Headquarters
Unitronics (1989) (R"G) Ltd.
Unitronics Building, Airport City
P.O.B. 300, Ben Gurion Airport, Israel 70100
Tel: + 972 (3) 977 8888 Fax: + 972 (3) 977 8877

Using the Unitronics Communication Driver for .Net

Using the Unitronics Communication Driver for .Net 2
1. Communication Channels 3
1.1. Serial channel 3
1.2. Ethernet channel 4
1.3. Ethernet Listener channel 5
2. Getting a PLC Object 6
3. Using the PLC Object 8
4. Using the PLC.ReadWrite 11

Headquarters
Unitronics (1989) (R"G) Ltd.
Unitronics Building, Airport City
P.O.B. 300, Ben Gurion Airport, Israel 70100
Tel: + 972 (3) 977 8888 Fax: + 972 (3) 977 8877

Using the Unitronics Communication Driver for .Net

The new .Net Com Drive supports both Vision PLCs and U90 PLCs.
It currently supports 3 ways/channels of communicating with the PLC:

- Serial channel
- Ethernet channel
- Ethernet Listener channel

With the first 2 channels, the PLC is the one that opens the communication, and in the third one the PC listens to a specific port and waits for a PLC to call.

When you want to perform an action on a PLC, you always do it by using a PLC object.
In order to get a PLC object, you need to define the way of communication first (Channel) and then use the `PLCFactory.GetPLC method` in order to get a PLC object back.

With both the first 2 channels you will either get a PLC object, or an exception if a connection could not be established with the PLC.

On the 3rd channel (The Ethernet Listener) you start listening to a port and you cannot predict when the PLC will call. This is why Events are being used.

UNITRONICS®

Headquarters
Unitronics (1989) (R"G) Ltd.
Unitronics Building, Airport City
P.O.B. 300, Ben Gurion Airport, Israel 70100
Tel: + 972 (3) 977 8888 Fax: + 972 (3) 977 8877

1. Communication Channels

1.1. Serial channel

The serial channel has several constructors. We suggest using either 1 of those 2:

```
Serial oSerial = new Serial(portName, baudRate, retry, timeOut, dataBits,
parity, stopBit);

Or

Serial oSerial = new Serial(portName, baudRate, retry, timeOut, dataBits,
parity, stopBit, autoDetectComParams);
```

This is because those 2 constructors cover most/all of the needed communication parameters.

The Timeout (just like with the other Channels) is in milliseconds. For having a timeout of 3 seconds, you need to write 3000.

On the second constructor the last argument is `autoDetectComParams` (which is true by default if you use other constructors).
This argument/property will let you decide if you want the serial channel to auto detect and synchronize the communication properties (Boundrate, databits, etc) with the PLC.

Please note that AutoDetect is disabled for RS485 (Any unit ID with a value of 64 to 127).

This is an example of code of creating a serial channel:

```
Serial serial = new Serial(SerialPortNames.COM1,BaudRate.BR115200,3,3000,
DataBits.DB8, System.IO.Ports.Parity.None,System.IO.Ports.StopBits.One);
```

Headquarters
Unitronics (1989) (R"G) Ltd.
Unitronics Building, Airport City
P.O.B. 300, Ben Gurion Airport, Israel 70100
Tel: + 972 (3) 977 8888 Fax: + 972 (3) 977 8877

1.2. Ethernet channel

The Ethernet channel has several constructors. We suggest using this one:

```
Ethernet ethernet = new Ethernet(string remoteIp, Int32 remotePort,
EthProtocol protocolType, int retry, int TimeOut);
```

This is an example of code of creating an Ethernet channel:

```
Ethernet ethernet = new Ethernet("192.168.1.100", 20256, EthProtocol.TCP, 3,
3000);
```

Headquarters
Unitronics (1989) (R"G) Ltd.
Unitronics Building, Airport City
P.O.B. 300, Ben Gurion Airport, Israel 70100
Tel: + 972 (3) 977 8888 Fax: + 972 (3) 977 8877

1.3. Ethernet Listener channel

The Ethernet listener channel currently has one constructor:

```
EthernetListener listener = new EthernetListener(Int32 localPort, int retry,
int TimeOut);
```

This is an example of code of creating an Ethernet channel:

```
EthernetListener listener = new EthernetListener(20256, 3, 3000);
```

2. Getting a PLC Object

To get a PLC object you need to use the `PLCFactory.GetPLC method.`

This method currently has 2 overloads:

```
public static PLC GetPLC(Channel channel, int unitId);
public static void GetPLC(EthernetListener ethernetListener);
```

The first overload accepts any channel and a unit ID. You can use either a Serial or an Ethernet channel (And in special cases EthernetListener).

The second overload, as you probably notice, doesn't return a PLC object.
This is because we need to wait for the PLC to call in order to get a PLC object.

On a serial or Ethernet channels we will do something like this:

```
PLC plc = PLCFactory.GetPLC(serial, 0);
```

It is important to wrap the GetPLC request in a try + catch since an exception will be thrown if the ComDrive will not be able to communicate with the PLC.
We suggest using UnitID = 0 whenever you are going to communicate / perform actions on the PLC which is connected directly to the PC.

On an Ethernet Listener:

```
EthernetListener listener = PLCFactory.GetChannel(20256);
if (listener == null)
    listener = new EthernetListener(20256, 3, 3000);

    listener.OnListenerConnectionAccepted += new
EthernetListener.ListenerConnectionAcceptedDelegate(OnConnect);
    listener.OnListenerConnectionClosed += new
EthernetListener.ListenerConnectionClosedDelegate(OnDisconnect);
            PLCFactory.GetPLC(listener);
```

This example ‘listens’ to port 20256.

Please note that first you go to the PLCFactory and request a listener by port.
This is because the Com Drive contains a List of channels that are in use or that were in use by the current AppDomain.

If you just create a listener and then add the "OnConnect" and "OnDisconnect" to its event, then there is a good chance that those functions will not be called, because the PLCFactory will use

the EthernetListener channel on the list (and you added those function to events of a channel that will not be used).

One important note is that the PLC you get in the event of the listener is the PLC on UnitID 0 (Direct Connection).
There is a chance that this **is** the PLC that you want to "talk" with, and that's fine, but what if you want to "talk", for example, with a PLC which is connected via CANbus to the caller (the PLC that called your PC)?

This is one of the special situations where you should use the regular/Synchronic GetPLC function.

For example:

```
private void OnConnect(PLC oPlc)
{
    Ethernet listener = oPlc.PLCChannel;
    try
    {
        plc = PLCFactory.GetPLC(listener, 41);
    }
    catch
    {
        System.Windows.Forms.MessageBox.Show("Could not communicate with the
PLC");
    }
}
```

The `OnListenerConnectionClosed` will be raised everything you call listener.Disconnect, plc.Disconnect, or the caller closed the connection.

The `OnListenerConnectionAccepted` will be raised only if a valid PLC object was created after someone called the PLC. If you do a Telnet 127.0.0.1 20256 then the event will not be raised since the caller is not a PLC.

Headquarters
Unitronics (1989) (R"G) Ltd.
Unitronics Building, Airport City
P.O.B. 300, Ben Gurion Airport, Israel 70100
Tel: + 972 (3) 977 8888 Fax: + 972 (3) 977 8877

3. Using the PLC Object

Every action that you want the PLC to carry must be made by using a PLC Object.
Every action you perform on a PLC object must be wrapped with try + catch. This provides protection from ComDrive Bugs, something you did wrong in your code, or in the event that the PLC does not reply (communication error for example).

There are several actions you can perform. For example:

plc.Run();
plc.Stop();
plc.Reset();
plc.Init();

plc.Abort();
plc.Disconnect();

plc.ReadWrite(`ref ReadWriteRequest[]`);

properties:
plc.Version
plc.UnitId
plc.RTC
plc.PLCChannel

With the first group of actions you can Run/Stop/Rest/Init the PLC.

The second group is meant for canceling actions you sent to the PLC ,or for closing the connection.

plc.Abort() will not Undo commands that you sent. It will just prevent all the commands that were not sent to the PLC to be sent.

You will see that you can request several ReadWrite actions from the PLC which may take more than few seconds. Since you can send them Asynchronously or create a new thread and send it throw it, then you will want a way to abort it.

Calling plc.Disconnect() (or event channel.Disconnect()) doesn't necessarily mean that the channel will close its connection.
This is not a bug, but it is actually made on purpose.

Headquarters
Unitronics (1989) (R"G) Ltd.
Unitronics Building, Airport City
P.O.B. 300, Ben Gurion Airport, Israel 70100
Tel: + 972 (3) 977 8888 Fax: + 972 (3) 977 8877

Consider one of the following situations:

1) You have several threads or several Asynchronous requests to the PLC that were made through the same PLC object, and you call plc.Disconnect().
2) There are several PLC objects which use the same channel and each of them requests something form a PLC using that channel.

Obviously, since other requests are being made by the same channel, then closing it for 1 thread might cause exceptions for the others.

The PLC will close the channel only of the queue of the channel is empty.

Furthermore, if you did manage to close the channel and 1 or more PLC objects send a request, the channel should be reopened (if it's Serial or Ethernet. A Listener cannot open a connection since it is not the one who initiates the connection).

Working Synchronously with 1 thread (Main GUI or class thread for that matter) and with 1 PLC object will make sure that connections will be closed when you ask it to. If you intend to work with several threads or several PLC objects, then you will need to make sure that you try to close the connection when nothing else try to use the channel.

Since Aborting requests is not instant (The current request need to be aborted, the PLC need to make sure that the PLC finished to reply, and that all the queued requests were removed) then plc.Abort() is being called Asynchronously and when abort is being finished then the following Event is being raised:

```
Plc.EventAbortCompleted
```

(So make sure to register to that event if you need to do something on the Abort Completed)

For example:
If you want to close that channel then send a plc.Abort() to each of the PLC object that use that channel. Since the Disconnect will not be ignored only if the channel is not busy sending or waiting for a reply then you need to register on the Abort Completed event and only if you got the event raised from PLC objects that used that channel then you can call the plc.Disconnect()

Headquarters
Unitronics (1989) (R"G) Ltd.
Unitronics Building, Airport City
P.O.B. 300, Ben Gurion Airport, Israel 70100
Tel: + 972 (3) 977 8888 Fax: + 972 (3) 977 8877

About the properties:
plc.Version
plc.UnitId
plc.RTC
plc.PLCChannel

Those properties are being taken from the PLC.
This means that if you write:
System.Diagnostics.Debug.Print(plc.Version.OSVersion);
Then the ComDrive will send a request to the PLC and request the version parameters.

This means that if you write:
System.Diagnostics.Debug.Print(plc.Version.OSVersion);
System.Diagnostics.Debug.Print(plc.Version.HWVersion);
System.Diagnostics.Debug.Print(plc.Version.PLCModel);
System.Diagnostics.Debug.Print(plc.Version.Boot);

Then you will cause the Com Drive to send 4 "Version" requests to the PLC instead of just 1!!

In my example, you can see the right solution:

```
PlcVersion version = plc.Version;
System.Diagnostics.Debug.Print(version.OSVersion);
System.Diagnostics.Debug.Print(version.HWVersion);
System.Diagnostics.Debug.Print(version.PLCModel);
System.Diagnostics.Debug.Print(version.Boot);
```

This way only 1 request was made to the PLC.

plc.RTC will return the time and date of the PLC in DateTime format.
Writing the following will set the date and time on the PLC:
plc.RTC = DateTime.Now;

The same thing applies to the UnitId. Getting its value will return the UnitId of the PLC, and setting its value changes the UnitId of the PLC.

4. Using the PLC.ReadWrite

One of the good features of this Communication driver is that you can request a bunch of requests in one call. The ComDrive will handle them separately, and will split the send data if needed (if the data exceeds the buffer for example).

For example, you can do this:

```
            Serial oSerial = new Serial(SerialPortNames.COM1,
BaudRate.BR115200, 3, 3000, DataBits.DB8, System.IO.Ports.Parity.None,
System.IO.Ports.StopBits.One);
            PLC oPLC;
            oPLC = PLCFactory.GetPLC(oSerial, 0);

            oPLC.SetExecuter(PLCExecuterType.ExecuterPartialBinaryMix);

            object[] values = new object[2048];

            for (int i = 0; i < 2048; i++)
            {
                values[i] = (object)i;
            }
            ReadWriteRequest[] readwrite = new ReadWriteRequest[5];

            readwrite[0] = new WriteOperands
            {
                NumberOfOperands = 2048,
                OperandType = OperandTypes.MI,
                StartAddress = 0,
                Values = values
            };

            readwrite[1] = new ReadOperands
            {
                NumberOfOperands = 200,
                OperandType = OperandTypes.ML,
                StartAddress = 0,
            };
            readwrite[2] = new ReadOperands
            {
                NumberOfOperands = 1024,
                OperandType = OperandTypes.MB,
                StartAddress = 0,
            };
            readwrite[3] = new ReadDataTables
            {
                StartAddress = 0,
                NumberOfBytesToReadInRow = 12,
                NumberOfRowsToRead = 5,
                RowSizeInBytes = 4
            };
            readwrite[4] = new ReadOperands
            {
                NumberOfOperands = 64,
```

```
    OperandType = OperandTypes.DW,
    StartAddress = 0,
};


try
{
    oPLC.ReadWrite(ref readwrite);
}
catch
{
    MessageBox.Show("Error");
}
```

As you can see, the code sends only 1 request in oPLC.ReadWrite. The request had several sub-requests in an Array. The code enables both reading and writing to and from operands and also reading and writing to and from DataTables.

NOTE: The order of the requests in the array may not be the same order in which they will be sent to the PLC.
If the order is important, then send several plc.ReadWrite commands.

(For example, you have an array of WriteOperand and then ReadOperand. In some cases the Read will be sent before the Write).

You can preserve the order in the following way:

```
oPLC.ReadWrite(ref WriteOperandRequest);
oPLC.ReadWrite(ref ReadOperandRequest);
```

Set/Get timer values, gives you a "bonus".

To request a Read of 2 timers:

```
readwrite[0] = new ReadOperands
{
    NumberOfOperands = 2,
    OperandType = OperandTypes.TimerPreset,
    StartAddress = 0,
};
```

In this case there is another property in the constructor that can be used: `TimerValueFormat`

This property can be used to define the format of the values that you send or receive from the PLC. The default format is TimeFormat.

In this format, each timer value uses an array of 4 fields, where each field represents the "section" in the timer. For example: the timer value: "11:22:34.56" will be delivered as a list of ushort:
list[0] = 11
list [1] = 22
list [2] = 34
list [3] = 56

If, on the other hand, you choose: SecondsFormat, then you will get the number of **Milliseconds**
In the previous timer example, this would result in: 4095456
((11*3600 + 22*60 + 34)*100 + 56)

```
readwrite[0] = new ReadOperands
{
    NumberOfOperands = 2,
    OperandType = OperandTypes.TimerPreset,
    StartAddress = 0,
    TimerValueFormat = TimerValueFormat.SecondsFormat
};
```

You will get the same result for `TimerValueFormat = TimerValueFormat.None`

To set Timer 0 to 12:34:56.78, for example, you can either use:

```
values[0] = (object)(new List<ushort> { 12, 34, 56, 78 });
ReadWriteRequest[] readwrite = new ReadWriteRequest[1];

readwrite[0] = new WriteOperands
{
    NumberOfOperands = 1,
    OperandType = OperandTypes.TimerPreset,
    StartAddress = 0,
    TimerValueFormat = TimerValueFormat.TimeFormat,
    Values = values

};
```

Or

```
values[0] = (object)(12*360000 + 34*6000 + 56*100 + 78);
ReadWriteRequest[] readwrite = new ReadWriteRequest[1];

readwrite[0] = new WriteOperands
{
    NumberOfOperands = 1,
    OperandType = OperandTypes.TimerPreset,
    StartAddress = 0,
```

```
            TimerValueFormat = TimerValueFormat.SecondsFormat,
            Values = values

        };
```

NOTE: When you write set a timer, make sure that the Value format match the TimerValueFormat that you choose, else you will get an exception.