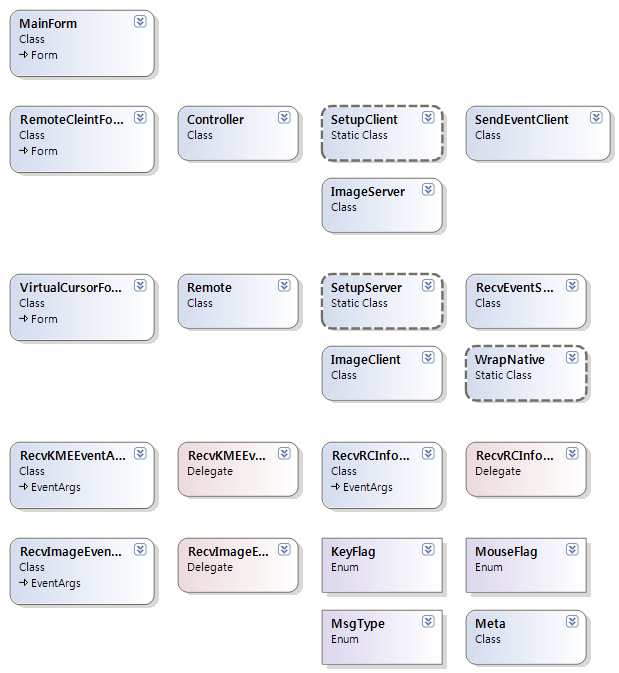
1. 원격 제어 프로그램 만들기

여기에서 만드는 원격 제어 프로그램은 클라이언트와 서버를 하나로 만들기로 합시다. 화면도 전체 화면을 제어할 수 있고 선택적인 영역을 제어하는 부분이나 특정 프로세스를 제어하는 부분은 제공하지 않을 것입니다.

먼저 윈도우즈 Forms 응용 프로젝트 템플릿을 선택하여 프로젝트를 생성하세요. 그리고 기본으로 제공하는 Form1의 이름을 솔루션 창에서 이름 바꾸기를 하여 MainForm으로 변경하세요. 솔루션 창에서 폼의 이름을 바꾸면 마법사에 의해 소스 코드의 클래스 이름도 자동으로 변환해 줍니다.

다음은 여기에서 만들 원격 제어 프로그램에 정의할 형식들입니다.



[그림 9.1] 원격 제어 프로그램에 정의할 형식들

|  |  |
| --- | --- |
| 클래스 명 | 역할 |
| MainForm | 원격 제어 서버 설정, 원격 제어 허용 |
| RemoteClientForm | 원격 호스트 화면 시각화 폼 |
| VirtualCursorForm | 가상(원격 제어) 커서 |
| Controller | 원격 제어 컨트롤 |
| Remote | 피 원격 제어 컨트롤 |
| SetupClient | 원격 제어 요청 송신(클라이언트 소켓) |
| SetupServer | 원격 제어 요청 수신(서버 소켓) |
| ImageServer | 원격 호스트 화면 수신(서버 소켓) |
| ImageClient | 원격 호스트 화면 송신(클라이언트 소켓) |
| SendEventClient | 원격 제어 키보드, 마우스 이벤트 송신(클라이언트 소켓) |
| RecvEventServer | 원격 제어 키보드, 마우스 이벤트 수신(서버 소켓) |
| RecvKMEEventArgs | 원격 제어 키보드, 마우스 이벤트 수신 이벤트 아규먼트 |
| RecvKMEEventHandler | 원격 제어 키보드, 마우스 이벤트 수신 이벤트 대리자 |
| RecvImageEventArgs | 원격 호스트 화면 수신 이벤트 아규먼트 |
| RecvImageEventHandler | 원격 호스트 화면 수신 이벤트 대리자 |
| RecvRCInfoEventArgs | 원격 제어 요청 Accept 이벤트 아규먼트 |
| RecvRCInfoEventHandler | 원격 제어 요청 Accept 이벤트 대리자 |
| KeyFlag | 키보드 이벤트 열거형 |
| MouseFlag | 마우스 이벤트 열거형 |
| MsgType | 키보드, 마우스 이벤트 열거형 |
| Meta | 키보드, 마우스 이벤트 관련 정보 |

[표 9.1] 원격 제어 프로그램 형식과 역할

* 1. 원격 제어 요청 및 허용

제일 먼저 원격 제어를 요청하는 SetupClient 클래스와 허용하는 SetupServer를 구현합시다.

SetupClient는 단순히 상대에게 누가 요청하는지 알려주는 역할만 할 것입니다. 값을 유지할 필요도 없고 단순한 기능을 수행하는 클래스여서 정적 클래스로 정의합시다.

public static class SetupClient

{

Setup 클라이언트에서는 원격 제어하고자 하는 상대 IP와 포트 정보를 입력 인자로 받습니다.

public static void Setup(string ip, int port)

{

입력 인자로 IPEndPoint 개체를 생성합니다.

IPAddress ipaddr = IPAddress.Parse(ip);

IPEndPoint ep = new IPEndPoint(ipaddr, port);

TCP 방식의 소켓을 생성합니다.

Socket sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

그리고 연결한 후에 바로 닫습니다.

sock.Connect(ep);

sock.Close();

원격 제어를 수용할 지 여부를 판단하는 곳에 누가 연결을 요청하였는지 알려주기 위함일 뿐 별다른 메시지 전달할 것은 없습니다.

}

}

원격 제어 요청을 수신하여 수락 혹은 거절하는 SetupServer에서는 상대측에서 원격 제어 요청이 온 시점을 알아야 합니다. 이를 위해 이벤트 처리를 할 수 있게 대리자와 이벤트 인자 형식을 정의합시다.

public class RecvRCInfoEventArgs:EventArgs

{

누가 요청했는지 알 수 있게 IPEndPoint 속성을 제공합시다.

public IPEndPoint IPEndPoint { get; private set; }

IP 주소를 문자열 형태로 변환하여 제공하고 포트 번호도 속성으로 제공합시다.

public string IPAddressStr

{

get { return IPEndPoint.Address.ToString(); }

}

public int Port

{

get { return IPEndPoint.Port; }

}

생성자에서는 EndPoint를 입력 인자로 받아 속성에 설정합니다.

internal RecvRCInfoEventArgs(EndPoint RemoteEndPoint)

{

IPEndPoint = RemoteEndPoint as IPEndPoint;

}

}

이벤트 처리를 위해 대리자 형식을 정의합시다.

public delegate void RecvRCInfoEventHandler(object sender, RecvRCInfoEventArgs e);

원격 제어 요청을 수락하거나 거절하는 SetupServer 클래스도 정적 클래스로 정의합시다.

public static class SetupServer

{

서버 측은 연결 요청을 수신하기 위한 Listening 소켓을 생성하는 부분과 연결 요청을 대기하고 수용하는 부분으로 나눌 수 있습니다. 특히 연결 요청을 대기하고 수용하는 부분은 무한 반복할 것으로 블로킹을 막기 위해 여기에서는 스레드를 사용할게요.

Listening 소켓을 멤버로 선언합시다.

static Socket lis\_sock;

무한 대기하는 부분의 스레드를 멤버로 선언합시다.

static Thread accept\_thread = null;

연결 요청이 왔을 때 이벤트 처리를 위해 RecvRCInfoEventHandler 형식의 이벤트를 멤버로 선언하세요.

static public event RecvRCInfoEventHandler RecvedRCInfo = null;

Setup서버를 시작하는 메서드에서도 IP 주소와 포트 번호를 입력 인자로 받습니다.

static public void Start(string ip, int port)

{

먼저 TCP 소켓을 생성합니다.

IPAddress ipaddr = IPAddress.Parse(ip);

IPEndPoint ep = new IPEndPoint(ipaddr, port);

lis\_sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

네트워크 인터페이스와 결합하고 Back 로그 큐를 설정합니다.

lis\_sock.Bind(ep);

lis\_sock.Listen(1);

연결 요청을 대기하고 수용하는 스레드를 생성하고 시작합니다.

ThreadStart ts = new ThreadStart(AcceptLoop);

accept\_thread = new Thread(ts);

accept\_thread.Start();

}

static void AcceptLoop()

{

try

{

연결 요청을 대기하고 수용하는 부분 Listen 소켓의 Accept 메서드를 호출는 것을 반복합니다.

while (true)

{

Socket do\_sock = lis\_sock.Accept();

if (RecvedRCInfo != null)

{

연결 요청을 수신하는 이벤트 핸들러가 있다면 이벤트 인자를 생성하여 이벤트를 실행합니다.

RecvRCInfoEventArgs e = new RecvRCInfoEventArgs(do\_sock.RemoteEndPoint);

RecvedRCInfo(null, e);

}

원격 제어를 요청한 이가 누구인지 확인하고 바로 연결을 닫습니다.

do\_sock.Close();

}

catch { Close(); }

}

SetupServer를 종료할 수 있게 Close 메서드를 정의합시다.

public static void Close()

{

대기 스레드와 소켓을 해제화 작업을 수행합니다.

if (accept\_thread != null){ accept\_thread = null; }

if (lis\_sock != null){ lis\_sock.Close(); lis\_sock = null; }

}

|  |
| --- |
| using System.Net;  using System.Net.Sockets;  namespace 원격제어기  {  public static class SetupClient  {  public static void Setup(string ip, int port)  {  IPAddress ipaddr = IPAddress.Parse(ip);  IPEndPoint ep = new IPEndPoint(ipaddr, port);  Socket sock = new Socket(AddressFamily.InterNetwork,  SocketType.Stream, ProtocolType.Tcp);  sock.Connect(ep);  sock.Close();  }  }  } |

[소스 9.1] SetupClient.cs

|  |
| --- |
| using System;  using System.Net;  namespace 원격제어기  {  public class RecvRCInfoEventArgs:EventArgs  {  public IPEndPoint IPEndPoint  {  get;  private set;  }  public string IPAddressStr  {  get  {  return IPEndPoint.Address.ToString();  }  }  public int Port  {  get  {  return IPEndPoint.Port;  }  }  internal RecvRCInfoEventArgs(EndPoint RemoteEndPoint)  {  IPEndPoint = RemoteEndPoint as IPEndPoint;  }  }  public delegate void RecvRCInfoEventHandler(object sender, RecvRCInfoEventArgs e);  } |

[소스 9.2] RecvRCInfoEventArgs.cs

|  |
| --- |
| using System.Net.Sockets;  using System.Threading;  using System.Net;  namespace 원격제어기  {  public static class SetupServer  {  static Socket lis\_sock;  static Thread accept\_thread = null;  static public event RecvRCInfoEventHandler RecvedRCInfo = null;  static public void Start(string ip, int port)  {  IPAddress ipaddr = IPAddress.Parse(ip);  IPEndPoint ep = new IPEndPoint(ipaddr, port);  lis\_sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,  ProtocolType.Tcp);  lis\_sock.Bind(ep);  lis\_sock.Listen(1);  ThreadStart ts = new ThreadStart(AcceptLoop);  accept\_thread = new Thread(ts);  accept\_thread.Start();  }  static void AcceptLoop()  {  try  {  while (true)  {  Socket do\_sock = lis\_sock.Accept();  if (RecvedRCInfo != null)  {  RecvRCInfoEventArgs e = new RecvRCInfoEventArgs(  do\_sock.RemoteEndPoint);  RecvedRCInfo(null, e);  }  do\_sock.Close();  }  }  catch  {  Close();  }  }  public static void Close()  {  if (accept\_thread != null)  {  accept\_thread = null;  }  if (lis\_sock != null)  {  lis\_sock.Close();  lis\_sock = null;  }  }  }  } |

[소스 9.3] SetupServer.cs

* 1. 제어 화면 송수신

이번에는 원격 제어를 허용한 호스트의 전체 화면을 제어하는 호스트에게 주기적으로 전송하고 수신하는 부분을 작성합시다.

먼저 이미지를 전송하는 ImageClient 클래스를 작성합시다.

public class ImageClient

{

전송에 사용할 소켓을 멤버 필드로 선언하세요.

Socket sock;

생성자에서는 상대 IP와 포트 번호를 입력 인자로 받습니다.

public ImageClient(string ip, int port)

{

입력 인자로 받은 정보로 IPEndPoint 개체를 생성합니다.

IPAddress ipaddr = IPAddress.Parse(ip);

IPEndPoint ep = new IPEndPoint(ipaddr, port);

그리고 TCP 소켓을 생성하고 상대 호스트 EndPoint에 연결 요청합니다.

sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

sock.Connect(ep);

}

이미지를 전송하는 메서드를 정의합시다.

public bool SendImage(Image img)

{

만약 소켓이 null이면 메서드를 종료합니다.

if (sock == null) { return false; }

이미지를 선형적으로 보내기 위해 메모리 스트림 개체를 생성합니다.

MemoryStream ms = new MemoryStream();

이미지를 메모리 스트림에 저장합니다.

img.Save(ms, ImageFormat.Jpeg);

그리고 메모리 스트림에 이미지를 저장한 버퍼를 얻어옵니다.

byte[] data = ms.GetBuffer();

try

{

소켓으로 전송할 이미지 버퍼의 길이를 byte 배열 개체로 만들어서 전송합니다.

byte[] lbuf = BitConverter.GetBytes(data.Length);

sock.Send(lbuf);

그리고 이미지를 전송합니다.

int trans = 0;

while (trans < data.Length)

{

trans += sock.Send(data,trans,data.Length-trans,SocketFlags.None);

}

sock.Close();

return true;

전송이 끝나면 소켓을 닫습니다.

sock.Close();

return true;

}

예외가 발생하면 응용을 끝냅니다.

catch{ Application.Exit(); return false; }

}

비동기로 이미지를 전송하는 메서드도 제공합시다. 이 부분은 대리자를 이용할게요.

public delegate bool SendImageDele(Image img);

public void SendImageAsync(Image img, AsyncCallback callback)

{

SendImage 메서드를 인자로 대리자를 생성하여 대리자를 비동기로 수행합니다.

SendImageDele dele = new SendImageDele(SendImage);

dele.BeginInvoke(img, callback, this);

}

이미지 전송 클라이언트를 닫는 메서드도 제공합시다.

public void Close()

{

sock.Close();

sock = null;

}

}

이미지 수신 서버를 구현합시다. 이미지 수신 서버에서는 이미지를 수신할 때마다 폼에 이를 알려주어야 합니다. 이 부분을 위해 이벤트 처리를 할 것입니다. 먼저 이미지를 수신하였을 때 이벤트를 처리하기 위해 대리자와 이벤트 인자 클래스를 정의합시다. RecvImageEventArgs 이름의 클래스를 추가하세요.

public class RecvImageEventArgs : EventArgs

{

이벤트 인자로 상대측 IPEndPoint를 가져오기 할 수 있는 속성을 제공합시다.

public IPEndPoint IPEndPoint

{

get; private set;

}

상대측 IP 주소 개체와 IP 주소 문자열과 포트 정보를 가져오기 할 수 있는 속성도 제공합시다.

public IPAddress IPAddress

{

get{ return IPEndPoint.Address; }

}

public string IPAddressStr

{

get{ return IPAddress.ToString(); }

}

public int Port

{

get{ return IPEndPoint.Port; }

}

이미지 개체와 이미지 크기에 관한 정보도 가져오기 할 수 있는 속성을 제공합시다.

public Image Image

{

get;

private set;

}

public Size Size

{

get{ return Image.Size; }

}

public int Width

{

get{ return Image.Width; }

}

public int Height

{

get{ return Image.Height; }

}

생성자에서는 상대측 IPEndPoint 정보와 이미지 개체를 입력 인자로 받아 속성을 설정합니다.

internal RecvImageEventArgs(IPEndPoint remote\_iep, Image image)

{

IPEndPoint = remote\_iep;

Image = image;

}

ToString 메서드를 재정의하여 상대측 IP 정보와 이미지의 폭과 높이를 문자열로 형성하여 제공합시다.

public override string ToString()

{

return string.Format("IP:{0} width:{1} height:{1}", IPAddressStr, Width, Height);

}

}

이미지를 수신할 때 처리를 위한 대리자를 정의합시다.

public delegate void RecvImageEventHandler(object sender, RecvImageEventArgs e);

이제 이미지 서버 클래스를 정의합시다.

public class ImageServer

{

먼저 Listen 소켓과 연결 요청을 대기하고 수용하는 부분을 위한 스레드를 멤버로 선언합시다.

Socket lis\_sock;

Thread accept\_thread = null;

이미지를 수신할 때 이벤트 처리를 위해 RecvImageEventHandler 형식의 이벤트를 선언합시다.

public event RecvImageEventHandler RecvedImage = null;

생성자에서는 IP와 포트 정보를 받습니다.

public ImageServer(string ip, int port)

{

먼저 전달받은 정보로 IPEndPoint 개체를 생성합니다.

IPAddress ipaddr = IPAddress.Parse(ip);

IPEndPoint ep = new IPEndPoint(ipaddr, port);

그리고 TCP 소켓을 생성하여 네트워크 인터페이스와 결합하고 백로그 큐를 설정합니다.

lis\_sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

lis\_sock.Bind(ep);

lis\_sock.Listen(1);

클라이언트 연결 요청을 대기하고 수락하는 스레드를 생성하고 시작합니다.

ThreadStart ts = new ThreadStart(AcceptLoop);

accept\_thread = new Thread(ts);

accept\_thread.Start();

}

void AcceptLoop()

{

try

{

클라이언트 연결 요청을 대기하고 수락하는 부분은 무한 반복합니다.

while (true)

{

Listen 소켓의 Accept 메서드를 호출합니다.

Socket do\_sock = lis\_sock.Accept();

이미지를 수신합니다. 이 부분은 별도의 메서드로 정의합시다.

Receive(do\_sock);

}

}

catch{ Close(); }

}

void Receive(Socket dosock)

{

먼저 수신할 이미지 길이를 수신합니다.

byte[] lbuf = new byte[4];

dosock.Receive(lbuf);

int len = BitConverter.ToInt32(lbuf, 0);

그리고 이미지를 수신합니다.

byte[] buffer = new byte[len];

int trans=0;

while (trans < len)

{

trans += dosock.Receive(buffer,trans,len-trans,SocketFlags.None);

}

만약 이미지 수신 이벤트 핸들러가 있으면 이벤트 인자를 생성하여 이벤트를 전송합니다. 이벤트 인자를 생성하기 위해 수신한 byte 배열을 이미지로 변환하는 부분은 별도의 메서드를 만들기로 합시다.

if (RecvedImage != null)

{

IPEndPoint iep = dosock.RemoteEndPoint as IPEndPoint;

RecvImageEventArgs e = new RecvImageEventArgs(iep, ConvertBitmap(buffer));

RecvedImage(this, e);

}

dosock.Close();

}

public Bitmap ConvertBitmap(byte[] data)

{

byte 배열을 이미지로 변환하기 위해 메모리 스트림 개체를 생성합니다.

MemoryStream ms = new MemoryStream();

메모리 스트림 개체에 수신한 이미지가 있는 버퍼의 내용을 기록합니다.

ms.Write(data, 0, (int)data.Length);

메모리 스트림 개체를 입력 인자로 Bitmap 개체를 생성하여 반환합니다.

Bitmap bitmap = new Bitmap(ms);

return bitmap;

}

이미지 서버를 닫는 메서드도 제공합시다.

public void Close()

{

스레드와 Listen 소켓을 해제하는 작업을 수행합니다.

if (accept\_thread != null){ accept\_thread = null; }

if (lis\_sock != null){ lis\_sock.Close(); lis\_sock = null; }

}

|  |
| --- |
| using System;  using System.Net.Sockets;  using System.Net;  using System.IO;  using System.Drawing.Imaging;  using System.Windows.Forms;  using System.Drawing;  namespace 원격제어기  {  public class ImageClient  {  Socket sock;  public ImageClient(string ip, int port)  {  IPAddress ipaddr = IPAddress.Parse(ip);  IPEndPoint ep = new IPEndPoint(ipaddr, port);  sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,  ProtocolType.Tcp);  sock.Connect(ep);  }  public bool SendImage(Image img)  {  if (sock == null)  {  return false;  }  MemoryStream ms = new MemoryStream();  img.Save(ms, ImageFormat.Jpeg);  byte[] data = ms.GetBuffer();  try  {  int trans = 0;  byte[] lbuf = BitConverter.GetBytes(data.Length);  sock.Send(lbuf);  while (trans < data.Length)  {  trans += sock.Send(data,trans,data.Length-trans,SocketFlags.None);  }  sock.Close();  sock = null;  return true;  }  catch  {  Application.Exit();  return false;  }  }  public delegate bool SendImageDele(Image img);  public void SendImageAsync(Image img, AsyncCallback callback)  {  SendImageDele dele = new SendImageDele(SendImage);  dele.BeginInvoke(img, callback, this);  }  public void Close()  {  if (sock != null)  {  sock.Close();  sock = null;  }  }  }  } |

[소스 9.4] ImageClient.cs

|  |
| --- |
| using System;  using System.Net;  using System.Drawing;  namespace 원격제어기  {  public class RecvImageEventArgs : EventArgs  {  public IPEndPoint IPEndPoint  {  get;  private set;  }  public IPAddress IPAddress  {  get  {  return IPEndPoint.Address;  }  }  public string IPAddressStr  {  get  {  return IPAddress.ToString();  }  }  public int Port  {  get  {  return IPEndPoint.Port;  }  }  public Image Image  {  get;  private set;  }  public Size Size  {  get  {  return Image.Size;  }  }  public int Width  {  get  {  return Image.Width;  }  }  public int Height  {  get  {  return Image.Height;  }  }  internal RecvImageEventArgs(IPEndPoint remote\_iep, Image image)  {  IPEndPoint = remote\_iep;  Image = image;  }  public override string ToString()  {  return string.Format("IP:{0} width:{1} height:{1}", IPAddressStr, Width, Height);  }  }  public delegate void RecvImageEventHandler(object sender, RecvImageEventArgs e);  } |

[소스 9.5] RecvImageEventArgs.cs

|  |
| --- |
| using System.Net.Sockets;  using System.Threading;  using System.Net;  using System.Drawing;  using System.IO;  using System;  namespace 원격제어기  {  public class ImageServer  {  Socket lis\_sock;  Thread accept\_thread = null;  public event RecvImageEventHandler RecvedImage = null;  public ImageServer(string ip, int port)  {  IPAddress ipaddr = IPAddress.Parse(ip);  IPEndPoint ep = new IPEndPoint(ipaddr, port);  lis\_sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,  ProtocolType.Tcp);  lis\_sock.Bind(ep);  lis\_sock.Listen(100);  ThreadStart ts = new ThreadStart(AcceptLoop);  accept\_thread = new Thread(ts);  accept\_thread.Start();  }  void AcceptLoop()  {  try  {  while (lis\_sock != null)  {  Socket do\_sock = lis\_sock.Accept();  Receive(do\_sock);  }  }  catch{ Close(); }  }  void Receive(Socket dosock)  {  byte[] lbuf = new byte[4];  dosock.Receive(lbuf);  int len = BitConverter.ToInt32(lbuf, 0);  byte[] buffer = new byte[len];  int trans=0;  while (trans < len)  {  trans += dosock.Receive(buffer,trans,len-trans,SocketFlags.None);  }  if (RecvedImage != null)  {  IPEndPoint iep = dosock.RemoteEndPoint as IPEndPoint;  RecvImageEventArgs e = new RecvImageEventArgs(iep, ConvertBitmap(buffer));  RecvedImage(this, e);  }  dosock.Close();  }  public Bitmap ConvertBitmap(byte[] data)  {  MemoryStream ms = new MemoryStream();  ms.Write(data, 0, (int)data.Length);  Bitmap bitmap = new Bitmap(ms);  return bitmap;  }  public void Close()  {  if (accept\_thread != null){ accept\_thread = null; }  if (lis\_sock != null){ lis\_sock.Close(); lis\_sock = null; }  }  }  } |

[소스 9.6] ImageServer.cs

* 1. 키보드, 마우스 이벤트 송수신

이번에는 제어하는 호스트의 키보드와 마우스 이벤트를 전송하고 제어당하는 호스트에서 이를 수신하는 부분을 구현합시다.

먼저 제어하는 호스트의 키보드와 마우스 이벤트를 전송하는 부분을 구현합시다. 먼저 프로젝트에 SendEventClient 클래스를 추가하세요.

전송하고 수신할 메시지 종류를 열거형으로 정의합시다.

public enum MsgType

{

MT\_KDOWN = 1, MT\_KEYUP, MT\_M\_LEFTDOWN,

MT\_M\_LEFTUP, MT\_M\_RIGHTUP, MT\_M\_RIGHTDOWN,

MT\_M\_MIDDLEDOWN, MT\_M\_MIDDLEUP, MT\_M\_MOVE

}

public class SendEventClient

{

상대측 IPEndPoint를 기억하는 멤버를 선언하세요.

IPEndPoint ep;

생성자에서는 상대측 IP와 포트를 입력인자로 받아서 IPEndPoint 개체를 생성합니다.

public SendEventClient(string ip, int port)

{

ep = new IPEndPoint(IPAddress.Parse(ip), port);

}

키를 누른 메시지를 전송하는 메서드를 구현합시다.

public void SendKeyDown(int key)

{

먼저 전송할 메시지 종류를 버퍼에 기록합니다.

byte[] data = new byte[9];

data[0] = (byte)MsgType.MT\_KDOWN;

그리고 누를 키를 byte 배열에 설정합니다. 맨 앞의 원소는 메시지 종류이므로 인덱스 1에서 4바이트에 설정해야 합니다. int 형식의 값을 byte 배열로 변환하여 설정합니다.

Array.Copy(BitConverter.GetBytes(key), 0, data, 1, 4);

이제 버퍼를 전송합니다. 버퍼를 전송하는 부분은 별도의 메서드 SendData를 정의합시다.

SendData(data);

}

private void SendData(byte[] data)

{

먼저 TCP 소켓을 생성하여 연결합니다.

Socket sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,

ProtocolType.Tcp);

sock.Connect(ep);

입력 인자로 받은 버퍼를 전송하고 연결을 끊습니다.

sock.Send(data);

sock.Close();

}

키를 뗀 메시지를 전송하는 메서드도 정의합시다. 구현 방법은 SendKeyDown과 비슷합니다.

public void SendKeyUp(int key)

{

먼저 메시지 종류를 버퍼의 맨 앞에 설정합니다.

byte[] data = new byte[9];

data[0] = (byte)MsgType.MT\_KEYUP;

그리고 키를 버퍼에 추가하여 전송합니다.

Array.Copy(BitConverter.GetBytes(key), 0, data, 1, 4);

SendData(data);

}

마우스를 눌렀을 때의 메시지를 전송하는 메서드를 정의합시다.

public void SendMouseDown(MouseButtons mouseButtons)

{

버튼 종류에 따라 메시지 종류를 설정합니다.

byte[] data = new byte[9];

switch (mouseButtons)

{

case MouseButtons.Left: data[0] = (byte)MsgType.MT\_M\_LEFTDOWN; break;

case MouseButtons.Right: data[0] = (byte)MsgType.MT\_M\_RIGHTDOWN; break;

case MouseButtons.Middle data[0] = (byte)MsgType.MT\_M\_MIDDLEDOWN; break;

}

그리고 data를 전송합니다.

SendData(data);

}

마우스를 떼었을 때의 메시지를 전송하는 메서드를 정의합시다.

public void SendMouseUp(MouseButtons mouseButtons)

{

버튼 종류에 따라 메시지 종류를 설정한 후에 메시지를 전송합니다.

byte[] data = new byte[9];

switch (mouseButtons)

{

caseMouseButtons.Left: data[0]=(byte)MsgType.MT\_M\_LEFTUP; break;

case MouseButtons.Right: data[0] = (byte)MsgType.MT\_M\_RIGHTUP; break;

case MouseButtons.Middle: data[0] = (byte)MsgType.MT\_M\_MIDDLEUP; break;

}

SendData(data);

}

마우스 이동 메시지를 전송하는 메서드를 정의합시다.

public void SendMouseMove(int x, int y)

{

메시지 종류를 설정하고 좌표를 추가한 후에 메시지를 전송합니다.

byte[] data = new byte[9];

data[0] = (byte)MsgType.MT\_M\_MOVE;

Array.Copy(BitConverter.GetBytes(x), 0, data, 1, 4);

Array.Copy(BitConverter.GetBytes(y), 0, data, 5, 4);

SendData(data);

}

}

메시지를 수신하는 서버에서는 수신한 버퍼의 내용을 분석하는 부분이 필요합니다. 여기에서는 Meta 클래스를 정의하여 분석한 정보를 표현합시다.

public class Meta

{

수신한 메시지 종류를 속성으로 제공합시다.

public MsgType Mt{ get; private set; }

누르거나 뗀 키를 속성으로 제공합시다.

public int Key{ get; private set; }

현재 마우스 좌표를 속성으로 제공합시다.

public Point Now{ get; private set; }

생성자는 수신한 버퍼를 입력 인자로 받습니다.

public Meta(byte[] data)

{

버퍼의 맨 앞에는 메시지 종류가 있습니다.

Mt = (MsgType)data[0];

메시지 종류에 따라 키에 관한 메시지와 마우스 이동에 관한 메시지를 분석합니다. 마우스를 누르거나 떼었을 때는 메시지 종류 외에 다른 부가적인 메시지가 없습니다.

switch (Mt)

{

case MsgType.MT\_KDOWN:

case MsgType.MT\_KEYUP: MakingKey(data); break;

case MsgType.MT\_M\_MOVE: MakingPoint(data); break;

}

}

마우스 이동일 때 현재 좌표를 분석하는 메서드를 정의합시다.

private void MakingPoint(byte[] data)

{

X 좌표는 버퍼 인덱스 1~4, Y 좌표는 버퍼 인덱스 5~8에 있습니다.

Point now = new Point(0, 0);

now.X = (data[4] << 24) + (data[3] << 16) + (data[2] << 8) + (data[1]);

now.Y = (data[8] << 24) + (data[7] << 16) + (data[6] << 8) + (data[5]);

Now = now;

}

키보드를 누르거나 떼었을 때의 키를 분석하는 메서드를 정의합시다.

private void MakingKey(byte[] data)

{

키는 버퍼 인덱스 1~4에 있습니다.

Key = (data[4] << 24) + (data[3] << 16) + (data[2] << 8) + (data[1]);

}

}

키보드와 마우스 이벤트 발생 메시지를 수신하였을 때 이를 통보하기 위해 이벤트 방식을 사용할 것입니다. 이를 위해 RecvMEEventArgs 클래스를 추가하세요.

public class RecvKMEEventArgs : EventArgs

{

수신한 메시지를 분석한 Meta 개체를 속성으로 제공합시다.

public Meta Meta{ get; private set; }

사용하기 쉽게 키와 좌표, 메시지 종류도 속성으로 제공합니다.

public int Key

{

get{ return Meta.Key; }

}

public Point Now

{

get{ return Meta.Now; }

}

public MsgType MT

{

get{ return Meta.Mt; }

}

생성자에서는 분석한 Meta 개체를 입력 인자로 받아 속성을 설정합니다.

internal RecvKMEEventArgs(Meta meta)

{

Meta = meta;

}

}

이벤트 형식에 사용할 대리자를 정의합시다.

public delegate void RecvKMEEventHandler(object sender, RecvKMEEventArgs e);

이제 RecvEventServer 클래스를 정의합시다.

public class RecvEventServer

{

서버의 Listen 소켓을 멤버로 선언합니다.

Socket lis\_sock;

메시지를 수신하였을 때 처리를 위해 이벤트를 멤버로 선언합니다.

public event RecvKMEEventHandler RecvedKMEvent = null;

클라이언트의 연결 요청과 수용을 위한 스레드를 멤버로 선언합시다.

Thread th;

생성자 메서드는 자신의 IP와 포트를 입력 인자로 받습니다.

public RecvEventServer(string ip, int port)

{

입력 인자로 받은 정보로 IPEndPoint 개체를 생성하여 TCP 소켓을 생성, 네트워크 인터페이스와 결합 및 Back 로그 큐를 설정합니다.

IPAddress ipaddr = IPAddress.Parse(ip);

IPEndPoint ep = new IPEndPoint(ipaddr, port);

lis\_sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

lis\_sock.Bind(ep);

lis\_sock.Listen(5);

클라이언트 연결 요청을 수용을 위한 스레드를 생성 및 가동합니다.

ThreadStart ts = new ThreadStart(AcceptLoop);

th = new Thread(ts);

th.Start();

}

클라이언트 연결 요청 및 수용하는 메서드를 구현합시다. 메서드 내에서 메시지를 수신하는 부분을 비동기 처리를 위해 대리자를 정의할게요.

public delegate void ReceiveLoopDele(Socket dosock);

void AcceptLoop()

{

Socket do\_sock;

메시지를 수순하는 부분을 처리할 대리자 개체를 생성합니다.

ReceiveLoopDele rld = new ReceiveLoopDele(ReceiveLoop);

try

{

클라이언트 연결 요청 및 수용하는 부분은 무한 반복합니다.

while (true)

{

ListenSocket의 Accept 메서드를 호출합니다.

do\_sock = lis\_sock.Accept();

그리고 메시지를 수신하는 대리자를 수행합니다.

rld.BeginInvoke(do\_sock, null, null);

}

}

catch

{

예외가 발생하면 Close 메서드를 호출하여 해제화합니다.

Close();

}

}

메시지를 수신하는 메서드를 구현합시다.

void ReceiveLoop(Socket dosock)

{

먼저 크기가 9인 버퍼를 생성하여 메시지를 수신합니다. 클라이언트 측에서는 전송할 메시지 종류에 관계없이 고정길이 버퍼에 데이터를 설정하여 보내게 구현하였습니다.

byte[] buffer = new byte[9];

int n = dosock.Receive(buffer);

메시지 수신 이벤트 핸들러가 있으면 이벤트를 발생합니다.

if (RecvedKMEvent != null)

{

RecvedKMEvent(this, new RecvKMEEventArgs(new Meta(buffer)));

}

소켓을 닫습니다.

dosock.Close();

}

메시지 수신 서버를 닫는 메서드를 제공합시다.

public void Close()

{

스레드와 Listen 소켓을 해제합니다.

if (th != null){ th = null; }

if (lis\_sock != null){ lis\_sock.Close(); lis\_sock = null; }

}

}

|  |
| --- |
| using System;  using System.Net;  using System.Net.Sockets;  using System.Windows.Forms;  namespace 원격제어기  {  public enum MsgType  {  MT\_KDOWN = 1, MT\_KEYUP, MT\_M\_LEFTDOWN,  MT\_M\_LEFTUP, MT\_M\_RIGHTUP, MT\_M\_RIGHTDOWN,  MT\_M\_MIDDLEDOWN, MT\_M\_MIDDLEUP, MT\_M\_MOVE  }  public class SendEventClient  {  IPEndPoint ep;  public SendEventClient(string ip, int port)  {  ep = new IPEndPoint(IPAddress.Parse(ip), port);  }  public void SendKeyDown(int key)  {  byte[] data = new byte[9];  data[0] = (byte)MsgType.MT\_KDOWN;  Array.Copy(BitConverter.GetBytes(key), 0, data, 1, 4);  SendData(data);  }  private void SendData(byte[] data)  {  Socket sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,  ProtocolType.Tcp);  sock.Connect(ep);  sock.Send(data);  sock.Close();  }  public void SendKeyUp(int key)  {  byte[] data = new byte[9];  data[0] = (byte)MsgType.MT\_KEYUP;  Array.Copy(BitConverter.GetBytes(key), 0, data, 1, 4);  SendData(data);  }  public void SendMouseDown(MouseButtons mouseButtons)  {  byte[] data = new byte[9];  switch (mouseButtons)  {  case MouseButtons.Left: data[0] = (byte)MsgType.MT\_M\_LEFTDOWN; break;  case MouseButtons.Right: data[0] = (byte)MsgType.MT\_M\_RIGHTDOWN; break;  case MouseButtons.Middle:data[0]=(byte)MsgType.MT\_M\_MIDDLEDOWN;break;  }  SendData(data);  }  public void SendMouseUp(MouseButtons mouseButtons)  {  byte[] data = new byte[9];  switch (mouseButtons)  {  case MouseButtons.Left: data[0] = (byte)MsgType.MT\_M\_LEFTUP; break;  case MouseButtons.Right: data[0] = (byte)MsgType.MT\_M\_RIGHTUP; break;  case MouseButtons.Middle: data[0] = (byte)MsgType.MT\_M\_MIDDLEUP; break;  }  SendData(data);  }  public void SendMouseMove(int x, int y)  {  byte[] data = new byte[9];  data[0] = (byte)MsgType.MT\_M\_MOVE;  Array.Copy(BitConverter.GetBytes(x), 0, data, 1, 4);  Array.Copy(BitConverter.GetBytes(y), 0, data, 5, 4);  SendData(data);  }  }  } |

[소스 9.7] SendEventClient.cs

|  |
| --- |
| using System.Drawing;  namespace 원격제어기  {  public class Meta  {  public MsgType Mt{ get; private set; }  public int Key{ get; private set; }  public Point Now{ get; private set; }  public Meta(byte[] data)  {  Mt = (MsgType)data[0];  switch (Mt)  {  case MsgType.MT\_KDOWN:  case MsgType.MT\_KEYUP:  MakingKey(data); break;  case MsgType.MT\_M\_MOVE:  MakingPoint(data); break;  }  }  private void MakingPoint(byte[] data)  {  Point now = new Point(0, 0);  now.X = (data[4] << 24) + (data[3] << 16) + (data[2] << 8) + (data[1]);  now.Y = (data[8] << 24) + (data[7] << 16) + (data[6] << 8) + (data[5]);  Now = now;  }  private void MakingKey(byte[] data)  {  Key = (data[4] << 24) + (data[3] << 16) + (data[2] << 8) + (data[1]);  }  }  } |

[소스 9.8] Meta.cs

|  |
| --- |
| using System;  using System.Drawing;  namespace 원격제어기  {  public class RecvKMEEventArgs : EventArgs  {  public Meta Meta{ get; private set; }  public int Key  {  get  {  return Meta.Key;  }  }  public Point Now  {  get  {  return Meta.Now;  }  }  public MsgType MT  {  get  {  return Meta.Mt;  }  }  internal RecvKMEEventArgs(Meta meta)  {  Meta = meta;  }  }  public delegate void RecvKMEEventHandler(object sender, RecvKMEEventArgs e);  } |

[소스 9.9] RecvKMEEventArgs.cs

|  |
| --- |
| using System.Net.Sockets;  using System.Threading;  using System.Net;  namespace 원격제어기  {  public class RecvEventServer  {  Socket lis\_sock;  public event RecvKMEEventHandler RecvedKMEvent = null;  Thread th;  public RecvEventServer(string ip, int port)  {  IPAddress ipaddr = IPAddress.Parse(ip);  IPEndPoint ep = new IPEndPoint(ipaddr, port);  lis\_sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,  ProtocolType.Tcp);  lis\_sock.Bind(ep);  lis\_sock.Listen(5);  ThreadStart ts = new ThreadStart(AcceptLoop);  th = new Thread(ts);  th.Start();  }  public delegate void ReceiveLoopDele(Socket dosock);  void AcceptLoop()  {  Socket do\_sock;  ReceiveLoopDele rld = new ReceiveLoopDele(ReceiveLoop);  try  {  while (true)  {  do\_sock = lis\_sock.Accept();  rld.BeginInvoke(do\_sock, null, null);  }  }  catch{ Close(); }  }  void ReceiveLoop(Socket dosock)  {  byte[] buffer = new byte[9];  int n = dosock.Receive(buffer);  if (RecvedKMEvent != null)  {  RecvedKMEvent(this, new RecvKMEEventArgs(new Meta(buffer)));  }  dosock.Close();  }  public void Close()  {  if (th != null)  {  th = null;  }  if (lis\_sock != null)  {  lis\_sock.Close();  lis\_sock = null;  }  }  }  } |

[소스 9.10] RecvEventServer.cs

* 1. 원격 제어 컨트롤러

이번에는 원격 제어하는 컨트롤러를 만들어 봅시다.

public class Controller

{

원격 제어 컨트롤러는 단일체 패턴으로 정의할 것입니다. 이를 위해 정적 개체를 위한 정적 멤버 필드를 선언하세요.

static Controller singleton;

단일체 개체를 참조할 수 있는 정적 속성을 제공합니다.

public static Controller Singleton

{

get{ return singleton; }

}

정적 생성자에서 단일체를 생성합니다.

static Controller()

{

singleton = new Controller();

}

그리고 기본 생성자는 private으로 접근 설정하여 다른 곳에서 개체를 생성할 수 없게 합니다.

private Controller()

{

}

이미지는 수신하므로 ImageServer 개체를 위한 멤버를 선언하세요.

ImageServer img\_sever = null;

키보드와 마우스 이벤트는 전송하므로 SendEventClient 개체를 위한 멤버를 선언하세요.

SendEventClient sce = null;

이미지를 수신하였을 때 처리를 위한 이벤트를 멤버로 선언합시다.

public event RecvImageEventHandler RecvedImage = null;

상대측 IP를 기억하는 멤버를 선언합시다.

string host\_ip;

SendEventClient 개체를 가져오기 할 수 있는 속성을 제공합니다.

public SendEventClient SendEventClient

{

get{ return sce; }

}

자신의 IP 주소를 구하는 속성을 제공합시다.

public string MyIP

{

get

{

먼저 호스트 이름을 얻어오고 호스트 목록 개체를 구합니다.

string host\_name = Dns.GetHostName();

IPHostEntry host\_entry = Dns.GetHostEntry(host\_name);

호스트 목록 개체의 주소 목록의 각 개체를 조사하여 주소 패밀리가 IPv4인 것을 찾습니다.

foreach (IPAddress ipaddr in host\_entry.AddressList)

{

if(ipaddr.AddressFamily==AddressFamily.InterNetwork){ return ipaddr.ToString(); }

}

없으면 빈 문자열을 반환합니다.

return string.Empty;

}

}

원격 제어 컨트롤로를 가동하는 메서드를 제공합시다.

public void Start(string host\_ip)

{

입력 인자로 받은 상대측 IP를 멤버에 설정합니다.

this.host\_ip = host\_ip;

이미지 서버를 가동합니다. 여기에서는 20004 포트로 하드 코딩할게요.

img\_sever = new ImageServer(MyIP, 20004);

이미지를 수신하였을 때 처리할 이벤트 핸들러를 등록합니다.

img\_sever.RecvedImage += new RecvImageEventHandler(img\_sever\_RecvedImage);

SetupClient를 설정합니다. 여기에서도 포트 번호는 20002로 하드 코딩할게요.

SetupClient.Setup(host\_ip, 20002);

}

void img\_sever\_RecvedImage(object sender, RecvImageEventArgs e)

{

이미지를 수신한 이벤트는 구독자에게 By Pass 합니다.

if (RecvedImage != null)

{

RecvedImage(this, e);

}

}

이벤트 전송 클라이언트를 설정하는 메서드를 추가합니다.

public void StartEventClient()

{

SendEventClient 개체를 생성합니다. 포트 번호는 20010으로 하드 코딩할게요.

sce = new SendEventClient(host\_ip, 20010);

}

서버의 가동을 멈추는 메서드를 제공합니다.

public void Stop()

{

if (img\_sever != null)

{

이미지 서버의 Close 메서드를 호출하여 해제화합니다.

img\_sever.Close();  
 그리고 img\_server를 null로 설정합니다.

img\_sever = null;

}

}

}

|  |
| --- |
| using System.Net;  using System.Net.Sockets;  namespace 원격제어기  {  public class Controller  {  static Controller singleton;  public static Controller Singleton  {  get{ return singleton; }  }  static Controller(){ singleton = new Controller(); }  private Controller(){ }  ImageServer img\_sever = null;  SendEventClient sce = null;  public event RecvImageEventHandler RecvedImage = null;  string host\_ip;  public SendEventClient SendEventClient  {  get{ return sce; }  }  public string MyIP  {  get  {  string host\_name = Dns.GetHostName();  IPHostEntry host\_entry = Dns.GetHostEntry(host\_name);  foreach (IPAddress ipaddr in host\_entry.AddressList)  {  if (ipaddr.AddressFamily == AddressFamily.InterNetwork)  {  return ipaddr.ToString();  }  }  return string.Empty;  }  }  public void Start(string host\_ip)  {  this.host\_ip = host\_ip;  img\_sever = new ImageServer(MyIP, 20004);  img\_sever.RecvedImage += new RecvImageEventHandler(img\_sever\_RecvedImage);  SetupClient.Setup(host\_ip, 20002);  }  void img\_sever\_RecvedImage(object sender, RecvImageEventArgs e)  {  if (RecvedImage != null)  {  RecvedImage(this, e);  }  }  public void StartEventClient()  {  sce = new SendEventClient(host\_ip, 20010);  }  public void Stop()  {  if (img\_sever != null)  {  img\_sever.Close();  img\_sever = null;  }  }  }  } |

[소스 9.11] Controller.cs

* 1. 원격 제어 대상 호스트

이번에는 원격 제어 대상 호스트를 구현합시다.

먼저 키보드와 마우스 이벤트를 래핑한 클래스를 정의합니다. 이 부분은 앞에서 만들었던 것과 대부분 비슷하므로 설명하지 않겠습니다.

|  |
| --- |
| using System;  using System.Runtime.InteropServices;  using System.Drawing;  namespace 원격제어기  {  [Flags]  public enum MouseFlag  {  ME\_MOVE = 1, ME\_LEFTDOWN = 2, ME\_LEFTUP = 4,  ME\_RIGHTDOWN = 8, ME\_RIGHTUP = 0x10, ME\_MIDDLEDOWN = 0x20,  ME\_MIDDLEUP = 0x40, ME\_WHEEL = 0x800, ME\_ABSOULUTE = 8000  }  /// <summary>  /// 키보드 이벤트 열거형  /// </summary>  [Flags]  public enum KeyFlag  {  /// <summary>  /// 키 누름  /// </summary>  KE\_DOWN = 0,  /// <summary>  /// 확장 키  /// </summary>  KE\_\_EXTENDEDKEY = 1,  /// <summary>  /// 키 뗌  /// </summary>  KE\_UP = 2  }  /// <summary>  /// Native(Win32 API) 기술 래핑 클래스(정적 클래스)  /// </summary>  public static class WrapNative  {  [DllImport("user32.dll")]  static extern void mouse\_event(int flag, int dx, int dy, int buttons, int extra);  [DllImport("user32.dll")]  static extern bool GetCursorPos(ref Point point);  [DllImport("user32.dll")]  static extern int SetCursorPos(int x, int y);  [DllImport("User32.dll")]  static extern void keybd\_event(byte vk, byte scan, int flags, int extra);  /// <summary>  /// 키 누름(DOWN) 이벤트를 발생시키는 메서드  /// </summary>  /// <param name="keycode">키</param>  public static void KeyDown(int keycode)  {  keybd\_event((byte)keycode, 0, (int)KeyFlag.KE\_DOWN, 0);  }  /// <summary>  /// 키 뗌(UP) 이벤트를 발생시키는 메서드  /// </summary>  /// <param name="keycode">키</param>  public static void KeyUp(int keycode)  {  keybd\_event((byte)keycode, 0, (int)KeyFlag.KE\_UP, 0);  }  /// <summary>  /// 마우스 좌표를 바꾸는 메서드  /// </summary>  /// <param name="x">바꿀 X 좌표</param>  /// <param name="y">바꿀 Y 좌표</param>  public static void Move(int x, int y)  {  SetCursorPos(x, y);  }  /// <summary>  /// 마우스 좌표를 바꾸는 메서드  /// </summary>  /// <param name="pt">바꿀 포인트</param>  public static void Move(Point pt)  {  Move(pt.X, pt.Y);  }  /// <summary>  /// 프로그램 방식으로 마우스 왼쪽 버튼 누름 이벤트 발생시키는 메서드  /// </summary>  public static void LeftDown()  {  mouse\_event((int)MouseFlag.ME\_LEFTDOWN, 0, 0, 0, 0);  }  /// <summary>  /// 프로그램 방식으로 마우스 왼쪽 버튼 뗌 이벤트 발생시키는 메서드  /// </summary>  public static void LeftUp()  {  mouse\_event((int)MouseFlag.ME\_LEFTUP, 0, 0, 0, 0);  }  }  } |

[소스 9.12] WrapNative.cs

public class Remote

{

원격 제어 대상 호스트도 단일체 패턴을 적용합시다. 단일체를 기억하는 정적 멤버 필드와 이를 가져오기 할 수 있는 속성을 제공합니다.

static Remote singleton;

public static Remote Singleton

{

get{ return singleton; }

}

정적 생성자에서 단일체를 생성하여 멤버에 설정합니다.

static Remote()

{

singleton = new Remote();

}

원격 제어를 요청할 때 처리를 위해 이벤트를 멤버로 선언합니다.

public event RecvRCInfoEventHandler RecvedRCInfo = null;

키보드나 마우스 이벤트를 수신하였을 때 처리할 이벤트를 멤버로 선언합니다.

public event RecvKMEEventHandler RecvedKMEvent = null;

키보드나 마우스 이벤트를 수신하는 서버를 위해 멤버를 선언합니다.

RecvEventServer res = null;

자신의 사각 영역을 기억하는 멤버 속성을 제공합시다.

public Rectangle Rect{ get; private set; }

기본 생성자는 접근 지정을 private로 설정하여 다른 클래스에서 개체를 생성할 수 없게 합니다.

private Remote()

{

자동화 요소의 루트 요소를 참조하여 사각 영역을 구합니다.

AutomationElement ae = AutomationElement.RootElement;

System.Windows.Rect rt = ae.Current.BoundingRectangle;

Rect = new Rectangle((int)rt.Left, (int)rt.Top, (int)rt.Width, (int)rt.Height);

SetupServer의 원격 제어 요청이 왔을 때 처리할 이벤트 핸들러를 등록합니다.

SetupServer.RecvedRCInfo += new RecvRCInfoEventHandler(SetupServer\_RecvedRCInfo);

그리고 SetupServer를 가동합니다. 여기에서는 포트 번호를 20002으로 하드 코딩할게요.

SetupServer.Start(MyIP, 20002);

}

원격 제어 요청 이벤트는 By Pass 합니다.

void SetupServer\_RecvedRCInfo(object sender, RecvRCInfoEventArgs e)

{

if (RecvedRCInfo != null){ RecvedRCInfo(this, e); }

}

자신의 IP를 가져오기 속성을 제공합시다.

public string MyIP

{

get{ return Controller.Singleton.MyIP; }

}

키보드와 마우스 이벤트를 수신하는 서버를 가동하는 메서드를 제공합시다.

public void RecvEventStart()

{

포트 번호는 20010으로 하드 코딩할게요.

res = new RecvEventServer(MyIP, 20010);

res 개체의 메시지를 수신하였을 때 처리할 이벤트 핸들러를 등록합니다.

res.RecvedKMEvent += new RecvKMEEventHandler(res\_RecvKMEEventHandler);

}

void res\_RecvKMEEventHandler(object sender, RecvKMEEventArgs e)

{

자신에게 등록한 이벤트 핸들러가 있으면 By Pass 합니다.

if (RecvedKMEvent != null){ RecvedKMEvent(this, e); }

이벤트 종류에 따라 래핑한 API 개체의 메서드를 호출하여 프로그램 방식으로 이벤트를 발생하게 합니다.

switch (e.MT)

{

case MsgType.MT\_KDOWN: WrapNative.KeyDown(e.Key); break;

case MsgType.MT\_KEYUP: WrapNative.KeyUp(e.Key); break;

case MsgType.MT\_M\_LEFTDOWN: WrapNative.LeftDown(); break;

case MsgType.MT\_M\_LEFTUP: WrapNative.LeftUp(); break;

case MsgType.MT\_M\_MOVE: WrapNative.Move(e.Now); break;

}

}

멈추는 메서드를 제공합시다.

public void Stop()

{

SetupServer를 닫고 키보드와 마우스 이벤트 메시지를 수신 서버도 닫습니다.

SetupServer.Close();

if (res != null){ res.Close(); res = null; }

}

}

|  |
| --- |
| using System.Windows.Automation;  using System.Drawing;  namespace 원격제어기  {  public class Remote  {  static Remote singleton;  public static Remote Singleton  {  get{ return singleton; }  }  static Remote(){ singleton = new Remote(); }  public event RecvRCInfoEventHandler RecvedRCInfo = null;  public event RecvKMEEventHandler RecvedKMEvent = null;  RecvEventServer res = null;  public Rectangle Rect{ get; private set; }  private Remote()  {  AutomationElement ae = AutomationElement.RootElement;  System.Windows.Rect rt = ae.Current.BoundingRectangle;  Rect = new Rectangle((int)rt.Left, (int)rt.Top, (int)rt.Width, (int)rt.Height);  SetupServer.RecvedRCInfo +=  new RecvRCInfoEventHandler(SetupServer\_RecvedRCInfo);  SetupServer.Start(MyIP, 20002);  }  void SetupServer\_RecvedRCInfo(object sender, RecvRCInfoEventArgs e)  {  if (RecvedRCInfo != null)  {  RecvedRCInfo(this, e);  }  }  public string MyIP  {  get{ return Controller.Singleton.MyIP; }  }  public void RecvEventStart()  {  res = new RecvEventServer(MyIP, 20010);  res.RecvedKMEvent += new RecvKMEEventHandler(res\_RecvKMEEventHandler);  }  void res\_RecvKMEEventHandler(object sender, RecvKMEEventArgs e)  {  if (RecvedKMEvent != null){ RecvedKMEvent(this, e); }  switch (e.MT)  {  case MsgType.MT\_KDOWN: WrapNative.KeyDown(e.Key); break;  case MsgType.MT\_KEYUP: WrapNative.KeyUp(e.Key); break;  case MsgType.MT\_M\_LEFTDOWN: WrapNative.LeftDown(); break;  case MsgType.MT\_M\_LEFTUP: WrapNative.LeftUp(); break;  case MsgType.MT\_M\_MOVE: WrapNative.Move(e.Now); break;  }  }  public void Stop()  {  SetupServer.Close();  if (res != null){ res.Close(); res = null; }  }  }  } |

[소스 9.12] Remote.cs

* 1. 폼

먼저 가상 커서를 정의합시다. 원격 제어를 하는 곳에서는 상대 호스트의 마우스 위치를 화면에 표시해야 합니다. 이 부분을 도구 스타일의 작은 폼으로 정의합시다. 프로젝트에 폼을 하나 추가합니다. 여기에서는 VirtualCursorForm이라고 명명할게요.

Form의 AutoSizeMode를 GrowAndShrink로 설정하세요. BackColor는 Red, FormBorderSytle은 None, TopMost를 True로 지정합니다.

폼의 Load 이벤트 핸들러를 등록하세요.

private void VirtualCursorForm\_Load(object sender, EventArgs e)

{

Remote 클래스 단일체의 키보드와 마우스 이벤트 수신 이벤트 핸들러를 등록하세요.

Remote.Singleton.RecvedKMEvent +=

new RecvKMEEventHandler(Singleton\_RecvedKMEEvent);

}

void Singleton\_RecvedKMEEvent(object sender, RecvKMEEventArgs e)

{

만약 수신한 메시지가 마우스 이동 이벤트일 때 위치를 변경합니다.

if (e.MT == MsgType.MT\_M\_MOVE)

{

Location = new Point(e.Now.X + 3, e.Now.Y + 3);

}

}

|  |
| --- |
| using System;  using System.Drawing;  using System.Windows.Forms;  namespace 원격제어기  {  public partial class VirtualCursorForm : Form  {  public VirtualCursorForm()  {  InitializeComponent();  }  private void VirtualCursorForm\_Load(object sender, EventArgs e)  {  Remote.Singleton.RecvedKMEvent +=  new RecvKMEEventHandler(Singleton\_RecvedKMEEvent);  }  void Singleton\_RecvedKMEEvent(object sender, RecvKMEEventArgs e)  {  if (e.MT == MsgType.MT\_M\_MOVE)  {  Location = new Point(e.Now.X + 3, e.Now.Y + 3);  }  }  }  } |

[소스 9.13] VirtualCursorForm.cs

이번에는 원격 제어할 때 대상 호스트의 화면을 표시하는 RemoteCleintForm 폼을 추가하세요.

자식 컨트롤로 PictureBox 컨트롤을 추가하고 이름을 pbox\_remote으로 정합시다. PictureBox 컨트롤의 Dock 속성을 Fill로 설정하고 SizeMode 속성르 StretchImage로 설정하세요.

멤버 필드로 이미지를 수신하였는지 확인하는 멤버와 이미지 크기를 기억할 멤버를 선언하세요.

bool check;

Size csize;

이미지 전송 클라이언트 개체를 가져오기하는 속성을 정의하여 코드를 간결하게 합시다.

SendEventClient EventSC

{

get{ return Controller.Singleton.SendEventClient; }

}

폼의 Load 이벤트 핸들러를 등록합니다.

private void RemoteCleintForm\_Load(object sender, EventArgs e)

{

원격 제어기 단일체의 이미지 수신 이벤트 핸들러를 등록합니다.

Controller.Singleton.RecvedImage +=

new RecvImageEventHandler(Singleton\_RecvImageEventHandler);

}

void Singleton\_RecvImageEventHandler(object sender, RecvImageEventArgs e)

{

if (check == false)

{

check가 false일 때 키보드, 마우스 이벤트를 송신하는 클라이언트를 가동하고 초기 설정합니다.

Controller.Singleton.StartEventClient();

check = true;

csize = e.Image.Size;

}

그리고 이벤트로 전달받은 이미지로 PictureBox 컨트롤의 Image 속성을 설정합니다.

pbox\_remote.Image = e.Image;

}

폼의 KeyDown 이벤트 핸들러를 등록합니다.

private void RemoteCleintForm\_KeyDown(object sender, KeyEventArgs e)

{

check가 참일 때 누른 키를 이벤트 전송 클라이언트에게 보냅니다.

if (check == true){ EventSC.SendKeyDown(e.KeyValue); }

}

폼의 KeyUp 이벤트를 등록합니다.

private void RemoteCleintForm\_KeyUp(object sender, KeyEventArgs e)

{

마찬가지로 check가 참일 때 뗀 키를 이벤트 전송 클라이언트에게 보냅니다.

if (check == true){ EventSC.SendKeyUp(e.KeyValue); }

}

폼의 MouseMove 이벤트도 등록합니다.

private void RemoteCleintForm\_MouseMove(object sender, MouseEventArgs e)

{

check가 참을 때 마우스 좌표를 인자를 변환하여 이벤트 전송 클라이언트에게 보냅니다.

if (check == true)

{

Point pt = ConvertPoint(e.X, e.Y); EventSC.SendMouseMove(pt.X, pt.Y);

}

}

private Point ConvertPoint(int x, int y)

{

마우스의 좌표를 대상 호스트의 좌표로 변환하여 반환합니다.

int nx = csize.Width \* x / pbox\_remote.Width;

int ny = csize.Height \* y / pbox\_remote.Height;

return new Point(nx, ny);

}

폼의 MouseDown 이벤트 핸들러도 등록합니다.

private void RemoteCleintForm\_MouseDown(object sender, MouseEventArgs e)

{

현재 마우스 좌표를 얻어와서 이벤트 전송 클라이언트에게 보냅니다.

if (check == true)

{

Text = e.Location.ToString();

EventSC.SendMouseDown(e.Button);

}

}

폼의 MouseUp 이벤트 핸들러도 등록합니다.

private void RemoteCleintForm\_MouseUp(object sender, MouseEventArgs e)

{

버튼 정보를 이벤트 전송 클라이언트에게 보냅니다.

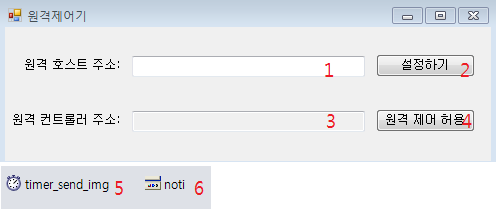
if (check == true){ EventSC.SendMouseUp(e.Button); }

}

|  |
| --- |
| using System;  using System.Drawing;  using System.Windows.Forms;  namespace 원격제어기  {  public partial class RemoteCleintForm : Form  {  bool check;  Size csize;  SendEventClient EventSC  {  get  {  return Controller.Singleton.SendEventClient;  }  }  public RemoteCleintForm()  {  InitializeComponent();  }  private void RemoteCleintForm\_Load(object sender, EventArgs e)  {  Controller.Singleton.RecvedImage +=  new RecvImageEventHandler(Singleton\_RecvImageEventHandler);  }  void Singleton\_RecvImageEventHandler(object sender, RecvImageEventArgs e)  {  if (check == false)  {  Controller.Singleton.StartEventClient();  check = true;  csize = e.Image.Size;  }  pbox\_remote.Image = e.Image;  }  private void RemoteCleintForm\_KeyDown(object sender, KeyEventArgs e)  {  if (check == true)  {  EventSC.SendKeyDown(e.KeyValue);  }  }  private void RemoteCleintForm\_KeyUp(object sender, KeyEventArgs e)  {  if (check == true)  {  EventSC.SendKeyUp(e.KeyValue);  }  }  private void RemoteCleintForm\_MouseMove(object sender, MouseEventArgs e)  {  if (check == true)  {  Point pt = ConvertPoint(e.X, e.Y);  EventSC.SendMouseMove(pt.X, pt.Y);  }  }  private Point ConvertPoint(int x, int y)  {  int nx = csize.Width \* x / pbox\_remote.Width;  int ny = csize.Height \* y / pbox\_remote.Height;  return new Point(nx, ny);  }  private void RemoteCleintForm\_MouseDown(object sender, MouseEventArgs e)  {  if (check == true)  {  Text = e.Location.ToString();  EventSC.SendMouseDown(e.Button);  }  }  private void RemoteCleintForm\_MouseUp(object sender, MouseEventArgs e)  {  if (check == true)  {  EventSC.SendMouseUp(e.Button);  }  }  }  } |

[소스 9.14] RemoteClientForm.cs

마지막으로 MainForm을 구현합시다. 프로젝트 생성 시에 만들어진 Form1의 이름을 MainForm으로 변경하세요.



[그림 9.2] MainForm 자식 컨트롤 배치

|  |  |  |  |
| --- | --- | --- | --- |
| 번호 | 컨트롤 타입 | 컨트롤 명 | 설명 |
| 1 | TextBox | tbox\_ip | 원격 제어 대상 호스트 IP 주소 입력 |
| 2 | Button | btn\_setting | 원격 제어 대상 호스트에게 요청 |
| 3 | TextBox | tbox\_controller\_ip | 원격 제어기 호스트 IP 주소 |
| 4 | Button | btn\_ok | 원격 제어 허용 |
| 5 | Timer | timer\_send\_img | 이미지 수신 타이머 |
| 6 | NotifyIcon | noti | 원격 제어를 당할 때 Notify 아이콘 |

[표 9.2] Form1의 컨트롤 배치

가상 커서와 원격 제어 화면과 상대 IP와 포트를 멤버를 선언합니다.

VirtualCursorForm vcf = null;

RemoteCleintForm rcf = null;

string sip;

int sport;

폼의 Load 이벤트 핸들러를 등록하세요.

private void MainForm\_Load(object sender, EventArgs e)

{

가상 커서 폼과 원격 제어 호스트 화면을 표시할 폼을 생성합니다.

vcf = new VirtualCursorForm();

rcf = new RemoteCleintForm();

원격 제어 요청을 수신하였을 때의 처리 이벤트 핸들러를 등록합니다.

Remote.Singleton.RecvedRCInfo += new RecvRCInfoEventHandler(Remote\_RecvedRCInfo);

}

void Remote\_RecvedRCInfo(object sender, RecvRCInfoEventArgs e)

{

상대 IP 주소를 텍스트 박스에 표시하고 IP와 포트 정보를 멤버에 설정합니다.

tbox\_controller\_ip.Text = e.IPAddressStr;

sip = e.IPAddressStr;

sport = e.Port;

btn\_ok.Enabled = true;

}

폼의 FormClosed 이벤트 핸들러를 등록하여 해제화 작업을 작성합니다.

private void MainForm\_FormClosed(object sender, FormClosedEventArgs e)

{

Remote.Singleton.Stop();

Controller.Singleton.Stop();

Application.Exit();

}

설정하기 버튼 클릭 이벤트 핸들러를 등록하세요.

private void btn\_setting\_Click(object sender, EventArgs e)

{

if (tbox\_ip.Text == Remote.Singleton.MyIP)

{

입력한 IP가 자신의 IP일 때는 제어할 수 없음을 알려줍니다.

MessageBox.Show("같은 호스트를 원격 제어할 수 없습니다.");

tbox\_ip.Text = string.Empty; return;

}

입력한 IP를 얻어와서 원격 제어기를 가동합니다.

string host\_ip = tbox\_ip.Text;

Rectangle rect = Remote.Singleton.Rect;

Controller.Singleton.Start(host\_ip);

화면 크기에 맞게 원격 제어 호스트의 화면을 띄우줄 rcf 폼의 크기를 설정하여 시각화합니다.

rcf.ClientSize = new Size(rect.Width - 40, rect.Height - 80);

rcf.Show();

}

원격 제어 허용 버튼의 클릭 이벤트 핸들러를 등록합니다.

private void btn\_ok\_Click(object sender, EventArgs e)

{

MainForm은 화면에서 숨깁니다.

this.Hide();

키보드와 마우스 이벤트를 수신하는 Remote 클래스의 단일체를 가동합니다.

Remote.Singleton.RecvEventStart();

이미지 수신 타이머도 가동하고 가상 커서도 시각화합니다.

timer\_send\_img.Start(); vcf.Show();

}

Notify 아이콘의 더블 클릭 이벤트 핸들러를 등록합니다. 여기에서는 화면에서 숨겼던 자신을 시각화합니다.

private void noti\_MouseDoubleClick(object sender, MouseEventArgs e)

{

this.Show();

}

이미지 수신 타이머 이벤트 핸들러를 등록합니다.

private void timer\_send\_img\_Tick(object sender, EventArgs e)

{

먼저 사각 영역을 얻어옵니다.

Rectangle rect = Remote.Singleton.Rect;

얻어온 사각 영역 크기의 비트맵 개체를 생성합니다.

Bitmap bitmap = new Bitmap(rect.Width, rect.Height);

이미지로부터 Graphics 개체를 구합니다.

Graphics gp = Graphics.FromImage(bitmap);

사각 영역의 크기로 Size 개체를 생성합니다.

Size size2 = new Size(rect.Width, rect.Height);

화면의 전체 이미지를 복사합니다.

gp.CopyFromScreen(new Point(0, 0), new Point(0, 0), size2);

Graphics 개체는 더 이상 할 것이 없으므로 해제합니다.

gp.Dispose();

try

{

이미지 전송 클라이언트 개체를 생성합니다. 이미지 수신 서버는 포트번호를 20004로 하드 코딩한 상태입니다.

ImageClient ic = new ImageClient(sip, 20004);

이미지 전속 클라이언트 개체의 비동기 방식으로 이미지를 전송하는 SendImageAsync 메서드를 호출합니다.

ic.SendImageAsync(bitmap, null);

}

catch

{

예외가 발생하면 이미지 전송 타이머를 멈추고 서버에 연결이 실패하였음을 알려줍니다.

timer\_send\_img.Stop();

MessageBox.Show("서버에 연결 실패");

this.Close();

}

}

원격 제어 프로그램 만들기를 마칠게요. 이를 응용하는 것은 여러분의 몫으로 남깁니다.

|  |
| --- |
| using System;  using System.Drawing;  using System.Windows.Forms;  namespace 원격제어기  {  public partial class MainForm : Form  {  VirtualCursorForm vcf = null;  string sip;  int sport;  RemoteCleintForm rcf = null;  public MainForm()  {  InitializeComponent();  }  private void MainForm\_Load(object sender, EventArgs e)  {  vcf = new VirtualCursorForm();  rcf = new RemoteCleintForm();  Remote.Singleton.RecvedRCInfo +=  new RecvRCInfoEventHandler(Remote\_RecvedRCInfo);  }  void Remote\_RecvedRCInfo(object sender, RecvRCInfoEventArgs e)  {  tbox\_controller\_ip.Text = e.IPAddressStr;  sip = e.IPAddressStr;  sport = e.Port;  btn\_ok.Enabled = true;  }  private void MainForm\_FormClosed(object sender, FormClosedEventArgs e)  {  Application.Exit();  Remote.Singleton.Stop();  Controller.Singleton.Stop();  }  private void btn\_setting\_Click(object sender, EventArgs e)  {  if (tbox\_ip.Text == Remote.Singleton.MyIP)  {  MessageBox.Show("같은 호스트를 원격 제어할 수 없습니다.");  tbox\_ip.Text = string.Empty;  return;  }  string host\_ip = tbox\_ip.Text;  Rectangle rect = Remote.Singleton.Rect;  Controller.Singleton.Start(host\_ip);  rcf.ClientSize = new Size(rect.Width - 40, rect.Height - 80);  rcf.Show();  }  private void btn\_ok\_Click(object sender, EventArgs e)  {  this.Hide();  Remote.Singleton.RecvEventStart();  timer\_send\_img.Start();  vcf.Show();  }  private void noti\_MouseDoubleClick(object sender, MouseEventArgs e)  {  this.Show();  }  private void timer\_send\_img\_Tick(object sender, EventArgs e)  {  Rectangle rect = Remote.Singleton.Rect;  Bitmap bitmap = new Bitmap(rect.Width, rect.Height);  Graphics gp = Graphics.FromImage(bitmap);  Size size2 = new Size(rect.Width, rect.Height);  gp.CopyFromScreen(new Point(0, 0), new Point(0, 0), size2);  gp.Dispose();  try  {  ImageClient ic = new ImageClient(sip, 20004);  ic.SendImageAsync(bitmap, null);  }  catch  {  timer\_send\_img.Stop();  MessageBox.Show("서버에 연결 실패");  this.Close();  }  }  }  } |

[소스 9.15] MainForm.cs