MTRN3500
Computing Applications in Mechatronics Systems

Ethernet Communication

T3 - 2020

UNSW SYDNEY

Use

- Many of the sensors we use has Ethernet Interfaces. For example, the LiDAR you will be using in Assignment 2 has an Ethernet Interface. The motion control system that drives the UGV also has an Ethernet interface. Therefore it is very useful to be able to write software that interfaces control computers to sensors and actuators via Ethernet.
- Ethernet has two protocols
  - TCP (Transmission Control Protocol)
  - UDP (User Datagram Protocol)

TCP

- First, our aim is not to learn about TCP protocol.
- We want to learn when to use TCP and how to use TCP.
- TCP features
  - TCP requires a connection to be established between two devices. This means the two devices know that one is sending the data and the other is receiving the data.
  - Similar to two people talking with each other.
  - TCP may have substantial latencies especially when the connection is weak.
  - TCP however, ensures data integrity
  - TCP is ideal to issue control command and receive high integrity return messages. For example SET ALTITUDE 24000

UDP

- UDP features
  - UDP does not require a connection.
  - Similar to a lecture.
  - UDP has substantially reduced latencies.
  - UDP does not ensure data integrity.
  - UDP may be used for low integrity data transmission. For example, to collect point cloud data or to transmit a fax message. If you miss a data point or two – no big deal

Clients and Servers

- A server has the ability to connect to many clients
- TCP servers have a listening ability to detect incoming requests for connections.
- When a client requests a connection the server can respond to it and establish a TCP connection.

- The clients can only make one connection.
- They do not have ability to listen.
- The connection requires a unique IP address and a port number.
- Many device we will connect to will not need a port number as they simply have one port and any port address will be ok for them.

Basic Communication principle

- All data that travel back and froth are binary. Therefore, we can represent them as binary bytes using **`unsigned char`** data type.
- Write operation:
  - Especially, when we send data to an Ethernet device, the device has the responsibility to interpret them the way it wants. We do not have to worry about that. We simply send it binary data bytes
  - However, the data at our disposal may be of different types. They may be **`double, float, int, unsigned char`**, ASCII strings or WCHAR strings.
  - We must convert the data to binary before sending them.

Basic Communication principle

- Read operation:
  - We know what we will get are the binary bytes. However, according to the understanding between the sender and the receiver, the binary data needs to be interpreted in an agreed way.
  - We must now do the reverse, i.e. decode the binary data bytes to suit the agreed interpretation.
  - We may have to decode the data into data types of **`double`**, **`float`**, **`int`**, **`unsigned char`**, ASCII strings or WCHAR strings.

Elements needed for TCP Communication

- We need a TCP client object or a server object
- We need to know an IP address
- We need a Port number, could be arbitrary.
- We need a place to store binary data to be transmitted.
- We need a place to store binary data received.
- We need a mechanism to send data.
- We need a mechanism to receive data.
- We must have encoding and decoding mechanisms.

TCP Client

- Mostly our sensors and actuators are servers.
- Therefore, our computers connecting to them must be Clients

- Instantiating a **`TcpClient`** object
- First include the statements below

```
using namespace System;
using namespace Net;
using namespace Sockets;
using namespace Text;
int main()
{
  // Declare handle to TcpClient object
  TcpClient^ Client;
```

TCP Client

- Get your port number

```
int PortNumber = 24000;
```

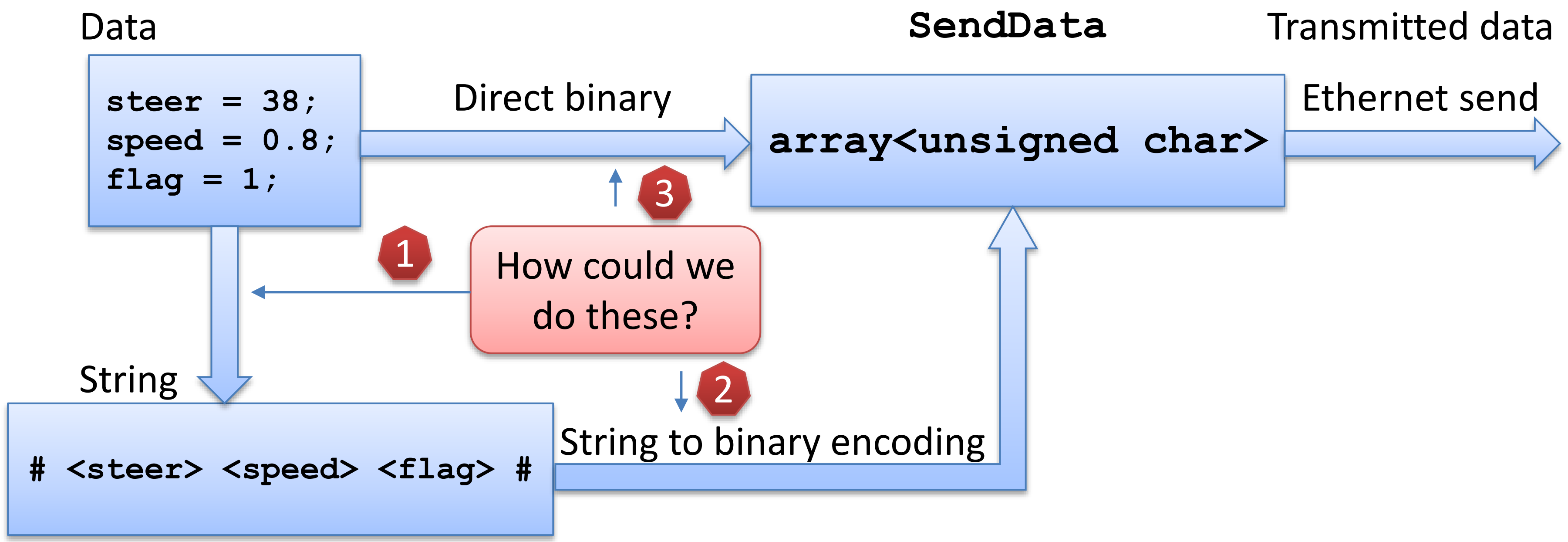
- Instantiate the TcpClient object and connect to it

```
Client = gcnew TcpClient(“192.168.5.100”, PortNumber);
```

- Configure Client

```
Client->Nodelay = true;
Client->ReceiveTimeout = 500;
Client->SendTimeout = 500;
Client->ReceiveBufferSize = 1024;
Client->SendBufferSize = 1024;
```

TCP Communication flow diagram (send)



TCP Send to a server

- Allocate space for send data buffer

```
SendData = gcnew array<unsigned char>(1024);
```

```
array<unsigned char>
```

- Prepare your data 1

```
Message = gcnew String("# ");
Message = Message + steer.ToString("F3")
                  + " " + speed.ToString("F3") + " 1 #";
```

OR

```
# <steer> <speed> <flag> #
```

```
Message = Message + steer.ToString("F3")
                  + " " + speed.ToString("F3") + " 0 #";
```

- Encoding data 2

```
SendData = Encoding::ASCII->GetBytes(Message);
```

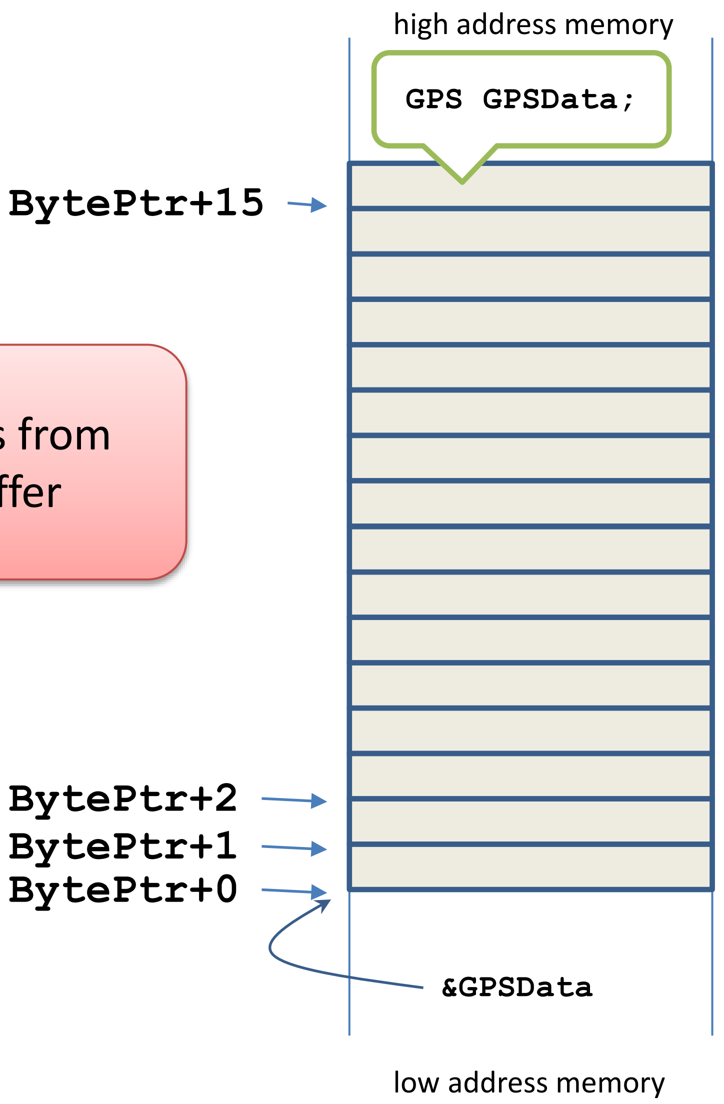
TCP Send to a server

- Direct Binary 3
  - This is best done when all the information to be transmitted is in a structure.

```
struct GPS
{
    double Lat;
    double Long;
};

GPS GPSData;
unsigned char* BytePtr;
BytePtr = (unsigned char*)(&GPSData);
for(int i = 0; i < sizeof(GPS);i++)
    SendData[i] = *(BytePtr+i);
```

Data movement is from structure to buffer



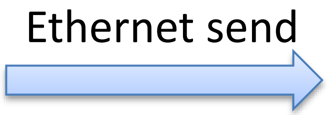
TCP send to server

- Create a stream object

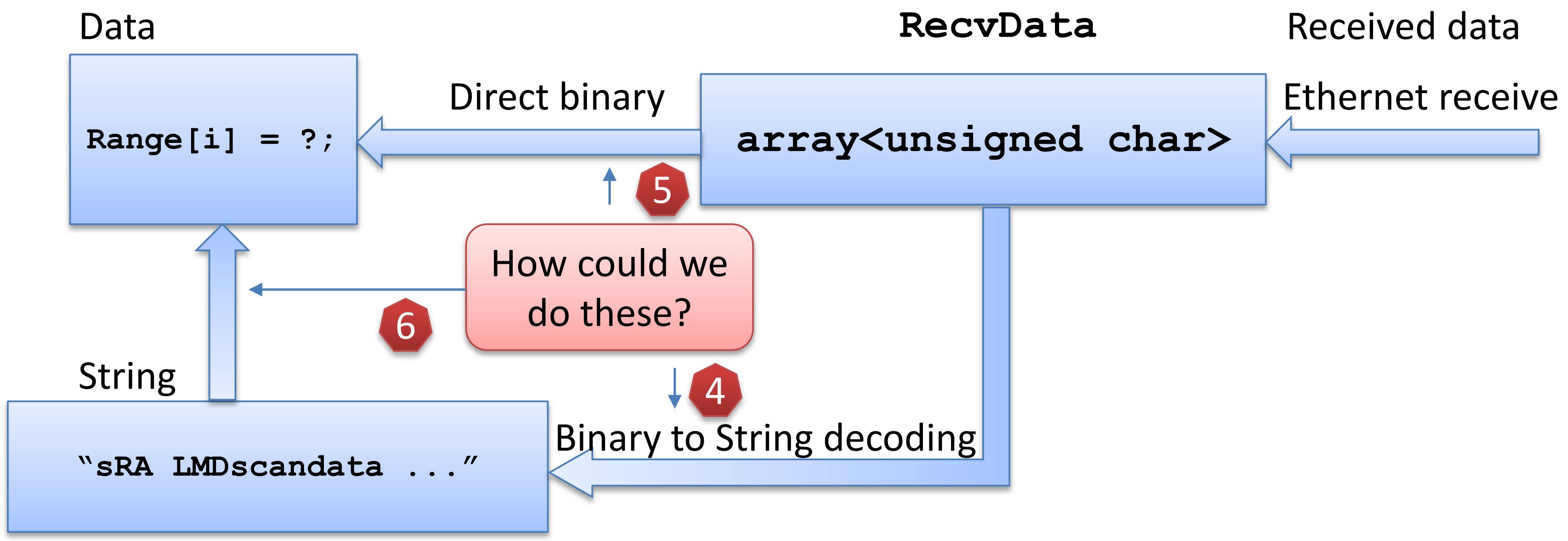
```
NetworkStream^ Stream = Client->GetStream();
```

- Sending

```
Stream->Write(SendData, 0, SendData->Length);
```



TCP Communication flow diagram (receive)



TCP receive from server

- Form a data buffer to receive data

```
array<unsigned char>^ RecvData;
RecvData = gcnew array<unsigned char>(5000);
```

- Get network stream

```
NetworkStream^ Stream = Client->GetStream();

Stream->Read(RecvData, 0, RecvData->Length);
```

- It may not read all your data in one read. Then use a loop

Ethernet receive

```
NumData = 0;
while(NumData != sizeof(GPS))
   NumData += Stream->Read(RecvData, NumData, sizeof(GPS)-NumData);
```

- By this time **RecvData** has data to fit **GPS** type object.

TCP receive from server

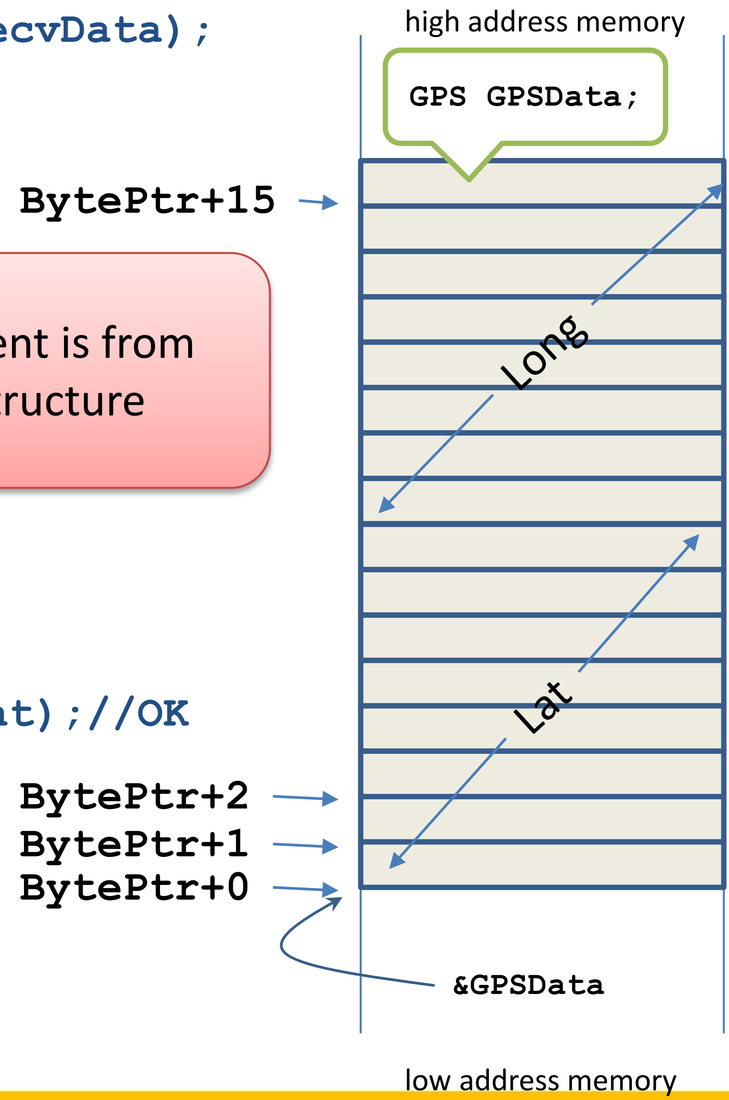
- Binary to String Decoding 4

```
LaserData = Encoding::ASCII->GetString(RecvData);
```

- Direct binary 5

Data movement is from buffer to structure

```
GPS GPSData;
unsigned char BytePtr;
BytePtr =(unsigned char*)&GPSData;
for(int i = 0; i < sizeof(GPS); i++)
    *(BytePtr+i) = RecvData[i];
Console::WriteLine("{0,12:F3}",GPSData.Lat);//OK
```



TCP receive from server

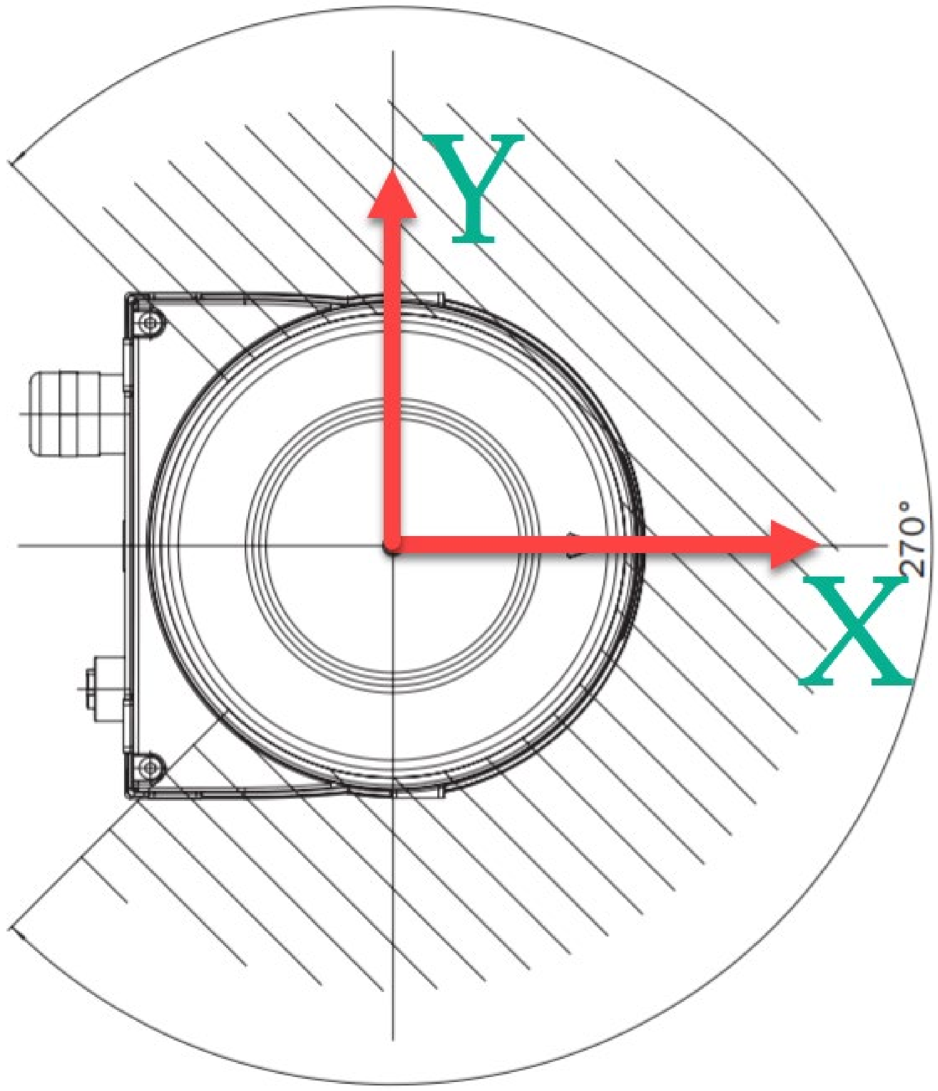
- String to data 6
  - Consider the string **LaserData** string

StartAngle
Resolution
Numdata

ranges in mm

```
sRA LMDscandata 0 1 9BF210 0 0 25E6 25EA 5B855C7F 5B85621E 0 0 7 0 0 1388
168 0 1 DIST1 3F800000 00000000 0 1388 169 152A 14A0 146E 1529 1573 15FE
17C8 17DF 1773 174B 174A 1746 174E 1764 16BA A79 A7F A6C 933 822 833 8DB
92C 88B 855 8A9 907 8C3 88F 849 81B 7E5 740 714 70A 710 742 7BE 7BF 7CD
7FA 80A 810 831 879 8D7 8A0 837 85B 97D BD2 BAC B5D B32 B08 9AF 6A6 65D
616 5EE 5D0 5C7 5D6 5EA 5F1 5F5 5F9 604 60A 623 629 625 63D 635 63A 65D
71A 777 7E4 8A5 C5C D08 D26 D16 CD4 B26 A89 A42 A2A A1D A10 A02 9E4 9EF
9DD 9DB 9DC 9D0 9D7 9F2 A00 A11 A26 A41 A8D A4D A06 A0E C0B 106D 10A1 1176
1153 11F0 1222 1269 1280 12A4 131D 138F 1380 133C 133F 1318 EB2 D1A EAF
ED9 F3B 10C5 143A 1495 1760 178F 115A 1098 107B 109F 10CF 12CC 13F5 136D
1333 1362 13DC 1630 1686 1699 174F 1824 1815 1816 180D 1800 17E1 17EE 37E
375 363 362 328 321 312 31F 314 30B 304 309 309 30A 30B 328 31E 303 307
305 304 309 306 304 313 313 30C 312 327 330 311 317 318 314 31B 312 312
31E 311 31D 31F 33E 369 3AD 3DE 43C 652 650 65F 669 65B 65E 661 66F 674
67A 678 66D 67A 683 68A 68A 697 6A6 6A2 6AE 6AB 6A5 6A1 671 641 617 5E8
5DE 5DE 5D9 5C8 5DF 5E5 602 607 624 64C 66B 6B8 52A 485 432 405 3ED 3F7
3E8 3D7 3D0 3C9 3D0 3CE 3B8 3BA 3A8 3AB 3A7 3A4 3A0 3A2 3A2 3AA 3A3 399
3A2 3A4 391 3AD 36A 196 0 11D 10A 111 10A 108 104 F0 ED DD D9 C9 D6 D2 CD
CC C5 AC AD BC B9 B6 AC B5 AD A6 AC A7 A6 B7 B0 AC AB AD B4 AF B2 B4 AB AF
B5 A2 A0 A9 A3 A2 A4 94 A0 A9 A5 9B A7 94 A9 AD A7 9B 9F A7 B0 A2 AF B9 B5
A8 B1 B2 C5 C6 C6 CB CC D0 C9 C1 C6 C2 B7 B4 9C 9A A1 AC 93 93 A3 A5 B3 AA
0 0 0 0 0 0
```

Laser coordinates



TCP receive from server

- String to data 6
  - We split **`LaserData`** into individual sub strings separated by spaces.

```
array<wchar_t>^ Space = { ' ' };
array<String^>^ StringArray = LaserData->Split(Space);
```

  –

  - This will make an array of references to strings. According to data description on LMS151 manual,
    - **`StartAngle`** is **`StringArray[23]`**,
    - **`Resolution`** is **`StringArray[24]`**,
    - Number of range data points (**`NumRanges`**) in **`StringArray[25]`** followed by
    - That many **`Range`** data in mm.

TCP receive from server

- String to data 6
- Getting **StartAngle, Resolution, NumRanges** and **Range**

```
double StartAngle = System::Convert::ToInt32(StringArray[23],16);
double Resolution = System::Convert::ToInt32(StringArray[24],16)/10000.0;
int NumRanges = System::Convert::ToInt32(StringArray[25],16);

array<double> ^Range  = gcnew array<double>(NumRangrs);
array<double> ^RangeX = gcnew array<double>(NumRanges);
array<double> ^RangeY = gcnew array<double>(NumRanges);

for(int i = 0; i < NumRanges; i++)
{
   Range[i] = System::Convert::ToInt32(StringArray[26+i],16);
   RangeX[i] = Range[i]*sin(i*Resolution);
   RangeY[i] = -Range[i]*cos(i*Resolution);
}
```

- This completes bi-directional Ethernet communication