Comunicaciones en .NET Compact Framework

Internet con sockets

- El compact Framework lleva incorporada una implementación de los sockets para windows.
- Con los sockets, un proceso se identifica por la dirección de IP de su máquina (156.35.151.234), el protocolo que utilizará (TCP ó UDP) y el puerto de protocolo a través del que se comunicará.
- Secuencia de una comunicación:
  - Cliente:
    - El cliente “crea” su socket con los tres datos anteriores.
    - Prepara la información con la dirección del servidor.
    - Establece una conexión hacia el socket del servidor.
    - Cuando se establece la conexión, se intercambian información.
  - Servidor:
    - El servidor crea su socket con los tres datos de localización.
    - Declara ese socket como “de servidor” (pasivo).
    - Se pone a la espera de conexiones por parte de los clientes.
    - Cuando llega una conexión, la acepta, y crea un socket (distinto del pasivo) para intercambiar información con el cliente.

Internet con sockets. Clases

- El espacio de nombres que contiene todas las clases necesarias para las comunicaciones en red es `System.Net`. Las clases relacionadas con los sockets se encuentran en `System.Net.Sockets`.
- La clase que contiene los datos de un socket (dirección IP y puerto de protocolo) es `System.Net.IPEndPoint`.
- El protocolo se especifica de dos maneras:
  - Si se usa una clase `Socket`, en el constructor.
  - Si se usan "listeners", en la propia clase usada: `TcpListener`, `TcpClient` y `UdpClient`.
- El acceso al Servidor de Nombres (DNS) se hace a través de los métodos estáticos de la clase `Dns`:
  - `GetHostByAddress`: Devuelve el nombre de una máquina a partir de su dirección IP
  - `GetHostByName`: Devuelve una instancia de una clase `IPHostEntry` a partir de un nombre de máquina.
  - `GetHostName`: Devuelve el nombre de la máquina en que se ejecuta.
  - `Resolve`: Devuelve una instancia `IPHostEntry` a partir de un nombre de máquina o de una dirección IP

Programación cliente con sockets.

1. Se necesita una variable de la clase `Socket` y otra del tipo `EndPoint`.
2. Se crea el socket del cliente y se inicializa el EndPoint con la dirección del servidor al que se quiere conectar.
3. Se conecta con el servidor. Si no se puede, se activa la excepción `SocketException`. Si se puede, se lee y escribe del socket.

```
Socket cli;
EndPoint serv;

Serv = new IPEndPoint(IPAddress.Parse("156.35.151.2"),13);
Try{
     cli= new Socket(AddressFamily.InterNetwork,
     SocketType.Stream, ProtocolType.Tcp);
     cli.Connect(serv);
…
}
catch (SocketException e) {
…}
```

Programación servidor con sockets.

1. Se necesita una variable de la clase `Socket`.
2. Se crea el socket del servidor, se indica con un EndPoint, la dirección en la que se escuchará y se “enlaza” el socket con esa dirección.
3. Se coloca a la escucha al servidor. Si llega una petición, se acepta.

```
Socket serv, cli;

serv = new new Socket(AddressFamily.InterNetwork,
    SocketType.Stream, ProtocolType.Tcp);
serv.Bind(new IPEndPoint(IPAddress.Any,13));
serv.Listen((int)SocketOptionName.MaxConnections);
cli= serv.Accept();
if (cli != null)
```

Métodos de la clase `Socket`.

`Bind()`	Enlaza el socket con una dirección local.
`Listen()`	Pone el socket en modo pasivo.
`Accept()`	Acepta una conexión cuando llega.
`Receive()`	Lee los datos que llegan por la red.
`Send()`	Envía datos a través de la red.
`Connect()`	Conecta un cliente con un servidor determinado.
`Close()`	Cierra definitivamente una conexión.
`Shutdown()`	Cierra parcialmente una conexión.

Envío y recepción de información.

Los métodos `Send()` y `Receive()` envían y reciben **bytes** (unsigned char).

El envío de datos interpretados exige el uso de clases para la codificación de la información.

Para facilitar las conversiones se disponen de las clases `Encoding`, `Convert` y `BitConverter`. La clase Convert se utiliza para convertir un tipo básico en otro tipo básico. En este caso, de `byte` a cualquier otro. La clase BitConverter hace algo similar, permitiendo convertir un conjunto de bytes en cualquier otro tipo básico, y viceversa.

Sin embargo, las comunicaciones en .Net están pensadas para que las conversiones de tipos se hagan desde y hacia cadenas de caracteres.

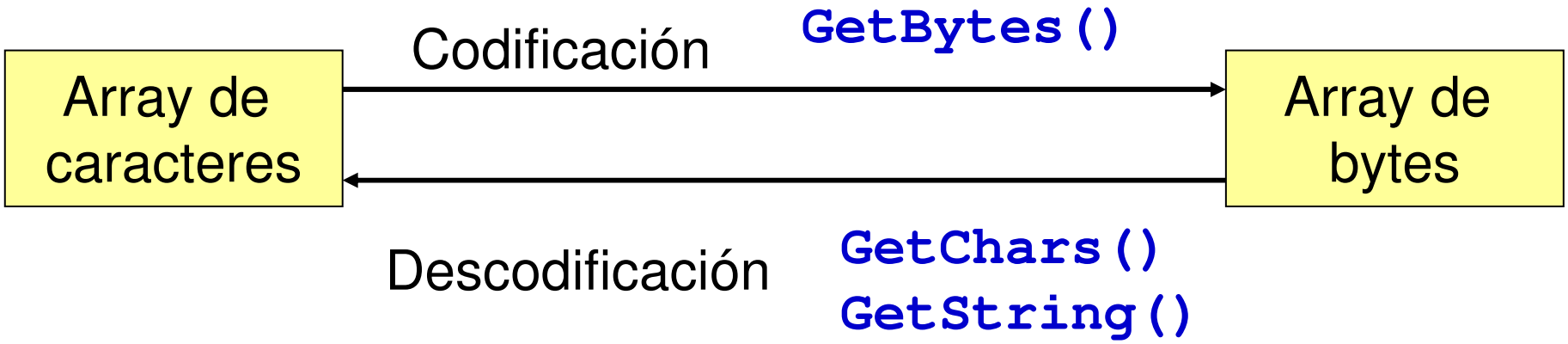
La clase `Encoding`

La clase `Encoding` nos permite codificar los distintos tipos de cadenas de caracteres. Tiene distintas implementaciones.

Clase	Descripción
`ASCIIEncoding`	Codificación ASCII de los caracteres Unicode.
`Encoding`	Convierte bloques de caracteres Unicode en bloques de bytes.
`UnicodeEncoding`	Codificación UTF16 de caracteres Unicode.
`UTF8Encoding`	Codificación UTF8 de caracteres Unicode.
`UTF7Encoding`	Codificación UTF7 de caracteres Unicode.

Métodos de la clase `Encoding`

El objetivo de los `Encoding` es realizar codificaciones y descodificaciones.



GetBytes()	Codifica un array de caracteres o un String en un array de bytes.
GetChars()	Descodifica un array de bytes en un array de caracteres.
GetString()	Descodifica un array de bytes en un String.

Programación Servidor. `TCPListener`

- La clase `TcpListener` permite esperar por conexiones en un determinado puerto y manejar fácilmente el envío y recepción de datos.

- Métodos disponibles:
  - `Start()` .- Inicia la escucha.
  - `AcceptTcpClient()` .- Acepta la conexión del cliente. Devuelve un clase `TcpClient`.
  - `AcceptSocket()` .- Acepta la conexión del cliente y devuelve una clase `Socket`.
  - `Stop()` .- Detiene la escucha y se pierden las solicitudes que estén en la cola.

- Métodos de especificar el puerto y dirección IP:

```
//asignación directa del puerto
TcpListener servidorTcp= new TcpListener(1300);

//puerto asignado por el sistema
TcpListener servidorTcp= new TcpListener(0);

//Usando un EndPoint
IPEndPoint EndP = new IPEndPoint(IPAddress.Parse("156.35.151.253"),
  1300);
TcpListener servidorTcp= new TcpListener(EndP);
```

Programación Cliente. `TCPClient`

- La clase `TcpClient` representa un cliente TCP que permite realizar aplicaciones sencillas.

- Métodos disponibles:
  - `Connect()` .- Conecta al cliente con el servidor TCP. Hay que suministrar máquina y puerto.
  - `GetStream()` .- Devuelve una clase NetworkStream usada para el envío y recepción de datos.
  - `Close()` .- Cierra la conexión.

- Métodos de especificar el servidor:

```
//Usando un EndPoint
IPEndPoint serv= new
  IPEndPoint(IPAddress.Parse("156.35.151.2"),1300);
TcpClient cli=new TcpClient(serv);

//Usando Connect
TcpClient cli= new TcpClient();
cli.connect("sirio.atc.uniovi.es", 1300);
```

Métodos de la clase `NetworkStream`

La clase `NetworkStream` permite enviar y recibir datos como si se leyese un fichero.

De un `Stream` se obtienen **`bytes`** que se deben/pueden codificar usando la clase Encoding.

Read()	Lee datos del stream y los deja en un array de bytes.
Write()	Manda los datos contenidos en un array de bytes a través de la conexión.

Ejemplo de uso de NetworkStream

```
//Usando Connect
TcpClient cli= new TcpClient();
cli.connect("sirio.atc.uniovi.es", 1300);

NetworkStream Strm;

Strm=cli.GetStream();
Byte[] buffer= new byte[40];
int nbytes= Strm.Read(buffer, 0 , buffer.Length);
String respuesta= Encoding.ASCII.GetString(buffer,0, nbytes);

// Imprimimos la respuesta
….
cli.close();
```

Infrarrojos

- Las comunicaciones con infrarrojos ocurren siempre entre un cliente y un servidor.
- Los dispositivos se identifican por su nombre y su ID. Los servicios dentro del dispositivo se identifican por un nombre.
- Secuencia de una comunicación:
  - Un dispositivo que desea establecer una comunicación por infrarrojos busca en su rango de alcance otros dispositivos.
  - Cuando encuentra algún dispositivo, establece una conexión con el servicio que desea y que el servidor debe de proveer.
- La clase central de las comunicaciones infrarrojas es `IrDAClient` en `System.Net.Sockets`. Actúa tanto como cliente como servidor.
- Para usar `IrDAClient` es necesario añadir una referencia a la librería de enlace dinámico **`System.Net.IrDA.dll`** (Menú **`Proyecto-> Agregar referencia`**)

Infrarrojos. Cliente

1. Crear una instancia de `IrDAClient`.
2. Obtener una lista de los dispositivos dentro de rango. Se usa `IrDAClient.DiscoverDevices`. Este método retorna un vector de `IrDADeviceInfo`.
3. Buscar en el vector el dispositivo que suministra el servicio deseado. Se llama a `IrDAClient.Connect()`.
4. Para leer datos de la conexión IR se usa un objeto `StreamReader` (o `BinaryReader`) asociándolo con el `Stream` de `IrDAClient` .
5. Para enviar datos a través de la conexión IR se usa también un objeto de tipo `StreamWriter` (o `BinaryWriter`) asociándolo al `Stream` de `IrDAClient` .
6. Se cierra la comunicación IR llamando al método `Close` de `IrDAClient` .

Infrarrojos. Servidor

1. Crear una instancia de `IrDAListener`. Recibe como parámetro el nombre del servicio a suministrar.
2. Invocar el método Start de `IrDAListener`. El proceso se bloquea hasta recibir una conexión.
3. Al recibir una conexión, crear una instancia de `IrDAClient` llamando al método `IrDAListener.AcceptIrDAClient()`.
4. Usar la instancia creada de `IrDAClient` para las comunicaciones.
5. Para leer datos de la conexión IR se usa un objeto `StreamReader` (o `BinaryReader`) asociándolo con el `Stream` de `IrDAClient` .
6. Para enviar datos a través de la conexión IR se usa también un objeto de tipo `StreamWriter` (o `BinaryWriter`) asociándolo al `Stream` de `IrDAClient` .
7. Se cierra la comunicación IR llamando al método `Close` de `IrDAClient` .

Código Cliente

```
// Crear una instancia de IrDAClient.
IrDAClient mIR = new IrDAClient();
//Obtener una lista de los dispositivos dentro de rango.
IrDeviceInfo cerca= mIR.DiscoverDevices(MAX)
// Buscar en el vector el dispositivo que suministra el servicio
// deseado.
IrDAEndpoint s=new IrDAEndpoint(cerca[0].DeviceID,"SERVICIO");
mIR.Connect(s);
// Para leer datos se usa un objeto StreamReader
StreamReader lee= new StreamReader(mIR.GetStream(),Encoding.ASCII);
lee.Read(...);
//Para enviar datos se usa un objeto StreamWriter
StreamWriter esc= new StreamWriter(mIR.GetStream(),Encoding.ASCII)
esc.WriteLine(...);
//Se cierra la comunicación
mIR.Close();
```

Código Servidor

```
// Crear una instancia de IrDAListener.
IrDAListener miIR= new IrDEListener("SERVICIO");
// Se pone a la escucha de peticiones
miIR.Start();
// Se recibe una conexión, se crea una instancia de IrDAClient
miCl=miIR.AcceptIrDAClient();
// Para leer datos se usa un objeto StreamReader
StreamReader lee= new StreamReader(mCl.GetStream(),Encoding.ASCII);
lee.Read(...);
//Para enviar datos se usa un objeto StreamWriter
StreamWriter esc= new StreamWriter(mCl.GetStream(),Encoding.ASCII)
esc.WriteLine(...);
//Se cierra la comunicación
mIR.Close();
```