DVT- Motoman Driver

Table of Contents

Introduction

This document contains information to help set up a Motoman-DVT- Moxa interface and test it. To implement the Motoman driver, DVT has created a set of memory registers reserved for Motoman global variables. Using special function calls in DVT's script tool, information from the inspections (e.g. position of an object) is placed into the registers were the XRC can access them through the Moxa Ethernet to Serial converter with a LOADV command using its standard RS-232 protocol.

The document describes the different data types used and the syntax of the functions needed to populate the registers in script. It then shows how the physical connection is achieved and how the Motoman driver is set and configured in FrameWork. A sample DVT script and Motoman job are also shown to illustrate the process. A reference to the syntax of the functions can also be found in the FrameWork help files.

Data Types Summary

The following table shows the Data types used in DVT scripts with their names and characteristics.

Data Types for Script			
Name	**Size**	**Description**	**Range**
Byte	8 bit	Integer - Unsigned	0 to 255
Short	16 bit	Integer - Signed	-32,768 to 32,767
Integer	32 bit	Integer - Signed	-2,147,483,648 to 2,147,483,647
Long	64 bit	Integer - Signed	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
Single	32 bit	Floating -Signed	-3.402823E38 to -1.401298E-45 for neg. values; 1.401298E-45 to 3.402823E38 for pos. values.
Double	64 bit	Floating -Signed	-1.79769313486231E308 to -4.94065645841247E-324 for neg. values; 4.94065645841247E-324 to 1.79769313486232E308 for pos. values.
String	Variable length	Byte array	No length Limitations

Table 1 Data types in DVT scripts

Table 2 shows how DVT and Motoman Data types relate. DVT has allocated enough space to hold all XRC global variables as shown in the last column. The variable numbers are used as arguments in the Motoman register management functions shown later. Notice that there may be some confusion for the user with the DVT-Motoman naming scheme (e.g. a Motoman Integer is really a DVT Short.) The function names hold a reference to Motoman names but the user must take care to pass the correct parameters in a DVT script.

Data Types for Script - Motoman					
DVT Type Names	**Motoman Type Names**	**Size**	**Description**	**Range**	**Variable Number**
Byte	Byte	8 bit	Integral - Unsigned	0 to 255	0 to 39
Short	Integer	16 bit	Integral - Signed	-32,768 to 32,767	0 to 39
Integer	Double Precision	32 bit	Integral - Signed	-2,147,483,648 to 2,147,483,647	0 to 39
Single	Real	32 bit	Floating -Signed	-3.402823E38 to -1.401298E-45 for neg. values; 1.401298E-45 to 3.402823E38 for pos. values.	0 to 39
	P-variables	24 Bytes	Group of 6 Real		0 to 127

Table 2 Motoman Global Data Types vs. DVT types

Register Management Functions for Motoman

MotoWriteByte()

Syntax:

```
MotoWriteByte(varNum as Byte, Value as Byte)
```

Returns:

0 = success

-1 = invalid Index or *varNum*

-2 = write error

-3 = other error

Example:

```
byte b;
b = blobselect.NumBlobs;
MotoWriteByte(15,b);
```

Note:

The Byte variable type includes values from 0 to 255.

The *varNum* parameter must be a number from 0 to 39.

MotoWriteInt()

Syntax:

MotoWriteInt(varNum as Byte, *Value* as Short*)*

Returns:

0 = success

-1 = invalid Index or *varNum*

-2 = write error

-3 = other error

Example:

```
short s;
s = blobselect.NumBlobs;
MotoWriteInt(15,s);
```

Note:

This function writes a DVT Short variable type to a Motoman Integer variable type.

The DVT Short variable type includes values from –32,768 to 32,767.

The *varNum* parameter must be a number from 0 to 39.

MotoWriteDouble()

Syntax:

MotoWriteDouble(varNum as Byte, *Value* as Long*)*

Returns:

0 = success

-1 = invalid Index or *varNum*

-2 = write error

-3 = other error

Example:

```
int d;
d = Sensor1.NumEdges;
MotoWriteDouble(25,d);
```

Note:

This function writes a DVT Integer variable type to a Motoman Double Precision integer double variable type.

The *varNum* parameter must be a number from 0 to 39.

MotoWriteReal()

Syntax:

```
MotoWriteReal(varNum as Byte, Value as Float)
```

Returns:

0 = success

-1 = invalid Index or *varNum*

-2 = write error

-3 = other error

Example:

```
float s;
s = FindCircle.Distance;
MotoWriteReal(29,s);
```

Note:

This function writes a DVT Float variable type to a Motoman Real variable type.

The *varNum* parameter must be a number from 0 to 39.

MotoWritePvar()

Syntax:

```
MotoWritePVar(varNum as Byte, x as Float,
y as Float, z as Float, thetax as Float,
thetay as Float, thetaz as Float)
```

Returns:

0 = success

-1 = invalid Index or *varNum*

-2 = write error

-3 = other error

Example:

```
float x, y, tx;
x = blobselect.BlobTransformedPoint.X[1]
y = blobselect.BlobTransformedPoint.Y[1]
tx = blobselect.BlobAngle[1]
MotoWritePVar(9,x,y,0,tx,0,0)
```

Note:

This function writes a series of DVT Float variable types to a Motoman P variable having a format of (x, y, z, θx, θy, θz) were each element is a Motoman Real data type.

The *varNum* parameter must be a number from 0 to 127.

Procedure for a Basic Test

The task is to write a P-variable within the DVT Legend Series and pass it to the controller. The sequence of events that need to happen is as follows:

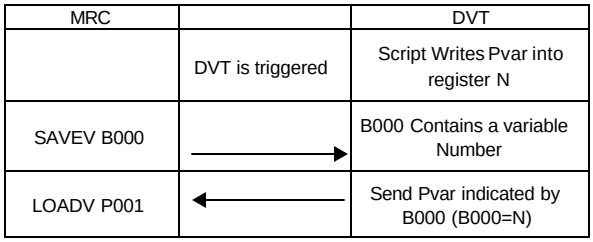


Figure 1 Sequence of Events in DVT XRC Communication

First, the script containing the Motoman Write functions is executed. This places the values from the inspection in the DVT registers reserved for Motoman. Next, the XRC controller sends a Byte variable to the Motoman driver running in the SmartImage Sensor. The content of this variable (in the range 0-255) is the DVT index of the variable that the XRC desires from the SmartImage Sensor. In the next operation, the XRC issues a LOADV command with a destination variable in the controller (P001 above). The DVT Motoman driver then sends a variable matching the requested data type and having an index as indicated by the byte variable (B000 above). The indexes of the variables is the XRC memory space and the DVT memory space do not have to be the same, but it is good programming practice to have them match. The example below will further clarify this process.

Set up of a Legend Camera with a Motoman XRC

Set up the DVT Camera, PC and Moxa unit through an Ethernet Hub or Switch. Then connect the Moxa serial port to the Motoman XRC RS 232 unit through a Null Modem Cable.

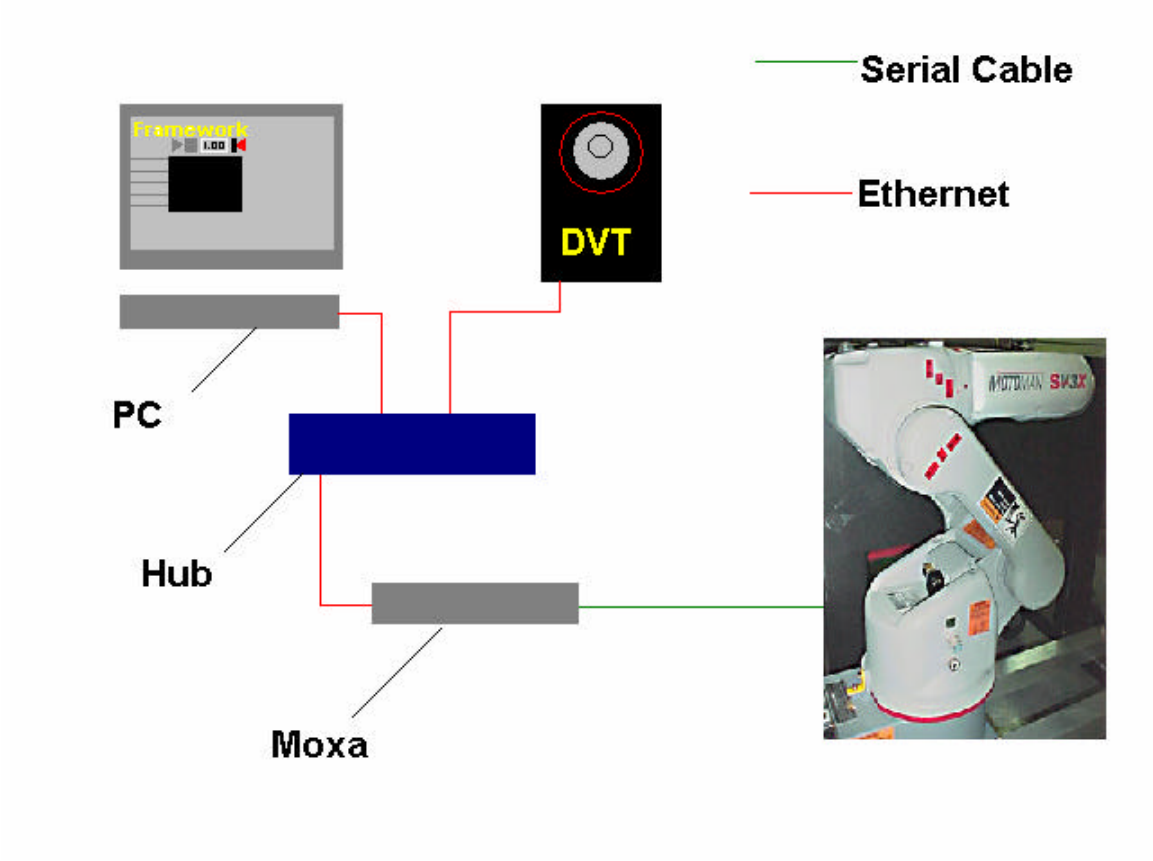


Figure 2 Setup of DVT, XRC Controller and Moxa Unit

1. Install Framework 2.4 or higher and download firmware to the Legend Camera. Establish a connection to the camera and set an IP and subnet mask. For this Example the camera IP address is 192.168.1.100 with a subnet of 255.255.0.0

2. The Moxa Unit is set up as shown below

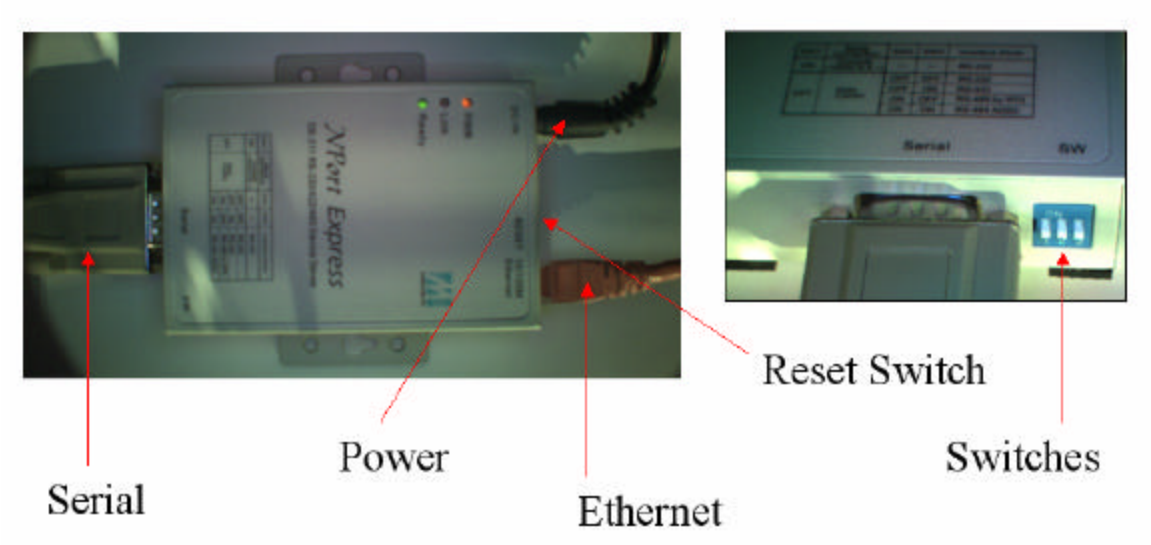
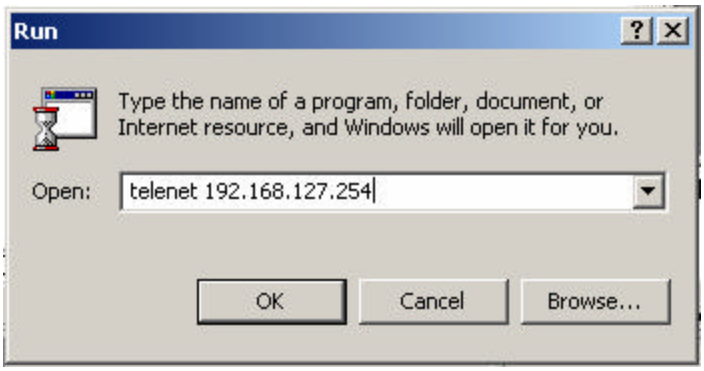


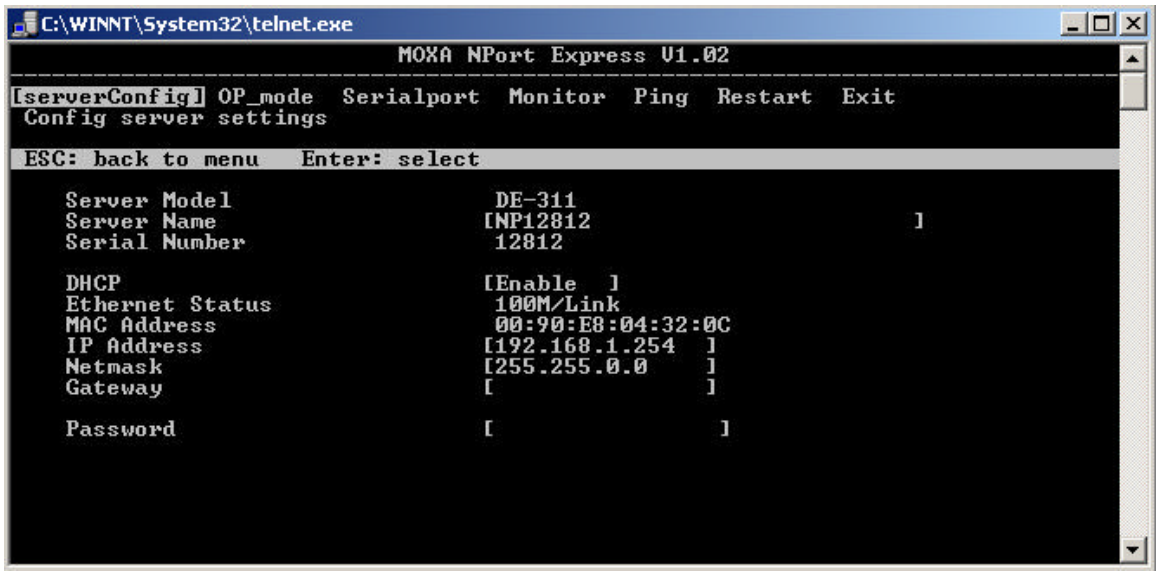
Figure 3 Moxa Hardware

3. Make sure that the three switches on the Moxa Unit are in the off position to communicate with the Moxa over ethernet. The default IP address is on the underside of the unit and is normally 192.168.127.254

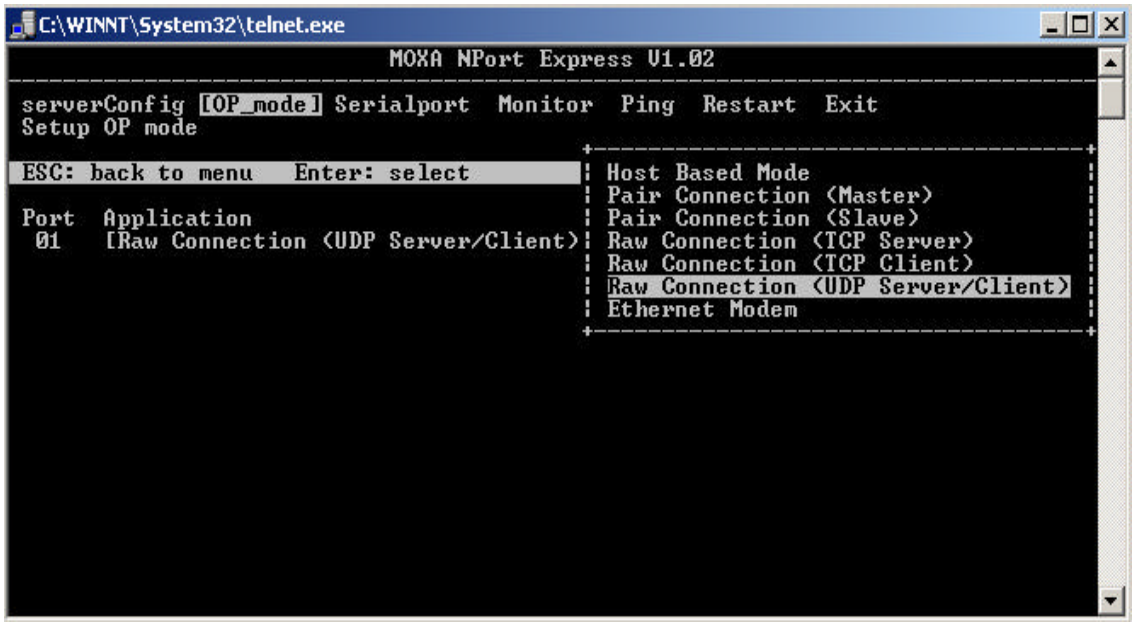
4. Telnet into the Moxa unit using the default IP address and then configure the connection type, serial and ethernet parameters.

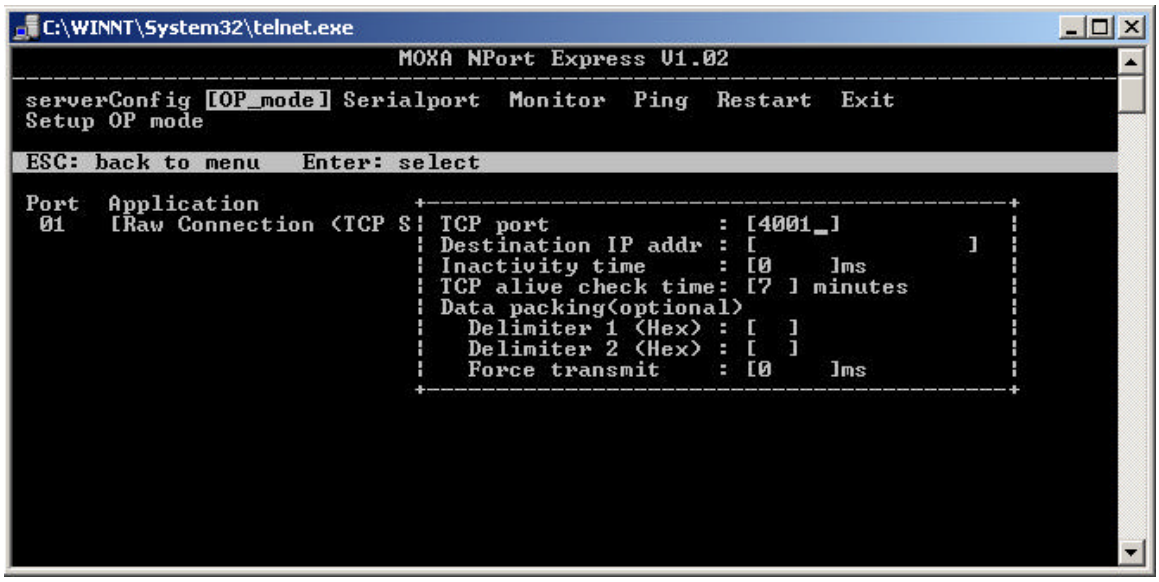


5. The Moxa needs to establish a Raw connection and must be on the same subnet at the DVT camera. It is also necessary to select a port for communications on the ethernet.

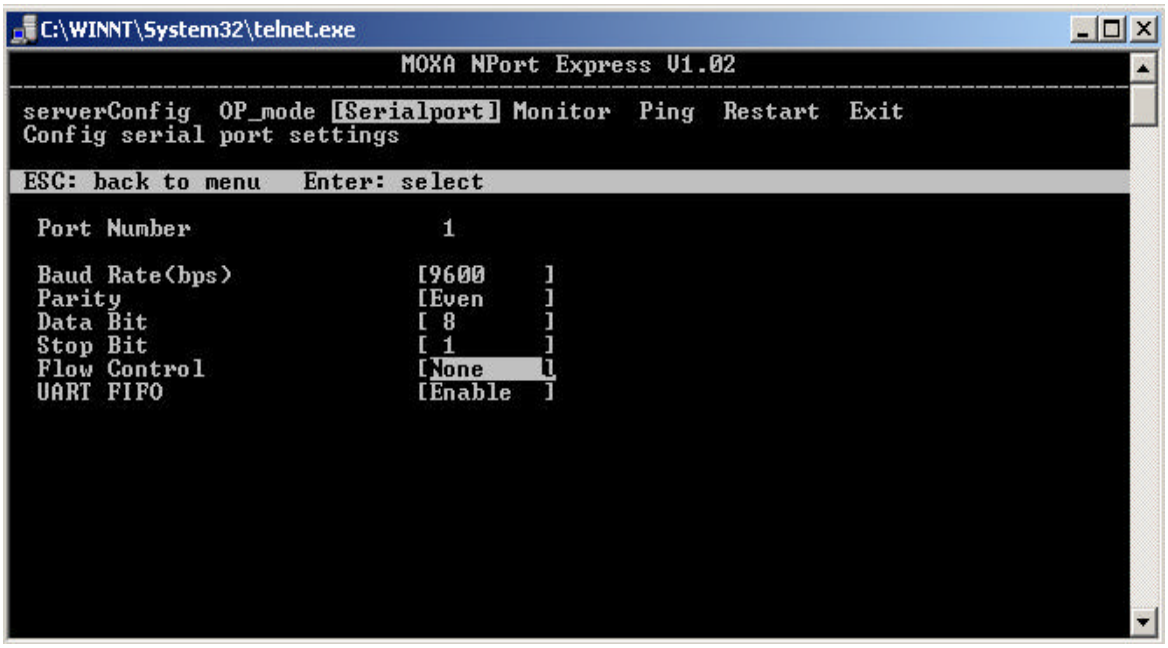


6. Setting the IP address and Subnet Mask





7. Setting the Connection type to Raw Connection (TCP Server/Client) and setting the port number



8. Set the above serial configuration to allow communications with the Motoman XRC with its default serial settings.

The Moxa unit is now configured and it is necessary to exit the setup menus. This will save the settings and Restart the unit. If you have configured the unit over the serial port it is important to change the switches to the off position. If you do not perform this operation no data will be received from the Moxa unit.

9. To ensure the correct communication protocol is used in the XRC, the standard port protocol must be set to BSC LIKE (parameter RS000 = 2). The BSC configuration parameters must be then set as follows:

•RS030 = 8 (8 data bits)

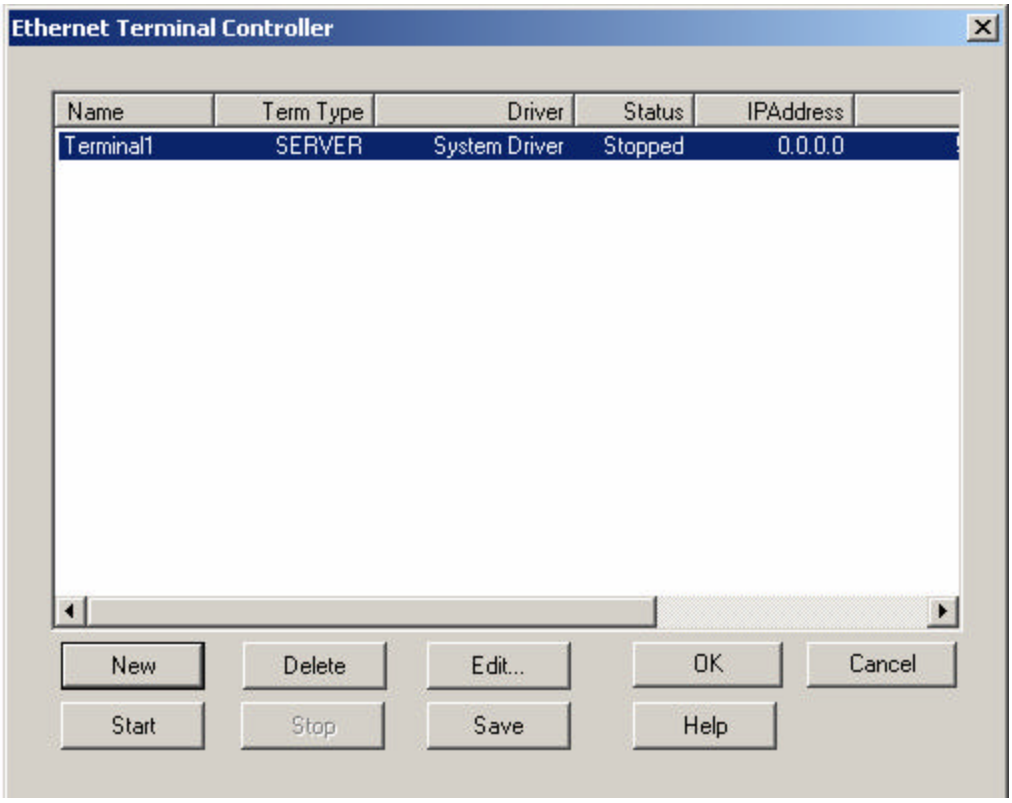
•RS031 = 0 (1 stop bit)

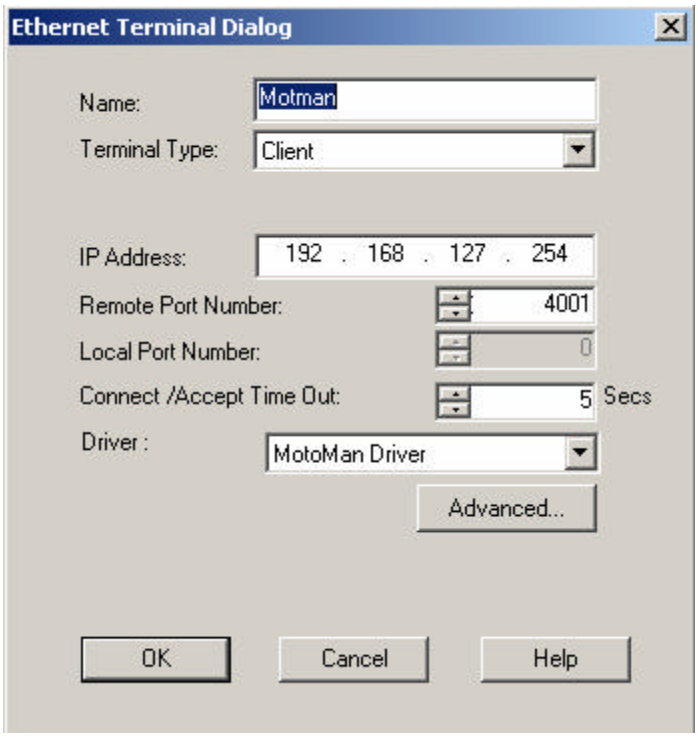
•RS032 = 2 (Even parity)

•RS033 = 7 (9600 baud)

The XRC must be set to local mode of operation (i.e. not in REMOTE mode) for communication with the DVT camera.

10. It is now necessary to set up the Ethernet Terminal Controller to Communicate between the Moxa and the DVT Legend series.





11. Set the Ethernet Terminal Controller as shown above, the DVT Camera is a Client and will send Data to the Moxa to the Moxa IP address in this case 192.168.127.254 (Default address of Moxa). Set the port number that is configured on the Moxa Unit and set the Driver to be the Motoman Driver.

12. On the Motoman controller, assign a value to a global byte variable (e.g. B000.) This value corresponds to the variable number desired from the DVT Camera. In this example, the content of byte variable B000 (value 7) causes the controller to load of Pvariable No. 7 from DVT system. On the XRC, set the target variable P007 to X, Y, Z, θx, θy, θz format.

13. The XRC Job used here was:

```
NOP
MOVJ
SAVEV B000 NWAIT
SWAIT
LOADV P007 NWAIT
SWAIT
END
```

14. The script used on the DVT Camera to populate a Pvariable with constant numbers is shown below. Notice the first argument, 7, which identifies the variable number to be written to. Since B000=7 on the controller, the data is transferred to P007 in the XRC upon execution of the LOADV command. It is good programming practice to use matching variable numbers on the XRC and the DVT sides although it is not required.

This script needs to run once in order for the values to be placed in the DVT registers. This is accomplished by triggering the system at least once. Under regular operation the XRC controller would be expected to trigger the DVT Camera with a digital I/O line (using DOUT statement in the job.) In real operation the script would pass variable sensor result to the MotoWritePvar function rather than constant values as in this example. A typical function call may look like:

```
MotoWritePVar(7, blobselect.BlobTransformedPoint.X[1],
              blobselect.BlobTransformedPoint.Y[1],0,0,
              blobselect.BlobAngle[1]
```

When variables different than Pvariables need to be transferred, basically follow the same procedure as above. The LOADV command changes to reflect the new type and the content of the byte variable that indicates the variable number is subject to the limits shown in the last column of Table 2.

15. After the script is executed, run the job on the controller and confirm the data has been transferred by displaying P007.

16. If you have a debug version of the firmware you can open a terminal and watch the details of data transfer. In order to watch a debug window open a Framework terminal. To achieve this Press CTRL-ALT-T while in FrameWork, a terminal window comes up. At the prompt type #Yd1048576. This action causes debug information to be output to the terminal window as the communication happens.

Running an XRC whilst Communicating

It is possible to communicate with an XRC Motoman Robot and a DVT camera whilst the XRC is performing other operations. This reduces cycle time and is an advantage of using the DVT camera. The example below shows two jobs being run in the XRC, one to move the robot and the second to read the data from the DVT camera.

MASTER JOB

```
NOP
PSTART JOB:MOTION SUB1    run the MOTION and COMMS jobs concurrently
PSTART JOB:COMMS SUB2
END
```

MOTION JOB

```
NOP
MOVJ                      move to standoff position
WAIT B001=1               wait for vision done flag
SET B002 1                handshake to acknowledge vision done flag
ADVINIT                   update XRC variables
MOVL P050 V=300.00        move to part
DOUT OT#(1) ON            grip part
SET B002 0                set flag to zero
MOVL
MOVJ                      move to part place position
DOUT OT#(1) ON            ungrip part
MOVJ                      move to start position
END
```

COMMS JOB

NOP

SET B001 0	*set flag to zero*
WAIT IN#(1)=ON	*wait for part present*
PULSE OT#(2)	*trigger camera*
SET B000 50	*set variable number to be written to*
SAVEV B000 NWAIT	
SWAIT	
LOADV P050 NWAIT	*load vision result into position variable 50*
SWAIT	
SET B001 1	set *flag to indicate to MOTION job that vision done*
ADVINIT	*update XRC variables*
WAIT B002 1	*wait for handshake to acknowledge vision done flag*

END

The DVT tools used here were positional tools to give X, Y and Theta angles. The DVT was controlled through a background script to allow accurate timing of the positional data relative to the inspection process and timing of a background script. Some simple I/O handshaking was implemented to tell the XRC when new positional data was available and the inspection had finished. The foreground script used and background script used, are shown below.

```
class Script
{
    public void inspect()
    {
        // Add your Code here
        float x , y, theta;
        x=selector.BlobPosition.X[0];
        y=selector.BlobPosition.Y[0];
        theta=selector.BlobAngle[0];
        this.Position.X = x;
        this.Position.Y = y;
        this.Angle = theta;
        RegisterWriteFloat( 100,x );
        RegisterWriteFloat( 200,y );
        RegisterWriteFloat( 300,theta );


    }
}
```

Foreground Script

File Edit Search Script

```
class parallelcoms
{
    public static void main()
    {

        sleep (5000); //  Allows the camera time to boot up
        int result;
        float x,y,theta, Return;
        long b=1;
        long Bit=0;

        while (true)
        {
            result = WaitOnInput(20,-1);// Waits for a trigger signal

            if(result == 0)

            {
            Image img;
            Product P;
            P=GetProduct("Product1");
            img = new Image();
            img.Acquire();
            img.Inspect(P);
            img.Save();
            sleep(200); // Allows the inspection to take place before reading Registers
            x=RegisterReadFloat( 100);
            y=RegisterReadFloat( 200);
            theta=RegisterReadFloat( 300);
            Return=MotoWritePVar( 50, x,y ,theta ,0 ,0 ,0 );
            //Sends the PVar data to the XRC

            SetOutputs((b<<24) ,0);//Sets an output to tell XRC DVT  job finished
            sleep(10);
            SetOutputs(0, (b<<24) );
            sleep (100);



            }
        }

    }
}
```

Background Script