:
        public DelegateAddMessage m_DelegateAddMessage;
```





```
public DelegateThreadFinished
m_DelegateThreadFinished;

#endregion

#region Constructor
public Form1()
{
    InitializeComponent();
    this.MinimizeBox = false;
    menuItemDisconn.Enabled = false;

    // initialize delegates
    m_DelegateAddMessage = new
DelegateAddMessage(this.AddMessage);
    m_DelegateThreadFinished = new
DelegateThreadFinished(this.ThreadFinished);

    // initialize events
    m_EventStopThread = new ManualResetEvent(false);
    m_EventThreadStopped = new
ManualResetEvent(false);
}
#endregion
#region Message Handlers

// Start thread button is pressed
private void btnStartThread_Click(object sender,
System.EventArgs e)
{
    textBoxMessages.Text = "";
    btnStartThread.Enabled = false;
    btnStopThread.Enabled = true;

    // reset events
    m_EventStopThread.Reset();
    m_EventThreadStopped.Reset();

    // create worker thread instance
    m_WorkerThread = new Thread(new
ThreadStart(this.WorkerThreadFunction));

    m_WorkerThread.Name = "Worker Thread Sample"; //
looks nice in Output window

    m_WorkerThread.Start();
}
```





```
    }

    // Stop Thread button is pressed
    private void btnStopThread_Click(object sender,
System.EventArgs e)
    {
        StopThread();
    }

    // Exit button is pressed.
    private void btnExit_Click(object sender,
System.EventArgs e)
    {
        this.Close();
    }

    // Form is closed.
    // Stop thread if it is active.
    private void MainForm_Closed(object sender,
System.EventArgs e)
    {
        StopThread();
    }

#endregion

#region Other Functions

    // Worker thread function.
    // Called indirectly from btnStartThread_Click
    private void WorkerThreadFunction()
    {
        BtThr btThr;

        btThr = new BtThr(m_EventStopThread,
m_EventThreadStopped, this);

        btThr.Run();
    }

    // Stop worker thread if it is running.
    // Called when user presses Stop button of form is
closed.
    private void StopThread()
    {
        //if (m_WorkerThread != null &&
m_WorkerThread.IsAlive) // thread is active
```





```
if (m_WorkerThread != null) // thread is active
{
    // set event "Stop"
    m_EventStopThread.Set();

    // wait when thread will stop or finish
    /*while (m_WorkerThread.IsAlive)
    {
        // We cannot use here infinite wait
        because our thread // makes synchronous calls to main form,
        this will cause deadlock.
        // Instead of this we wait for event some
        appropriate time // (and by the way give time to worker
        thread) and // process events. These events may
        contain Invoke calls.
        if (WaitHandle.WaitAll(
            (new ManualResetEvent[] {
m_EventThreadStopped })),
            100,
            true))
        {
            break;
        }

        Application.DoEvents();
    }*/
}

ThreadFinished(); // set initial state of
buttons
}

// Add string to list box.
// Called from worker thread using delegate and
Control.Invoke
private void AddMessage(String s)
{
    textBoxMessages.Text += s;
}

// Set initial state of controls.
// Called from worker thread using delegate and
Control.Invoke
private void ThreadFinished()
```







```
{
    btnStartThread.Enabled = true;
    btnStopThread.Enabled = false;
}

#endregion

private void menuItemConn_Click(object sender,
EventArgs e)
{
    if (serialPortTx.IsOpen)
        serialPortTx.Close();

    serialPortTx.Open();
    menuItemConn.Enabled = false;
    menuItemDisconn.Enabled = true;
}

private void menuItemDisconn_Click(object sender,
EventArgs e)
{
    serialPortTx.Close();
    menuItemConn.Enabled = true;
    menuItemDisconn.Enabled = false;
}

private void buttonSend_Click(object sender, EventArgs
e)
{
    //
    if (textBoxSend.Text.Length != 0)
    {
        serialPortTx.WriteLine(textBoxSend.Text);
        AddMessage(textBoxSend.Text + "\r\n");
        //textBoxMessages.Text += textBoxSend.Text +
"\r\n";
        textBoxSend.Text = "";
    }
}
}
```

BtThr.cs





```
using System;

using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.IO.Ports;

namespace Bluetooth_Chat
{
    public class BtThr
    {
        #region Members

        // Main thread sets this event to stop worker thread:
        ManualResetEvent m_EventStop;

        // Worker thread sets this event when it is stopped:
        ManualResetEvent m_EventStopped;

        // Reference to main form used to make synchronous user
        interface calls:
        Form1 m_form;

        SerialPort spRx;

        #endregion

        #region Functions

        public BtThr(ManualResetEvent eventStop,
                    ManualResetEvent eventStopped,
                    Form1 form)
        {
            m_EventStop = eventStop;
            m_EventStopped = eventStopped;
            m_form = form;
            this.spRx = new
System.IO.Ports.SerialPort("COM7");
        }

        // Function runs in worker thread.
        public void Run()
        {
            int i;
            String s;
            s = " ";
        }
    }
}
```





```
try
{
    spRx.Open();
}
catch(Exception e)
{
    m_form.Invoke(m_form.m_DelegateThreadFinished,
null);
    return;
}

for (i = 1; i <= 1000; i++)
{
    s = "";
    try
    {
        s = spRx.ReadLine() + "\r\n";
    }
    catch (Exception e)
    {
    }

    Thread.Sleep(1000);

    // Make synchronous call to main form.
    // MainForm.AddString function runs in main
thread.
    // To make asynchronous call use
BeginInitInvoke
    m_form.Invoke(m_form.m_DelegateAddMessage,
new Object[] {s});

    // check if thread is cancelled
    /*if ( m_EventStop.WaitOne(0, true) )
    {
        // inform main thread that this thread
stopped
        m_EventStopped.Set();

        return;
    }*/
}
spRx.Close();

// Make asynchronous call to main form
```





```
        // to inform it that thread finished
        m_form.Invoke(m_form.m_DelegateThreadFinished,
null);
    }

    #endregion
}
}
```





## Dodatek C

### *Kod programu do wysyłania krótkich wiadomości tekstowych (SMS)*

Form1.cs

```
using System;

using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Microsoft.WindowsMobile.PocketOutlook;
using Microsoft.WindowsMobile.PocketOutlook.MessageInterception;
using System.Reflection;

namespace SMS_Msg
{
    public partial class Form1 : Form
    {
        MessageInterceptor msgInterceptor;
        Guid appID = new Guid("{5FCE51C5-44D6-48b8-8CAF-F333975A39E3}");

        public Form1()
        {
            InitializeComponent();
            this.MinimizeBox = false;
        }

        private void buttonSendSms_Click(object sender,
EventArgs e)
        {
            try
            {
                SmsMessage sms = new SmsMessage();

                sms.Body = textBoxMsg.Text;

                sms.To.Add(new Recipient(textBoxNo.Text));
                sms.RequestDeliveryReport = true;
                sms.Send();
                statusBar1.Text = "Message sent!";
            }
        }
    }
}
```





```
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

private void Form1_Load(object sender, EventArgs e)
{
    if
(MessageInterceptor.IsApplicationLauncherEnabled(appID.ToString()
g()))
    {
        msgInterceptor = new
MessageInterceptor(appID.ToString(), true);
    }
    else
    {
        msgInterceptor = new
MessageInterceptor(InterceptionAction.NotifyAndDelete, true);

        msgInterceptor.MessageCondition = new
MessageCondition(MessageProperty.Body,
MessagePropertyComparisonType.StartsWith, "Hello", false);

        string appPath =
Assembly.GetExecutingAssembly().GetName().CodeBase;

msgInterceptor.EnableApplicationLauncher(appID.ToString(),
appPath);
    }

    msgInterceptor.MessageReceived += new
MessageInterceptorEventHandler(msgInterceptor_MessageReceived)
;
}

void msgInterceptor_MessageReceived(object sender,
MessageInterceptorEventArgs e)
{
    //Obsługa odbioru wiadomości
    SmsMessage smsReceived = (SmsMessage)e.Message;
    textBoxMsgReceived.Text = "";
    textBoxMsgReceived.Text = "Message from: " +
smsReceived.From.ToString() + "\n";
}
```





```
textBoxMsgReceived.Text += smsReceived.Body;
statusBar1.Text = "Message received!";

String appPath =
Assembly.GetExecutingAssembly().GetName().CodeBase;
System.Diagnostics.Process.Start(appPath, "");
}

private void button1_Click(object sender, EventArgs e)
{
    if
(MessageInterceptor.IsApplicationLauncherEnabled(appID.ToString()))
    {
        msgInterceptor.DisableApplicationLauncher();

        msgInterceptor.MessageReceived -=
msgInterceptor_MessageReceived;
        msgInterceptor.Dispose();
        if
(!MessageInterceptor.IsApplicationLauncherEnabled(appID.ToString()))
        {
            MessageBox.Show("Application
unregistered.");
        }
        else
        {
            MessageBox.Show("Application not
unregistered");
        }
    }
}
}
```

