Closer to Real, ROBOTIS

Dynamixel
DX-113, DX-116, DX-117

Contents

1. Dynamixel DX-Series

1-1. Overview and Characteristics of the DX-Series

Dynamixel DX-Series
The Dynamixel robot actuator is a smart, modular actuator that incorporates a gear reducer and a control circuitry with networking functionality, all in a single package. Despite its compact size, it can produce large torque and is made with special materials to provide the necessary strength and structural resilience to withstand large external forces. It also has the ability to detect and act upon internal conditions such as changes in internal temperature or supply voltage. There are three models (DX-113, DX-116, and DX-117) in the DX series of the Dynamixel robot actuator family and they have many advantages over similar products.

Precision Control
Position and speed can be controlled with a resolution of 1024 steps.

Compliance Driving
The degree of compliance can be adjusted and specified in controlling position.

Feedback
Feedback for angular position, angular velocity, and load torque are available.

Alarm System
The Dynamixel series robot actuator can alert the user when parameters deviate from user defined ranges (e.g. internal temperature, torque, voltage, etc) and can also appropriately handle the problems by itself.

Communication
Wiring is easy with daisy chain connection, and it support communication speeds up to 1M BPS.

High-performance Motors
Models DX-116 and DX-117 use the RE-MAX Series Coreless DC Motors, which are the top of the line Swiss Maxon motors, allowing them to produce large output torques and high accelerations.

Distributed Control
The actuation schedule can be set with a single command packet, thus enabling the main processor to control many Dynamixel units even with very few resources.

Engineering Plastic
The main body of the unit is made with high quality engineering plastic which enables it to handle high torque loads.

Metal Gear   All gears are made with metal to ensure durability.

Axis Bearing   A bearing is used at the final axis to ensure no efficiency degradation with high external loads on the output shaft.

Status LED   The LED can indicate the error status to the user.

1-2. Main Specifications

	DX-116		DX-117		DX-113
Weight(g)	66		66		58
Gear Reduction Ratio	142.5		192.6		192.6
Input Voltage	12	16	12	16	12
Final Max Holding Torque(kgf.cm)	21.38	28.50	28.89	38.52	10.20
Sec/60degree	0.127	0.095	0.172	0.129	0.150

Resolution   0.35°
Operating Angle   300°
Voltage   DX116,117 : 12V~16V(Recommended voltage: 14.4V)
                            DX113 : 12V
Max. Current   1200mA
Operating Temp.   -5℃ ~ +85℃
Command Signal   Digital Packet
Protocol Type   Half duplex Asynchronous Serial Communication (8bit,1stop,No Parity)
Link (Physical)   RS 485 Multi Drop(daisy chain type Connector)
ID   254 ID (0~253)
Communication Speed   7343bps ~ 1 Mbps
Feedback   Position, Temperature, Load, Input Voltage, etc.
Material   Full Metal Gear, Engineering Plastic Body
Motor   Swiss MAXON Motor (DX-116, DX-117). DX-113 uses a cored motor

2. Dynamixel Operation

2-1. Mechanical Assembly

Follow the figure below for the mechanical assembly of the Dynamixel actuator.

Nut(8EA) Horn Screw for Horn

Screw for mount(8EA)
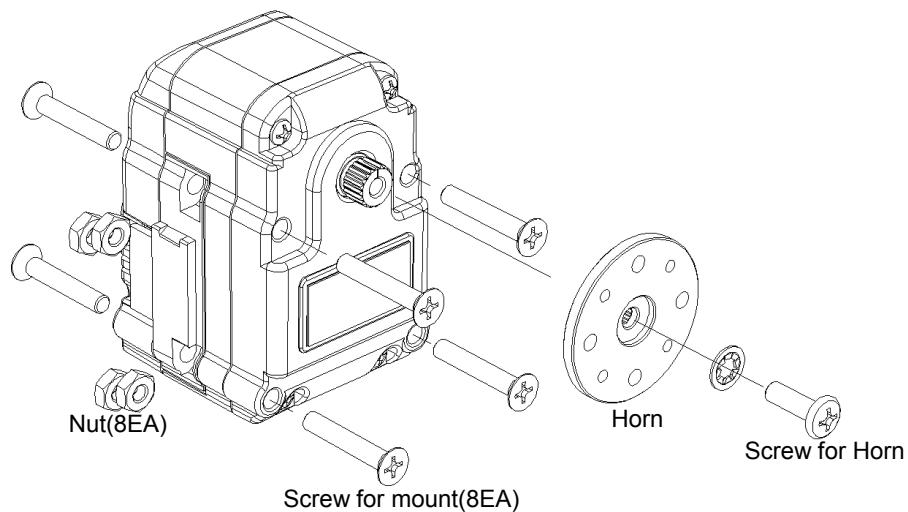
The 8 sets of screws and nuts are only used for attaching the Dynamixel actuator to other parts.

2-2. Connector Assembly

Assemble the connectors as shown below. Attach the wires to the terminals using the correct crimping tool. If you do not have access to a crimping tool, solder the terminals to the wires to ensure that they do not become loose during operation.

1) Striping
2) Inserting
3) Forming
4) Formed Wire
5) Assembling
6) Complete
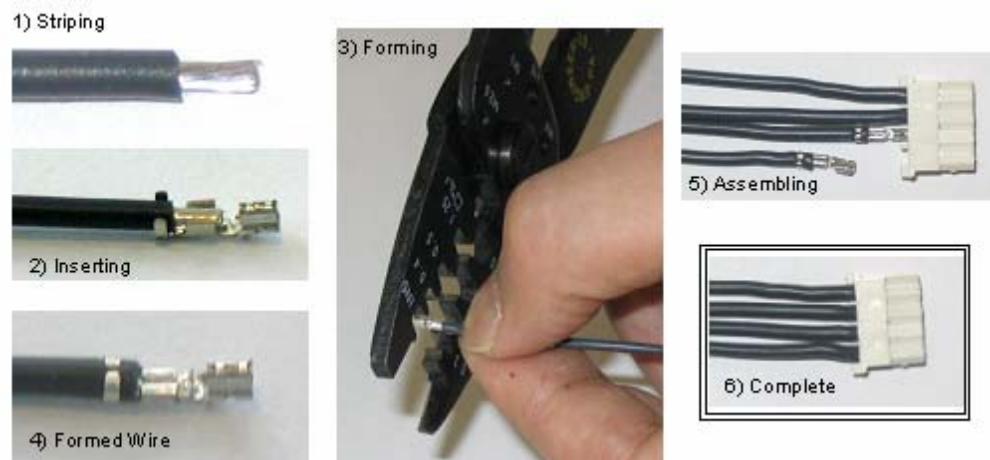
2-3. Dynamixel Wiring

Pin Assignment

The connector pin assignments are as the following. The two connectors on the Dynamixel actuator are internally connected to each other.

Pin 1 : GND
Pin 2 : +12V~18V
Pin 3 : D+ (RS485 Signal)
Pin 4 : D- (RS485 Signal)
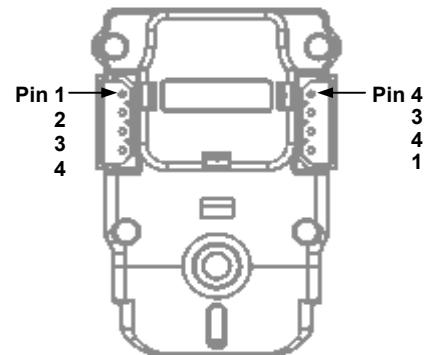
Wire Link

Connect the pins to pins that have the same number as shown below.
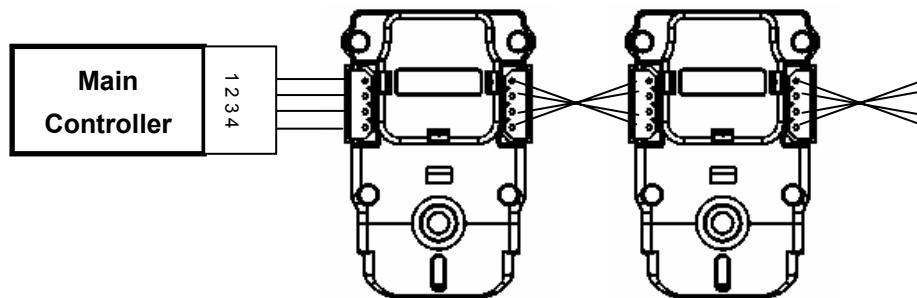
Main Controller

To operate the Dynamixel actuators, the main controller must support RS485. You can design and build your own controller, but the use of the CM-2 Dynamixel controller board is recommended.

PC LINK

A PC can be used to control the Dynamixel actuator via the CM-2 controller.
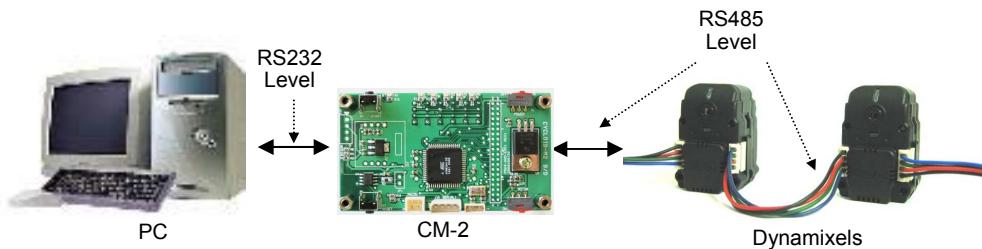
Stand Alone

The CM-2 board can be directly mounted on a robot that is built with Dynamixel actuators.

CM-2 Board on Robot

For usage details, please refer to the CM-2 manual.

Connection to UART

To control the Dynamixel actuators, the main controller needs to convert its UART signals from TTL level to RS485 level. The recommended circuit diagram for this is shown below.
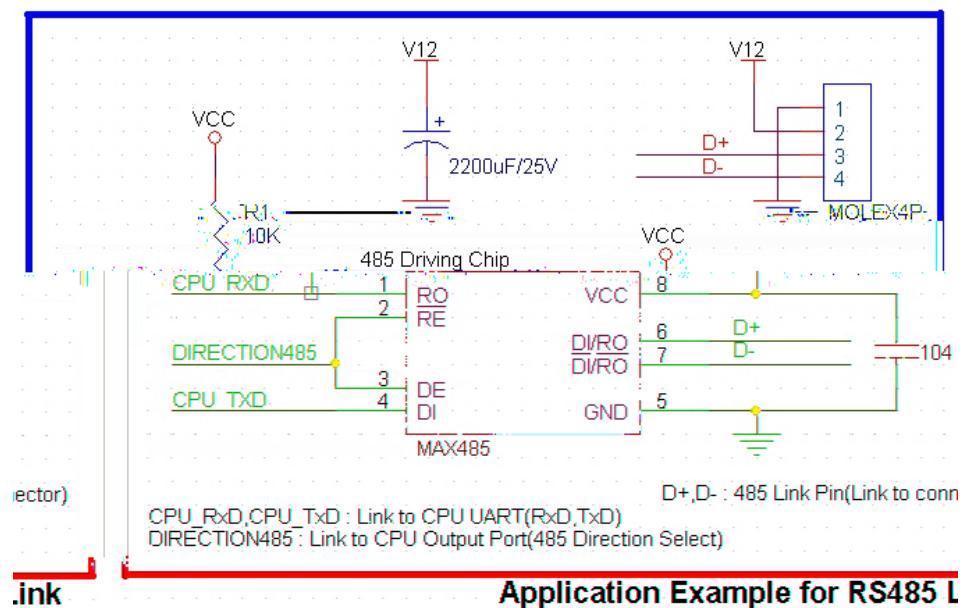
The direction of data signals on the TTL level TxD and RxD depends on the DIRECTION485 level as the following.

• When the DIRECTION485 level is High: the TxD signal is outputted as D+, D-
• When the DIRECTION485 level is Low: the D+, D- signal is inputted to RxD

RS485

The communication protocol used by the Dynamixel actuator, RS485 (IEEE485), uses the multi-drop method of connecting multiple terminals on a single node. Thus a protocol that does not allow multiple transmissions at the same time should be maintained on a RS485 network.

[RS485 Multi Drop Link]
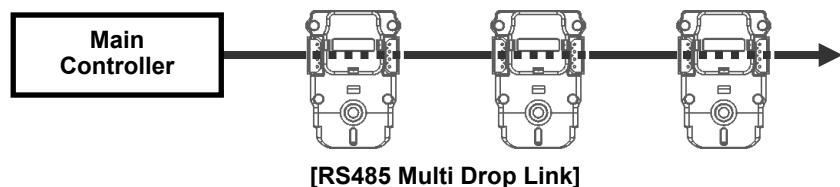
Note

Please ensure that the pin assignments are correct when connecting the Dynamixel actuators. Check the current consumption after the wiring is completed. The current consumption of a single Dynamixel actuator unit in standby mode should be no larger than 50mA.

Connection Status Verification

When power is applied to the Dynamixel actuator, the LED blinks twice to confirm its connection.

Inspection

If the above operation was not successful, check the connector pin assignment and the voltage/current limit of the power supply.

3. Communication Protocol

3-1. Communication Overview

Packet   The Main Controller communicates with the Dynamixel by sending and receiving data packets. There are two types of packets, the Instruction Packet (Main Controller to Dynamixel) and the Status Packet. (Dynamixel to Main Controller)
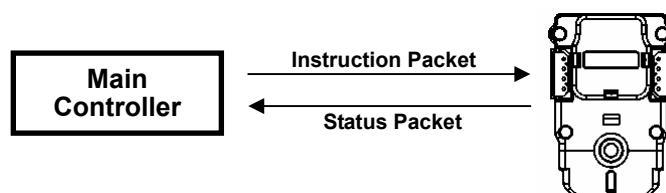
Communication   For the system connection below, if the main controller sends an instruction packet with the ID set to N, only the Dynamixel with this ID value will return its respective status packet and perform the required instruction.
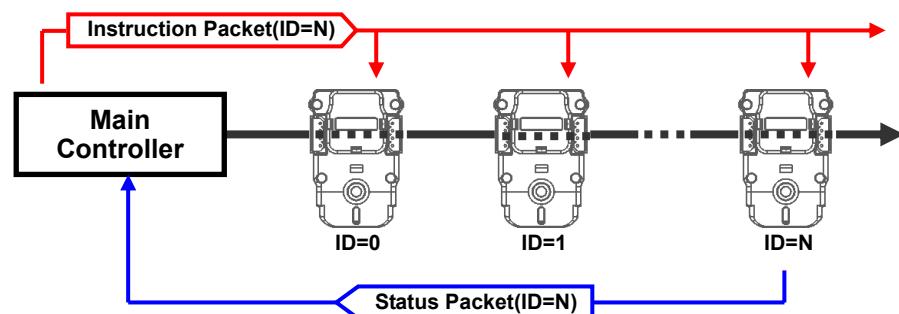
Unique ID   Communication problems will arise if multiple Dynamixel's have the same ID value. This will cause multiple packets to be sent simultaneously resulting in packet collisions. It is imperative that ID values are unique within each data network.

Protocol   The Asynchronous Serial Communication word consists of 8 bits, 1 Stop bit and no parity.

3-2. Instruction Packet

The structure of the Instruction Packet is as follows:

Instruction Packet   OXFF 0XFF ID LENGTH INSTRUCTION PARAMETER1 …PARAMETER N CHECK SUM

The packet byte definitions are as follows.

0XFF 0XFF   Two 0XFF bytes indicate the start of an incoming packet.

ID   Unique ID of a Dynamixel. The ID can range from 0X00 to 0XFD (254 IDs are available)

Broadcasting ID   ID ID 0XFE is the Broadcast ID which is assigned to all of the connected Dynamixel's. Status packets will not be returned with a broadcasting ID.

LENGTH   The length of the Status Packet. The value is "Parameter number (N) + 2"

INSTRUCTION   The instruction for the Dynamixel to perform.

PARAMETER0…N   Used if there is additional information to be sent other than the Instruction.

CHECK SUM   The calculation method for the 'Check Sum' is as follows:
Check Sum = ~(ID + Length + Instruction + Parameter1 + … Parameter N)
If the calculated value is bigger than 255, the lower byte becomes the checksum.
~ represents the Not or complement operation

3-3. Status Packet

The Status Packet is the response packet from the Dynamixel to the Main Controller after receiving an instruction packet. The structure of Status Packet is as follows :

OXFF 0XFF ID LENGTH ERROR PARAMETER1 PARAMETER2…PARAMETER N CHECK SUM

The meaning of each byte within the packet is as follows :

0XFF 0XFF   Two 0XFF bytes indicate the start of a packet.

ID   ID of the Dynamixel which is returning the packet.

LENGTH   The length of the Status Packet. The value is "Parameter number (N) + 2".

ERROR   Dynamixel communication error flags. The meaning of each bit is as follows:

Bit	Name	Details
Bit 7	0	-
Bit 6	Instruction Error	Set to 1 if an undefined instruction is given without the reg_write instruction.
Bit 5	Overload Error	Set to 1 if the specified torque can't control the load.
Bit 4	Checksum Error	Set to 1 if the checksum of the intruction packet is incorrect
Bit 3	Range Error	Set to 1 if the instruction is out of the usage range.
Bit 2	Overheating Error	Set as 1 if the internal temperature of Dynamixel is out of the operative range as set in the control table.
Bit 1	Angle Limit Error	Set as 1 if the goal position is set outside of the range between CW Angle Limit and CCW Angle Limit
Bit 0	Input Voltage Error	Set to 1 if the voltage is out of the operative range set in the control table.

PARAMETER0…N   Used when additional information is required.

CHECK SUM   SUM Calculation method of 'Check Sum'is as follows:

Check Sum = ~(ID + Length + Instruction + Parameter1 + … Parameter N)

If the calculated value is bigger than 255, the lower byte becomes the checksum.

~ represents the Not or complement operation

3-4. Control Table

Address	Item	Access	Initial Value
0(0X00)	Model Number(L)	RD	116(0x74)
1(0X01)	Model Number(H)	RD	0(0x00)
2(0X02)	Version of Firmware	RD	?
3(0X03)	ID	RD,WR	1(0x01)
4(0X04)	Baud Rate	RD,WR	34(0x22)
5(0X05)	Return Delay Time	RD,WR	250(0xFA)
6(0X06)	CW Angle Limit(L)	RD,WR	0(0x00)
7(0X07)	CW Angle Limit(H)	RD,WR	0(0x00)
8(0X08)	CCW Angle Limit(L)	RD,WR	255(0xFF)
9(0X09)	CCW Angle Limit(H)	RD,WR	3(0x03)
10(0x0A)	(Reserved)	-	0(0x00)
11(0X0B)	the Highest Limit Temperature	RD,WR	85(0x55)
12(0X0C)	the Lowest Limit Voltage	RD,WR	60(0X3C)
13(0X0D)	the Highest Limit Voltage	RD,WR	190(0xBE)
14(0X0E)	Max Torque(L)	RD,WR	255(0XFF)
15(0X0F)	Max Torque(H)	RD,WR	3(0x03)
16(0X10)	Status Return Level	RD,WR	2(0x02)
17(0X11)	Alarm LED	RD,WR	4(0x04)
18(0X12)	Alarm Shutdown	RD,WR	4(0x04)
19(0X13)	(Reserved)	RD,WR	0(0x00)
20(0X14)	Down Calibration(L)	RD	?
21(0X15)	Down Calibration(H)	RD	?
22(0X16)	Up Calibration(L)	RD	?
23(0X17)	Up Calibration(H)	RD	?
24(0X18)	Torque Enable	RD,WR	0(0x00)
25(0X19)	LED	RD,WR	0(0x00)
26(0X1A)	CW Compliance Margin	RD,WR	0(0x00)
27(0X1B)	CCW Compliance Margin	RD,WR	0(0x00)
28(0X1C)	CW Compliance Slope	RD,WR	32(0x20)
29(0X1D)	CCW Compliance Slope	RD,WR	32(0x20)
30(0X1E)	Goal Position(L)	RD,WR	[Addr36]value
31(0X1F)	Goal Position(H)	RD,WR	[Addr37]value
32(0X20)	Moving Speed(L)	RD,WR	0
33(0X21)	Moving Speed(H)	RD,WR	0
34(0X22)	Torque Limit(L)	RD,WR	[Addr14] value
35(0X23)	Torque Limit(H)	RD,WR	[Addr15] value
36(0X24)	Present Position(L)	RD	?
37(0X25)	Present Position(H)	RD	?
38(0X26)	Present Speed(L)	RD	?
39(0X27)	Present Speed(H)	RD	?
40(0X28)	Present Load(L)	RD	?
41(0X29)	Present Load(H)	RD	?
42(0X2A)	Present Voltage	RD	?
43(0X2B)	Present Temperature	RD	?
44(0X2C)	Registered Instruction	RD,WR	0(0x00)
45(0X2D)	(Reserved)	-	0(0x00)
46[0x2E]	Moving	RD	0(0x00)
47[0x2F]	Lock	RD,WR	0(0x00)
48[0x30)	Punch(L)	RD,WR	32(0x20)
49[0x31)	Punch(H)	RD,WR	0(0x00)

EEPROM Area

RAM Area

Control Table

The Control Table consists of data for conditions and movement of the Dynamixel. By writing the values in the control table, you can move the Dynamixel and detect the condition of the Dynamixel.

RAM and EEPROM

The data values for the RAM Area will be set to the default initial values on power on. The data values for the EEPROM Area are non-volatile and will be available next power on.

Initial Value

The Initial Value column of the control table shows the Factory Default Values for the case of EEPROM Area Data. For the RAM Area Data, the initial value column gives the power on data values.

Please note the following meanings for data assigned to each address in the control table.

Address 0x00,0x01

Model Number.　In the case of the DX-116, the value is 0X0074(116).

Address 0x02

Firmware Version.

Address 0x03

ID. Unique ID number to identify the Dynamixel. Different ID's are required to be assigned to "linked" Dynamixels.

Address 0x04

Baud Rate. Determines the Communication Speed. The Calculation method is:
Speed(BPS) = 2000000/(Address4+1)

Data Value as per Major Baud Rate

Adress4	BPS Set	Target BPS	Error
1	1000000.0	1000000.0	0.000%
3	500000.0	500000.0	0.000%
4	400000.0	400000.0	0.000%
7	250000.0	250000.0	0.000%
9	200000.0	200000.0	0.000%
16	117647.1	115200.0	-2.124%
34	57142.9	57600.0	0.794%
103	19230.8	19200.0	-0.160%
207	9615.4	9600.0	-0.160%

Note

A maximum Baud Rate error of 3% is within the UART communication tolerance.

Address 0x05   **Return Delay Time.** The time taken after sending the Instruction Packet, to receive the requested Status Packet. The delay time is given by 2uSec *Address5 value.

Address 0x06,0x07,0x08,0x09

Operating Angle Limit. Set the operating angle to restrict the Dynamixel's angular range. The Goal Position needs to be within the range of:-
CW Angle Limit <= Goal Position <= CCW Angle Limit
An Angle Limit Error will occur if this relationship is not satisfied.

Address 0x0B   **the Highest Limit Temperature.** The upper limit of the Dynamixel's operative temperature. If the Dynamixel's internal temperature is higher than this value, an Over Heating Error Bit (Bit 2 of the Status Packet) will be set. An alarm will be set in Address 17,18. The values are in Degrees Celsius.

Address 0x0C,0x0D   **the Lowest (Highest) Limit Voltage.** Setting the operative upper and lower limits of the Dynamixel's voltages.
If the present voltage (Address42) is out of the specified range, a Voltage Range Error bit will be set in the Status Packet and an alarm executed will be set in Address's 17,18. The values are 10 times the actual voltages. For example, if the Address 12 value is 80, then the lower voltage limit is set to 8V.

Address 0x0E,0x0F, 0x22,0x23

Max Torque. The max torque output for the Dynamixel. When it is set to '0', the Dynamixel enters a Torque Free Run condition. The Max Torque (Torque Limit) is assigned to EEPROM (Address 0X0E,0x0F) and RAM (Address 0x22,0x23) and a power on condition will copy EEPROM values to RAM. The torque of a Dynamixel is limited by (Address0x22,0x23) of RAM.

Address 0X10   **Status Return Level.** To determine whether the Dynamixel will return the Status Packet after the transmission of an Instruction Packet.

Address16	Return of Status Packet
0	Do net respond to any instruction
1	Respond only to READ_DATA instruction
2	Respond to all instructions

In the case of an instruction which uses the Broadcast ID (0XFE), regardless of the Address 0x10 value, the Status Packet will not be returned.

Address 0X11   **Alarm LED.** When an Error occurs, if the corresponding Bit is set to 1, then the LED blinks.

Bit	Function
Bit 7	0
Bit 6	If set to 1, LED blinks when Ins truction Error occurs
Bit 5	If set to 1, LED blinks when Overload Error occurs
Bit 4	If set to 1, LED blinks when Checksum Error occurs
Bit 3	If set to 1, LED blinks when Range Error occurs
Bit 2	If set to 1, LED blinks when Overheating Error occurs
Bit 1	If set to 1, LED blinks when Angle Limit Error occurs
Bit 0	If set to 1, LED blinks when Input Voltage Error occurs

This function operates as the logical "OR"ing of all set bits. For example, when the register is set to 0X05, the LED will blink when a Voltage Error occurs or when an Overheating Error occurs. Upon returning to a normal condition from an error state, the LED stops blinking after 2 seconds.

Address 0X12   **Alarm Shutdown.** When an Error occurs, if the corresponding Bit is set to a 1, then the Dynamixel will shut down (Torque off).

Bit	Function
Bit 7	0
Bit 6	If set to 1, torque off when Ins truction Error occurs
Bit 5	If set to 1, torque off when Overload Error occurs
Bit 4	If set to 1, torque off when Checksum Error occurs
Bit 3	If set to 1, torque off when Range Error occurs
Bit 2	If set to 1, torque off when Overheating Error occurs
Bit 1	If set to 1, torque off when Angle Limit Error occurs
Bit 0	If set to 1, torque off when Input Voltage Error occurs

This function operates as the logical "OR"ing of all set bits. However, unlike the Alarm LED, after returning to a normal condition, it maintains a torque off status. To remove this restriction, Torque Enable (Address0X18) is required to be set to 1.

Address 0x14~0x17   **Calibration.** Data used for compensating for the differences between Robotis products. Users cannot change this area.

From Address 0x18 in the RAM area.

Address 0x18   **Torque Enable.** When power is first applied the Dynamixel enters the Torque Free Run condition. To allow torque to be applied Address 0x18 must be set to 1. (Torque Enabled Condition)

Address 0x19   **LED**.   LED is on when set to 1 and LED is off if set to 0.

Address 0x1A~0x1D   **Compliance Margin and Slope.** The Dynamixel controls Compliance by setting the Margin and Slope. If used well Compliance will absorb the shocks. The following graph demonstrates the use of Compliance values (length of A,B,C & D) relative to Position Error and applied torque.

> **A : CCW Compliance Slope(Address0x1D)**
> **B : CCW Compliance Margin(Address0x1B)**
> **C : CW Compliance Margin(Address0x1A)**
> **D : CW Compliance Slope (Address0x1C)**
> **E : Punch(Address0x30,31)**
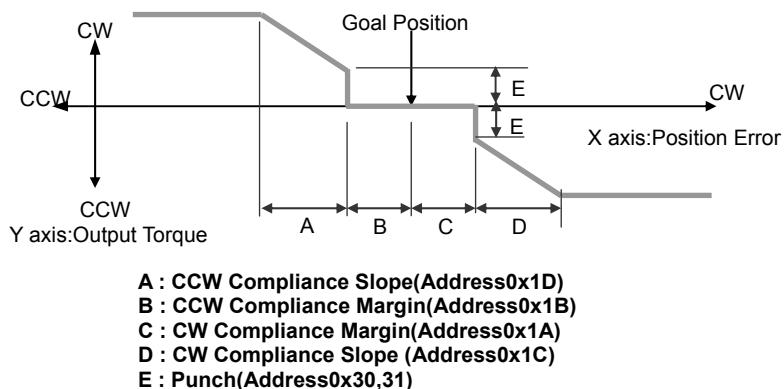
Address 0X1E,0x1F   **Goal Position.** Requested Angular Position for the Dynamixel to move to. If this is set to 0x3ff, then the goal position will be 300°.
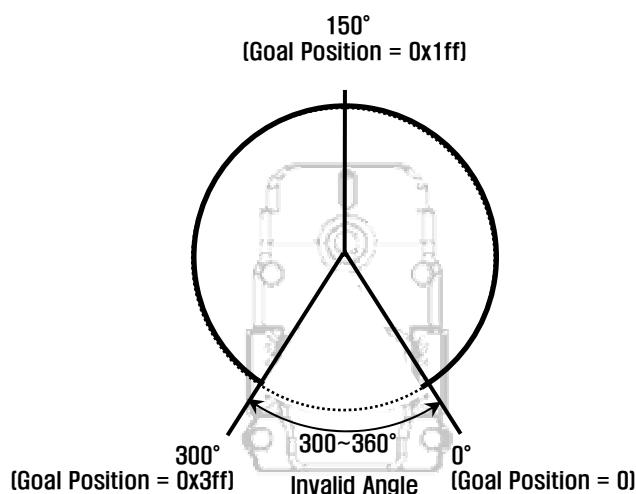
Address 0x20,0x21   **Moving Speed.** The angular speed to move to the Goal Position. If set to the maximum values of 0x3ff, it moves at 70RPM.

Address 0x24,0x25   **Present Position.** Current position of the Dynamixel.

Address 0x26,0x27   **Present Speed.** Current Speed of the Dynamixel

Address 0x28,0x29   **Present Load.** Load size on the Dynamixel in action. Bit 10 is the direction of the load.

BIT	15~11	10	9	8	7	6	5	4	3	2	1	0
Value	0	Load Direction	Load Value									

Load Direction = 0 : CCW Load,   Load Direction = 1: CW Load

Address 0x2A   **Present Voltage.** The voltage applied to the Dynamixel. The value is 10 times the actual voltage. For example, 10V is read as 100(0x64).

Address 0x2B   **Present Temperature.** Current internal Dynamixel temperature (Degrees Celsius).

Address 0x2C   **Registered Instruction.** Set to 1 when a REG_WRITE instruction is made. After an Action instruction and an action it is reset to 0.

Address 0x2E   **Moving.** Set to 1 when the Dynamixel moves by its own power.

Address 0x2F   **Lock.** If set to 1, only Address 0x18 ~ Address 0x23 can be written to. Other areas are not permitted. Once locked, it can only be unlocked by powering down.

Address 0x30,0x31   **Punch.** Minimum current being supplied to the motor during an action. The minimum value is 0x20 and the maximum value as 0x3ff.

Range

Each Register has an operative range. Write instructions made outside of these ranges will return an error. The following table summarises the data range for each register. 16 bit data registers are indicated as (L) and (H), two bytes. Each byte of a two byte register can be written to independently.

Write Address	Writing Item	Length (bytes)	Min	Max
3(0X03)	ID	1	0	253(0xfd)
4(0X04)	Baud Rate	1	0	254(0xfe)
5(0X05)	Return Delay Time	1	0	254(0xfe)
6(0X06)	CW Angle Limit	2	0	1023(0x3ff)
8(0X08)	CCW Angle Limit	2	0	1023(0x3ff)
11(0X0B)	the Highest Limit Temperature	1	0	150(0x96)
12(0X0C)	the Lowest Limit Voltage	1	50(0x32)	250(0xfa)
13(0X0D)	the Highest Limit Voltage	1	50(0x32)	250(0xfa)
14(0X0E)	Max Torque	2	0	1023(0x3ff)
16(0X10)	Status Return Level	1	0	2
17(0X11)	Alarm LED	1	0	127(0x7f)
18(0X12)	Alarm Shutdown	1	0	127(0x7f)
19(0X13)	(Reserved)	1	0	1
24(0X18)	Torque Enable	1	0	1
25(0X19)	LED	1	0	1
26(0X1A)	CW Compliance Margin	1	0	254(0xfe)
27(0X1B)	CCW Compliance Margin	1	0	254(0xfe)
28(0X1C)	CW Compliance Slope	1	1	254(0xfe)
29(0X1D)	CCW Compliance Slope	1	1	254(0xfe)
30(0X1E)	Goal Position	2	0	1023(0x3ff)
32(0X20)	Moving Speed	2	0	1023(0x3ff)
34(0X22)	Torque Limit	2	0	1023(0x3ff)
44(0X2C)	Registered Instruction	1	0	1
47(0X2F)	Lock	1	1	1
48(0X30)	Punch	2	0	1023(0x3ff)

[Control Table Data Range and Length for Writing]

4. Instruction Set and Examples

The following Instructions are available.

Instruction	Function	Value	Number of Parameter
PING	No action. Used to obtain a Dynamixel Status Packet.	0x01	0
READ DATA	Read the values in the Control table.	0x02	2
WRITE DATA	Write the values to the Control Table.	0x03	2 ~
REG WRITE	Similar to WRITE DATA, but stay in standby mode until write upon the action instruction.	0x04	2 ~
ACTION	Start the action registered by REG WRITE.	0x05	0
RESET	Change the values of the Dynamixel in the control table back to the Factory Default Values	0x06	0

4-1. WRITE_DATA

Function   Write data into the control table of the Dynamixel
Length   N+3 (Writing Data is N)
Instruction   0X03
Parameter1   Start Address of the Area to write Data
Parameter2   1st Data to write
Parameter3   2nd Data to write
Parameter N+1   Nth Data to write

Example 1   Set ID of connected Dynamixel as 1

Write 1 into the Address 3 of the Control Table. The ID is transmitted using Broadcasting ID (0xFE).

Instruction Packet : 0XFF 0XFF 0XFE 0X04 0X03 0X03 0X01 0XF6

| ID | LENGTH | INSTRUCTION | PARAMETERS | CHECKSUM |
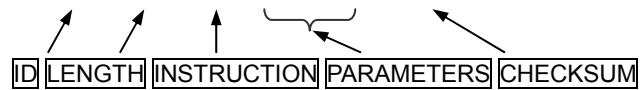
Because it was transmitted by Broadcast ID(0XFE), no return status packet.

4-2. READ_DATA

Function	Read data from the Control Table of Dynamixel
Length	0X04
Instruction	0X02
Parameter1	Starting Address of Data to Read
Parameter2	length of Data to Read

Example 2   Read the internal temperature of the Dynamixel with ID=1

Read 1 byte from the Address 0x2B values of the Control Table

Instruction Packet : 0XFF 0XFF 0X01 0X04 0X02 0X2B 0X01 0XCC`

| ID | LENGTH | INSTRUCTION | PARAMETERS | .. | CHECKSUM |
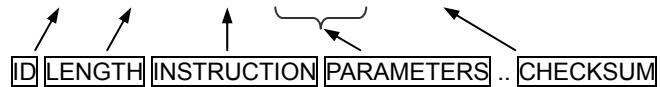
The returned Status Packet will be as follows

Status Packet : 0XFF 0XFF 0X01 0X03 0X00 0X20 0XDB

| ID | LENGTH | ERROR | PARAMETER1 | CHECKSUM |
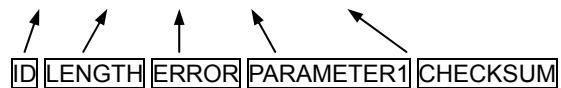
The value read is 0x20.The current Dynamixel's internal temperature is approximately 32℃ (0X20).

4-3. REG_WRITE and ACTION

4-3-1. REG_WRITE

Function	REG_WRITE instruction is similar to the WRITE_DATA instruction, but the execution timing is different. When the Instruction Packet is received the values are saved into the Buffer and the Write instruction is under a standby status. The Registered Instruction register (Address 0x2C) is set to 1. After an Action Instruction Packet is received the registered Write instruction is executed.
Length	N+3 (The number of Write Data bytes is N)
Instruction	0X04
Parameter1	Start Address for Write Data
Parameter2	1st Data to Write
Parameter3	2nd Data to Write
Parameter N+1	N+1 Nth Data to Write

4-3-2. ACTION

Function	Execute the WRITE instruction written by REG_WRITE
Length	0X02
Instruction	0X05
Parameter	NONE

The ACTION instruction is useful when multiple Dynamixels needs to move simultaneously. When controlling multiple units, slight time delays occur between the 1st unit to receive an instruction and the last one. The Dynamixel approach fixes this problem through the use of the ACTION instruction.

|	
Broadcasting	When sending ACTION instructions to move more than two Dynamixel units, the Broadcast ID (0XFE) should be utilised.

4-4. PING

Function	Used to request a specific Dynamixel status packet or to check the existence of a Dynamixel with a particular ID
Length	0X02
Instruction	0X01
Parameter	NONE

Example 3   To obtain the status packet of a Dynamixel with ID=1

Instruction Packet : 0XFF 0XFF 0X01 0X02 0X01 0XFB`

ID LENGTH INSTRUCTION CHECKSUM
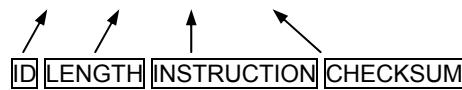
The returned Status Packet is as follow;

Status Packet : 0XFF 0XFF 0X01 0X02 0X00 0XFC

ID LENGTH ERROR CHECKSUM

4-5. RESET

Function   Restore the condition of the Control Table of the Dynamixel back to the Factory Default
values.

Length   0X02

Instruction   0X06

Parameter   NONE

Example 4   Reset Dynamixe with ID=0

Instruction Packet : 0XFF 0XFF 0X00 0X02 0X06 0XF7`

ID LENGTH INSTRUCTION CHECKSUM
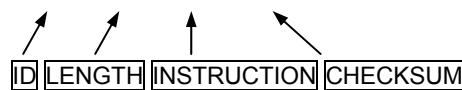
The returned Status Packet is as follows;

Status Packet : 0XFF 0XFF 0X00 0X02 0X00 0XFD

ID LENGTH ERROR CHECKSUM

Please note that after a RESET instruction, the ID of the Dynamixel is changed to 1.

5. Example

Used to explain through example with the assumption that the Dynamixel has been Reset (ID = 1, Baudrate = 57142BPS)

Example 6   Read the Model Number and Firmware Version of a Dynamixel with ID=1

Instruction Packet   Instruction = READ_DATA, Address = 0x00, Length = 0x03

Communication   ->[Dynamixel]:FF FF 01 04 02 00 03 F5 (LEN:008)
<-[Dynamixel]:FF FF 01 05 00 <u>74 00</u> <u>08</u> 7D (LEN:009)

Status Packet Result   Model Number = 116(0x74), Firmware Version = 0x08

Example 7   Change ID number of Dynamixel from 1 to 0.

Instruction Packet   Instruction = WRITE_DATA, Address = 0x03, DATA = 0x00

Communication   ->[Dynamixel]:FF FF 01 04 03 03 <u>00</u> F4 (LEN:008)
<-[Dynamixel]:FF FF 01 02 00 FC (LEN:006)

Status Packet Result   NO ERROR

Example 8   Change Baud Rate of Dynamixel to 1M bps.

Instruction Packet   Instruction = WRITE_DATA, Address = 0x04, DATA = 0x01

Communication   ->[Dynamixel]:FF FF 00 04 03 04 <u>01</u> F3 (LEN:008)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result   NO ERROR

Example 9   Reset Return Delay Time of Dynamixel with ID=0 to 4us.
A Return Delay Time Value of 1 corresponds to 2us.

Instruction Packet   Instruction = WRITE_DATA, Address = 0x05, DATA = 0x02

| **Communication** | ->[Dynamixel]:FF FF 00 04 03 05 <u>02</u> F1 (LEN:008) |
| | <-[Dynamixel]:FF FF 00 02 00 FD (LEN:006) |

Status Packet Result	NO ERROR
	The best approach is to set the Return Delay Time to the minimum value the Main
	Controller will allow.

Example 10 <u>Limit the the operative angles of a Dynamixel with ID=0 to 0~150°.</u>
If CCW Angle Limit is 0x3ff, it is 300°, therefore the values for 150°is 0x1ff.

Instruction Packet	Instruction = WRITE_DATA, Address = 0x08, DATA = 0xff, 0x01
Communication	->[Dynamixel]:FF FF 00 05 03 08 <u>FF 01</u> EF (LEN:009)
	<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

| **Status Packet Result** | NO ERROR |

Example 11 <u>Reset the upper limit temperature of the Dynamixel with ID=1 to 80°.</u>

Instruction Packet	Instruction = WRITE_DATA, Address = 0x0B, DATA = 0x50
Communication	->[Dynamixel]:FF FF 00 04 03 0B <u>50</u> 9D (LEN:008)
	<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

| **Status Packet Result** | NO ERROR |

Example 12 <u>Set the operative voltage of a Dynamixel with ID=0 to 10V ~ 17V.</u>
10V is expressed as 100(0x64) and 17V as 170(0xAA).

Instruction Packet	Instruction = WRITE_DATA, Address = 0x0C, DATA = 0x64, 0xAA
Communication	->[Dynamixel]:FF FF 00 05 03 0C <u>64 AA</u> DD (LEN:009)
	<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

| **Status Packet Result** | NO ERROR |

Example 13 Make the Dynamixel with ID=0 perform only 50% of the maximum torque.

Set the max torque values within the EEPROM area to 50% (0x1ff) of the maximum

value (0x3ff)

Instruction Packet Instruction = WRITE_DATA, Address = 0x0E, DATA = 0xff, 0x01

Communication ->[Dynamixel]:FF FF 00 05 03 0E FF 01 E9 (LEN:009)

<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR

After a power off and on, you can check the effect of the changes in max torque.

Example 14 Stop the Dynamixel with ID=0 from returning a Status Packet.

Instruction Packet Instruction = WRITE_DATA, Address = 0x10, DATA = 0x00

Communication ->[Dynamixel]:FF FF 00 04 03 10 00 E8 (LEN:008)

<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR

The Status Packet will not be returned for the next instruction.

Example 15 If temperature values are higher than those defined operative temperatures, set the

alarm to make the Dynamixel blink and then shut down the Dynamixel (Torque off).

Overheating Error is Bit 2, therefore set the alarm value to 0x04.

Instruction Packet Instruction = WRITE_DATA, Address = 0x11, DATA = 0x04, 0x04

Communication ->[Dynamixel]:FF FF 00 05 03 11 04 04 DE (LEN:009)

<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR

Example 16   Turn on the LED of the Dynamixel with ID=0 and enable the torque.

Instruction Packet   Instruction = WRITE_DATA, Address = 0x18, DATA = 0x01, 0x01

Communication   ->[Dynamixel]:FF FF 00 05 03 18 01 01 DD (LEN:009)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

**Status Packet Result**   NO ERROR

Physical confirmation of an enabled torque can be obtained by attempting to rotate the

motor with your hand.

Example 17   Set the Dynamixel with ID=0 to have a Compliance Margin = 1 and Compliance

Slope=0x40

Compliance   The following graph shows the Angle Error and Torque Output.
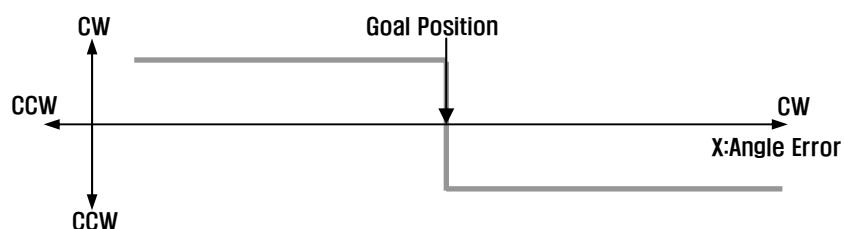
If the position is slightly deviated from the goal position, the motor will generate a high

torque to try to adjust its position to that of the goal position. The true control method is

different due to the inertia. The condition provided in the above example can be shown in

the graph below:-
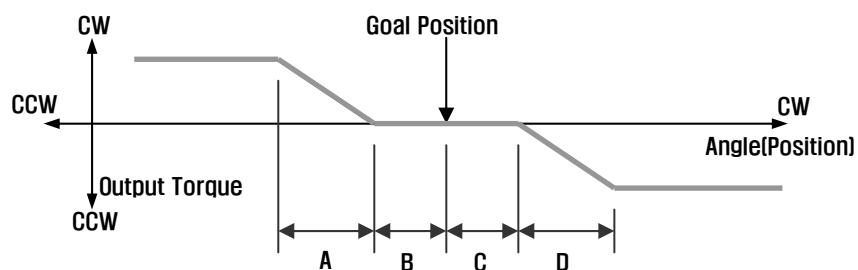
A : CCW Compliance Slope(Address0x1D) = 0x40(Approximately 18.8°)

B : CCW Compliance Margin(Address0x1B) = 0x01 (Approximately 0.29°)

C : CW Compliance Margin(Address0x01A) = 0x01(Approximately 0.29°)

D : CW Compliance Slope(Address0x1C) = 0x40 (Approximately 18.8°)

Instruction Packet	Instruction = WRITE_DATA, Address = 0x1A, DATA = 0x01, 0x01, 0x40, 0x40
Communication	->[Dynamixel]:FF FF 00 07 03 1A 01 01 40 40 59 (LEN:011) <-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)
Status Packet Result	NO ERROR The effect of a Compliance Slope changes at the boundary of 2n (n is positive number), that is, the effect of the values of Compliance between 0x11 and 0x20 are the same.
Example 18	Position Dynamixel with ID=0 at Position 180°after moving it at the speed of 35RPM. Set Address 0x1E(Goal Position) = 0x200, Address 0x20(Moving Speed) = 0x200
Instruction Packet	Instruction = WRITE_DATA, Address = 0x1E, DATA = 0x00, 0x02, 0x00, 0x02
Communication	->[Dynamixel]:FF FF 00 07 03 1E 00 02 00 02 D3 (LEN:011) <-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)
Status Packet Result	NO ERROR
Example 19	Set the position of a Dynamixel (ID=0) to an angular Position of 0°and another Dynamixel (ID=1) to an angular Position of 300°. Make sure both Dynamixels start at the same time. If you use WRITE_DATA instruction, two Dynamixel can not start at the same time, therefore use REG_WRITE and ACTION.
Instruction Packet	ID=0, Instruction = REG_WRITE, Address = 0x1E, DATA = 0x00, 0x00 ID=1, Instruction = REG_WRITE, Address = 0x1E, DATA = 0xff, 0x03 ID=0xfe(Broadcasting ID), Instruction = ACTION,
Communication	->[Dynamixel]:FF FF 00 05 04 1E 00 00 D8 (LEN:009) <-[Dynamixel]:FF FF 00 02 00 FD (LEN:006) ->[Dynamixel]:FF FF 01 05 04 1E FF 03 D5 (LEN:009) <-[Dynamixel]:FF FF 01 02 00 FC (LEN:006) ->[Dynamixel]:FF FF FE 02 05 FA (LEN:006) <-[Dynamixel]: //No return packet against broadcasting ID
Status Packet Result	NO ERROR

Example 20 Prevent the Dynamixel with ID=0 from changing values other than within the range between Address 0x18 and Address 0x23.

Set Address 0x2F(Lock) to 1.

Instruction Packet Instruction = WRITE_DATA, Address = 0x2F, DATA = 0x01

Communication ->[Dynamixel]:FF FF 00 04 03 2F 01 C8 (LEN:008)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR

If Locked, it can only be unlocked by removing power.

If trying to access other data areas whilst locked, an error will be returned.

->[Dynamixel]:FF FF 00 05 03 30 40 00 87 (LEN:009)
<-[Dynamixel]:FF FF 00 02 08 F5 (LEN:006)

Range Error

Example 21 Set the minimum punch (output) in the Dynamixel with ID=0 to 0x40.

Instruction Packet Instruction = WRITE_DATA, Address = 0x30, DATA = 0x40, 0x00

Communication ->[Dynamixel]:FF FF 00 05 03 30 40 00 87 (LEN:009)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR

<u>Appendix</u>

RS-485 RS-485 is a protocol used for serial communication which operates by forming a bus with multiple clients connected to a single line. Thus, transmission and reception cannot occur at the same time, and while one client is transmitting, all the other clients need to be in input mode. The Main Controller that controllers the Dynamixel actuators sets the RS485 communication direction to be input mode, and only when it is transmitting an Instruction Packet, it changes the direction to be output mode.
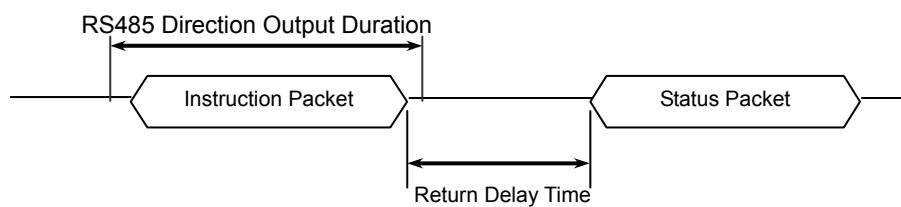
Return Delay Time The time it takes for the Dynamixel actuator to return the Status Packet after receiving an Instruction Packet. The Default Value is 160 uSec and can be changed via the Control Table at Address 5. The Main Controller needs to change the RS485 communication direction during the Return Delay Tim after sending an instruction packet.

485 Direction For RS-485, the timing to change the direction to receiving mode right after the ending of the transmission is important. The bit definitions within the register that indicates UART_STATUS are as the following.

TXD_BUFFER_READY_BIT: Indicates that the transmission DATA can be loaded into the Buffer. Note that this only means that the SERIAL TX BUFFER is empty, and does not necessarily mean that the all the data transmitted before has left the CPU.

TXD_SHIFT_REGISTER_EMPTY_BIT: Set when all the Transmission Data has completed its transmission and left the CPU.
The TXD_BUFFER_READY_BIT is used when one byte is to be transmitted via the serial communication channel, and an example is shown below.

```
TxDByte(byte bData)
{
while(!TXD_BUFFER_READY_BIT);   //wait until data can be loaded.
SerialTxDBuffer = bData;         //data load to TxD buffer
}
```

When changing the direction of RS-485, the TXD_SHIFT_REGISTER_EMPTY_BIT must be checked.

**The following is an example program that sends an Instruction Packet.**

LINE 1	PORT_485_DIRECTION = TX_DIRECTION;
LINE 2	TxDByte(0xff);
LINE 3	TxDByte(0xff);
LINE 4	TxDByte(bID);
LINE 5	TxDByte(bLength);
LINE 6	TxDByte(bInstruction);
LINE 7	TxDByte(Parameter0); TxDByte(Parameter1); …
LINE 8	DisableInterrupt(); // interrupt should be disable
LINE 9	TxDByte(Checksum); //last TxD
LINE 10	while(!TXD_SHIFT_REGISTER_EMPTY_BIT); //Wait till last data bit has been sent
LINE 11	PORT_485_DIRECTION = RX_DIRECTION; //485 direction change to RXD
LINE 12	EnableInterrupt(); // enable interrupt again

Please note the important lines between LINE 8 and LINE 12. Line 8 is necessary since an interrupt here may cause a delay longer than the return delay time and corruption to the front of the status packet may occur.

Byte to Byte Time   The delay time between bytes when sending an instruction packet. If the delay time is over 100ms, then the Dynamixel actuator recognizes this as a communication problem and waits for the next header (0xff 0xff) of a packet again.

Byte To Byte Time
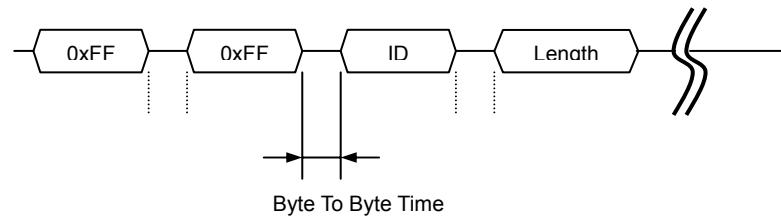
The following is the source code of a program (Example.c) that accesses the Dynamixel actuator using the Atmega 128.

C Language Example : Dinamixel access with Atmega128

```
/*
 * The Example of Dynamixel Evaluation with Atmega128
 * Date : 2004.7.20
 */

#define ENABLE_BIT_DEFINITIONS
//#include <io.h>
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

typedef unsigned char byte;
typedef unsigned int word;
#define ON 1
#define OFF 0
#define _ON 0
#define _OFF 1

//--- Control Table Address ---
//EEPROM AREA
#define P_MODEL_NUMBER_L      0
#define P_MODOEL_NUMBER_H     1
#define P_VERSION             2
#define P_ID                  3
#define P_BAUD_RATE           4
#define P_RETURN_DELAY_TIME   5
#define P_CW_ANGLE_LIMIT_L    6
#define P_CW_ANGLE_LIMIT_H    7
#define P_CCW_ANGLE_LIMIT_L   8
#define P_CCW_ANGLE_LIMIT_H   9
#define P_SYSTEM_DATA2        10
#define P_LIMIT_TEMPERATURE   11
#define P_DOWN_LIMIT_VOLTAGE  12
#define P_UP_LIMIT_VOLTAGE    13
#define P_MAX_TORQUE_L        14
#define P_MAX_TORQUE_H        15
#define P_RETURN_LEVEL        16
#define P_ALARM_LED           17
#define P_ALARM_SHUTDOWN      18
#define P_OPERATING_MODE      19
#define P_DOWN_CALIBRATION_L  20
#define P_DOWN_CALIBRATION_H  21
#define P_UP_CALIBRATION_L    22
#define P_UP_CALIBRATION_H    23

#define P_TORQUE_ENABLE          (24)
#define P_LED                    (25)
#define P_CW_COMPLIANCE_MARGIN   (26)
#define P_CCW_COMPLIANCE_MARGIN  (27)
#define P_CW_COMPLIANCE_SLOPE    (28)
#define P_CCW_COMPLIANCE_SLOPE   (29)
#define P_GOAL_POSITION_L        (30)
#define P_GOAL_POSITION_H        (31)
#define P_GOAL_SPEED_L           (32)
#define P_GOAL_SPEED_H           (33)
#define P_TORQUE_LIMIT_L         (34)
#define P_TORQUE_LIMIT_H         (35)
#define P_PRESENT_POSITION_L     (36)
#define P_PRESENT_POSITION_H     (37)
#define P_PRESENT_SPEED_L        (38)
#define P_PRESENT_SPEED_H        (39)
#define P_PRESENT_LOAD_L         (40)
#define P_PRESENT_LOAD_H         (41)
#define P_PRESENT_VOLTAGE        (42)
#define P_PRESENT_TEMPERATURE    (43)
#define P_REGISTERED_INSTRUCTION (44)
#define P_PAUSE_TIME             (45)
#define P_MOVING (46)
#define P_LOCK                   (47)
#define P_PUNCH_L                (48)
#define P_PUNCH_H                (49)

//--- Instruction ---
#define INST_PING          0x01
```

```
#define INST_READ          0x02
#define INST_WRITE         0x03
#define INST_REG_WRITE     0x04
#define INST_ACTION        0x05
#define INST_RESET         0x06
#define INST_DIGITAL_RESET 0x07
#define INST_SYSTEM_READ   0x0C
#define INST_SYSTEM_WRITE  0x0D
#define INST_SYNC_WRITE    0x83
#define INST_SYNC_REG_WRITE 0x84

#define CLEAR_BUFFER gbRxBufferReadPointer = gbRxBufferWritePointer
#define DEFAULT_RETURN_PACKET_SIZE 6
#define BROADCASTING_ID 0xfe

#define TxD8 TxD81
#define RxD8 RxD81

//Hardware Dependent Item
#define DEFAULT_BAUD_RATE 34   //57600bps at 16MHz
#define RS485_TXD PORTE |= _BV(PE2); //_485_DIRECTION = 1
#define RS485_RXD PORTE &= ~_BV(PE2);//PORT_485_DIRECTION = 0

//#define PORT_485_DIRECTION PORTE.P2 //Bit2 of PortE is linked to
//                         MAX485 direction pin.
//#define TXD0_FINISH  UCSR0A,6  //This bit is for checking TxD Buffer
//                               in CPU is empty or not.
//#define TXD1_FINISH  UCSR1A,6

#define SET_TxD0_FINISH   sbi(UCSR0A,6)
#define RESET_TXD0_FINISH cbi(UCSR0A,6)
#define CHECK_TXD0_FINISH bit_is_set(UCSR0A,6)
#define SET_TxD1_FINISH   sbi(UCSR1A,6)
#define RESET_TXD1_FINISH cbi(UCSR1A,6)
#define CHECK_TXD1_FINISH bit_is_set(UCSR1A,6)

#define RX_INTERRUPT 0x01
#define TX_INTERRUPT 0x02
#define OVERFLOW_INTERRUPT 0x01
#define SERIAL_PORT0 0
#define SERIAL_PORT1 1

#define LED_M0_ON cbi(PORTE,3) //PORTE_Bit3
#define LED_M1_ON cbi(PORTE,4) //PORTE_Bit4
#define LED_M2_ON cbi(PORTE,6) //PORTE_Bit6
#define LED_E0_ON cbi(PORTE,7) //PORTE_Bit7
#define LED_E1_ON cbi(PORTB,0) //PORTB_Bit0

#define LED_M0_OFF sbi(PORTE,3) //PORTE_Bit3
#define LED_M1_OFF sbi(PORTE,4) //PORTE_Bit4
#define LED_M2_OFF sbi(PORTE,6) //PORTE_Bit6
#define LED_E0_OFF sbi(PORTE,7) //PORTE_Bit7
#define LED_E1_OFF sbi(PORTB,0) //PORTB_Bit0

#define BIT_LED_M0 0x08 //Port E
#define BIT_LED_M1 0x10 //Port E
#define BIT_LED_M2 0x40 //Port E
#define BIT_LED_E0 0x80 //Port E
#define BIT_LED_E1 0x01 //Port B

#define BIT_RS485_DIRECTION 0x04 //Port E
void TxD81(byte bTxdData);
void TxD80(byte bTxdData);
void TxDString(byte *bData);
void TxD8Hex(byte bSentData);
void TxD32Dec(long lLong);
byte RxD81(void);
void MiliSec(word wDelayTime);
void PortInitialize(void);
void SerialInitialize(byte bPort, byte bBaudrate, byte bInterrupt);
byte TxPacket(byte bID, byte bInstruction, byte bParameterLength);
byte RxPacket(byte bRxLength);
void PrintBuffer(byte *bpPrintBuffer, byte bLength);

// --- Gloval Variable Number ---
```

```
volatile byte gbpRxInterruptBuffer[256];
byte gbpParameter[128];
byte gbpRxBufferReadPointer;
byte gbpRxBuffer[128];
byte gbpTxBuffer[128];
volatile byte gbpRxBufferWritePointer;

int main(void)
{
  byte bCount,bID, bTxPacketLength,bRxPacketLength;

  PortInitialize(); //Port In/Out Direction Definition
  RS485_RXD; //Set RS485 Direction to Input State.

//RS485 Initializing(RxInterrupt)
  SerialInitialize(SERIAL_PORT0, DEFAULT_BAUD_RATE,RX_INTERRUPT);

//RS232 Initializing(None Interrupt)
  SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,0);

//RS485 RxBuffer Clearing.
  gbRxBufferReadPointer = gbRxBufferWritePointer = 0;

  sei();  //Enable Interrupt -- Compiler Function
  TxDString("\r\n  [The  Example  of  Dynamixel  Evaluation  with
  ATmega128, GCC-AVR]");

//Dynamixel Communication Function Execution Step.
// Step 1. Parameter Setting (gbpParameter[]). In case of no parameter
  instruction(Ex. INST_PING), this step is not needed.
//  Step  2.   TxPacket(ID, INSTRUCTION,LengthOfParameter);   --Total
  TxPacket Length is returned
// Step 3. RxPacket(ExpectedReturnPacketLength); -- Real RxPacket
  Length is returned
// Step 4 PrintBuffer(BufferStartPointer,LengthForPrinting);


  bID = 1;
  TxDString("\r\n\n Example 1. Scanning Dynamixels(0~9). -- Any Key to
  Continue."); RxD8();
  for(bCount = 0; bCount < 0x0A; bCount++)
  {
    bTxPacketLength = TxPacket(bCount, INST_PING,0);
    bRxPacketLength = RxPacket(255);
    TxDString("\r\n TxD:"); PrintBuffer(gbpTxBuffer, bTxPacketLength);
    TxDString(", RxD:");    PrintBuffer(gbpRxBuffer, bRxPacketLength);
    if(bRxPacketLength == DEFAULT_RETURN_PACKET_SIZE)
    {
      TxDString(" Found!! ID:");TxD8Hex(bCount);
      bID = bCount;
    }
  }

TxDString("\r\n\n Example 2. Read Firmware Version. -- Any Key to
  Continue."); RxD8();
  gbpParameter[0] = P_VERSION; //Address of Firmware Version
  gbpParameter[1] = 1; //Read Length
  bTxPacketLength = TxPacket(bID, INST_READ,2);
  RxPacketLength=RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1]);
  TxDString("\r\n TxD:"); PrintBuffer(gbpTxBuffer, bTxPacketLength);
  TxDString("\r\n RxD:"); PrintBuffer(gbpRxBuffer, bRxPacketLength);
  if(bRxPacketLength == DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1])
  {
    TxDString("\r\n Return Error      : ");TxD8Hex(gbpRxBuffer[4]);
    TxDString("\r\n Firmware Version  : ");TxD8Hex(gbpRxBuffer[5]);
  }


  TxDString("\r\n\n Example 3. LED ON -- Any Key to Continue.");
  RxD8();
  gbpParameter[0] = P_LED; //Address of LED
  gbpParameter[1] = 1; //Writing Data
  bTxPacketLength = TxPacket(bID, INST_WRITE,2);
  bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
  TxDString("\r\n TxD:"); PrintBuffer(gbpTxBuffer, bTxPacketLength);
  TxDString("\r\n RxD:"); PrintBuffer(gbpRxBuffer, bRxPacketLength);
```

```
  TxDString("\r\n\n Example 4. LED OFF -- Any Key to Continue.");
  RxD8();
  gbpParameter[0] = P_LED; //Address of LED
  gbpParameter[1] = 0; //Writing Data
  bTxPacketLength = TxPacket(bID, INST_WRITE,2);
  bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
  TxDString("\r\n TxD:"); PrintBuffer(gbpTxBuffer, bTxPacketLength);
  TxDString("\r\n RxD:"); PrintBuffer(gbpRxBuffer, bRxPacketLength);


  TxDString("\r\n\n Example 5.  Read Control Table.  -- Any Key to
  Continue."); RxD8();
  gbpParameter[0] = 0; //Reading Address
  gbpParameter[1] = 49; //Read Length
  bTxPacketLength = TxPacket(bID, INST_READ,2);
  bRxPacketLength                                          =
  RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1]);

  TxDString("\r\n TxD:"); PrintBuffer(gbpTxBuffer, bTxPacketLength);
  TxDString("\r\n RxD:"); PrintBuffer(gbpRxBuffer, bRxPacketLength);
  if(bRxPacketLength == DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1])
  {
    TxDString("\r\n");
    for(bCount = 0; bCount < 49; bCount++)
    {
      TxD8('[');TxD8Hex(bCount);TxDString("]:");
  TxD8Hex(gbpRxBuffer[bCount+5]);TxD8(' ');
    }
  }

  TxDString("\r\n\n Example 6. Go 0x200 with Speed 0x100 -- Any Key
  to Continue."); RxD8();
  gbpParameter[0] = P_GOAL_POSITION_L; //Address of Firmware Version
  gbpParameter[1] = 0x00; //Writing Data P_GOAL_POSITION_L
  gbpParameter[2] = 0x02; //Writing Data P_GOAL_POSITION_H
  gbpParameter[3] = 0x00; //Writing Data P_GOAL_SPEED_L
  gbpParameter[4] = 0x01; //Writing Data P_GOAL_SPEED_H
  bTxPacketLength = TxPacket(bID, INST_WRITE,5);
  bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
  TxDString("\r\n TxD:"); PrintBuffer(gbpTxBuffer, bTxPacketLength);
  TxDString("\r\n RxD:"); PrintBuffer(gbpRxBuffer, bRxPacketLength);

  TxDString("\r\n\n Example 7. Go 0x00 with Speed 0x40 -- Any Key to
  Continue."); RxD8();
  gbpParameter[0] = P_GOAL_POSITION_L; //Address of Firmware Version
  gbpParameter[1] = 0x00; //Writing Data P_GOAL_POSITION_L
  gbpParameter[2] = 0x00; //Writing Data P_GOAL_POSITION_H
  gbpParameter[3] = 0x40; //Writing Data P_GOAL_SPEED_L
  gbpParameter[4] = 0x00; //Writing Data P_GOAL_SPEED_H
  bTxPacketLength = TxPacket(bID, INST_WRITE,5);
  bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
  TxDString("\r\n TxD:"); PrintBuffer(gbpTxBuffer, bTxPacketLength);
  TxDString("\r\n RxD:"); PrintBuffer(gbpRxBuffer, bRxPacketLength);

  TxDString("\r\n\n Example 8. Go 0x3ff with Speed 0x3ff -- Any Key to
  Continue."); RxD8();
  gbpParameter[0] = P_GOAL_POSITION_L; //Address of Firmware Version
  gbpParameter[1] = 0xff; //Writing Data P_GOAL_POSITION_L
  gbpParameter[2] = 0x03; //Writing Data P_GOAL_POSITION_H
  gbpParameter[3] = 0xff; //Writing Data P_GOAL_SPEED_L
  gbpParameter[4] = 0x03; //Writing Data P_GOAL_SPEED_H
  bTxPacketLength = TxPacket(bID, INST_WRITE,5);
  bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
  TxDString("\r\n TxD:"); PrintBuffer(gbpTxBuffer, bTxPacketLength);
  TxDString("\r\n RxD:"); PrintBuffer(gbpRxBuffer, bRxPacketLength);

  TxDString("\r\n\n Example 9. Torque Off -- Any Key to Continue.");
  RxD8();
  gbpParameter[0] = P_TORQUE_ENABLE; //Address of LED
  gbpParameter[1] = 0; //Writing Data
  bTxPacketLength = TxPacket(bID, INST_WRITE,2);
  bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
  TxDString("\r\n TxD:"); PrintBuffer(gbpTxBuffer, bTxPacketLength);
  TxDString("\r\n RxD:"); PrintBuffer(gbpRxBuffer, bRxPacketLength);
  TxDString("\r\n\n End. Push reset button for repeat");
  while(1);
}
```

```
/*
About Register and value of bits, vide Mega128 Data Sheet.
*/
void PortInitialize(void)
{
  DDRA = DDRB = DDRC = DDRD = DDRE = DDRF = 0;  //Set all port to
   input direction first.
  PORTB = PORTC = PORTD = PORTE = PORTF = PORTG = 0x00; //PortData
   initialize to 0
  cbi(SFIOR,2); //All Port Pull Up ready

  //Set 5 LED port and RS485Direction port to output
  DDRB |= (BIT_LED_E1);
  DDRE                                                      |=
   (BIT_RS485_DIRECTION|BIT_LED_M0|BIT_LED_M1|BIT_LED_M2|BIT_LED_E0);

  //TurnOff LED
  LED_M0_OFF; LED_M1_OFF;LED_M2_OFF;LED_E0_OFF;LED_E1_OFF;
}
/*
TxPacket() send data to RS485.
TxPacket() needs 3 parameter; ID of Dynamixel, Instruction byte,
  Length of parameters.
TxPacket() return length of Return packet from Dynamixel.

*/
byte TxPacket(byte bID, byte bInstruction, byte bParameterLength)
{
    byte bCount,bCheckSum,bPacketLength;

    gbpTxBuffer[0] = 0xff;
    gbpTxBuffer[1] = 0xff;
    gbpTxBuffer[2] = bID;
    gbpTxBuffer[3]               =               bParameterLength+2;
  //Length(Paramter, Instruction, Checksum)
    gbpTxBuffer[4] = bInstruction;
    for(bCount = 0; bCount < bParameterLength; bCount++)
    {
        gbpTxBuffer[bCount+5] = gbpParameter[bCount];
    }
    bCheckSum = 0;
    bPacketLength = bParameterLength+4+2;
    for(bCount = 2; bCount < bPacketLength-1; bCount++) //except
0xff,checksum
    {
        bCheckSum += gbpTxBuffer[bCount];
    }
    gbpTxBuffer[bCount] = ~bCheckSum; //Writing Checksum with Bit
  Inversion

    RS485_TXD;
    for(bCount = 0; bCount < bPacketLength; bCount++)
    {
        sbi(UCSR0A,6);//SET_TXD0_FINISH;
        TxD80(gbpTxBuffer[bCount]);
    }
    while(!CHECK_TXD0_FINISH); //Wait until TXD Shift register empty
    RS485_RXD;
    return(bPacketLength);
}

#define RX_WAIT_TIMEOUT    0xf000
#define RX_TIMEOUT_COUNT2  3000L
#define RX_TIMEOUT_COUNT1  (RX_TIMEOUT_COUNT2*10L)

/*
RxPacket() read data from buffer.
RxPacket() need a Parameter; Total length of Return Packet.
RxPacket() return Length of Return Packet.
*/

byte RxPacket(byte bRxPacketLength)
{
  unsigned long ulCounter;
  byte bCount, bLength, bChecksum;
  byte bTimeout;
```

```
  bTimeout = 0;
  for(bCount = 0; bCount < bRxPacketLength; bCount++)
  {
    ulCounter = 0;
    while(gbRxBufferReadPointer == gbRxBufferWritePointer)
    {
      if(ulCounter++ > RX_TIMEOUT_COUNT1)
      {
        bTimeout = 1;
        break;
      }
    }
    if(bTimeout) break;
    gbpRxBuffer[bCount]                                      =
  gbpRxInterruptBuffer[gbRxBufferReadPointer++];
  }
  bLength = bCount;
  bChecksum = 0;

  if(gbpTxBuffer[2] != BROADCASTING_ID)
  {
    if(bTimeout && bRxPacketLength != 255)
    {
      TxDString("\r\n [Error:RxD Timeout]");
      CLEAR_BUFFER;
    }

    if(bLength > 3) //checking is available.
    {
      if(gbpRxBuffer[0] != 0xff || gbpRxBuffer[1] != 0xff )
      {
        TxDString("\r\n [Error:Wrong Header]");
        CLEAR_BUFFER;
        return 0;
      }
      if(gbpRxBuffer[2] != gbpTxBuffer[2] )
      {
        TxDString("\r\n [Error:TxID != RxID]");
        CLEAR_BUFFER;
        return 0;
      }
      if(gbpRxBuffer[3] != bLength-4)
      {
        TxDString("\r\n [Error:Wrong Length]");
        CLEAR_BUFFER;
        return 0;
      }
      for(bCount = 2; bCount < bLength; bCount++) bChecksum +=
  gbpRxBuffer[bCount];
        if(bChecksum != 0xff)
        {
          TxDString("\r\n [Error:Wrong CheckSum]");
          CLEAR_BUFFER;
          return 0;
        }
    }
  }
  return bLength;
}

/*
PrintBuffer() print data in Hex code.
PrintBuffer() needs two parameter; name of Pointer(gbpTxBuffer,
  gbpRxBuffer)
*/
void PrintBuffer(byte *bpPrintBuffer, byte bLength)
{
    byte bCount;
    for(bCount = 0; bCount < bLength; bCount++)
    {
        TxD8Hex(bpPrintBuffer[bCount]);
        TxD8(' ');
    }
    TxDString("(LEN:");TxD8Hex(bLength);TxD8(')');
}

/*
```

```
Print value of Baud Rate.
*/
void PrintBaudrate(void)
{
  TxDString("\r\n
  RS232:");TxD32Dec((16000000L/8L)/((long)UBRR1L+1L) );  TxDString("
  BPS,");
  TxDString("  RS485:");TxD32Dec((16000000L/8L)/((long)UBRR0L+1L) );
  TxDString(" BPS");
}

/*Hardware Dependent Item*/
#define TXD1_READY              bit_is_set(UCSR1A,5) //(UCSR1A_Bit5)
#define TXD1_RESET
#define TXD1_DATA               (UDR1)
#define RXD1_READY              bit_is_set(UCSR1A,7)
#define RXD1_RESET
#define RXD1_DATA               (UDR1)

#define TXD0_READY              bit_is_set(UCSR0A,5)
#define TXD0_RESET
#define TXD0_DATA               (UDR0)
#define RXD0_READY              bit_is_set(UCSR0A,7)
#define RXD0_RESET
#define RXD0_DATA               (UDR0)

/*
SerialInitialize() set Serial Port to initial state.
Vide Mega128 Data sheet about Setting bit of register.
SerialInitialize() needs port, Baud rate, Interrupt value.

*/
void SerialInitialize(byte bPort, byte bBaudrate, byte bInterrupt)
{
  if(bPort == SERIAL_PORT0)
  {
    UBRR0H = 0; UBRR0L = bBaudrate;
    UCSR0A = 0x02;  UCSR0B = 0x18;
    if(bInterrupt&RX_INTERRUPT) sbi(UCSR0B,7); // RxD interrupt enable
    UCSR0C = 0x06; UDR0 = 0xFF;
    sbi(UCSR0A,6);//SET_TXD0_FINISH; // Note. set 1, then 0 is read
  }
  else if(bPort == SERIAL_PORT1)
  {
    UBRR1H = 0; UBRR1L = bBaudrate;
    UCSR1A = 0x02;  UCSR1B = 0x18;
    if(bInterrupt&RX_INTERRUPT) sbi(UCSR1B,7); // RxD interrupt enable
    UCSR1C = 0x06; UDR1 = 0xFF;
    sbi(UCSR1A,6);//SET_TXD1_FINISH; // Note. set 1, then 0 is read
  }
}

/*
TxD8Hex() print data seperatly.
ex> 0x1a -> '1' 'a'.
*/
void TxD8Hex(byte bSentData)
{
  byte bTmp;

  bTmp =((byte)(bSentData>>4)&0x0f) + (byte)'0';
  if(bTmp > '9') bTmp += 7;
  TxD8(bTmp);
  bTmp =(byte)(bSentData & 0x0f) + (byte)'0';
  if(bTmp > '9') bTmp += 7;
  TxD8(bTmp);
}

/*
TxD80() send data to USART 0.
*/
void TxD80(byte bTxdData)
{
  while(!TXD0_READY);
  TXD0_DATA = bTxdData;
}

/*
TXD81() send data to USART 1.
*/
void TxD81(byte bTxdData)
{
  while(!TXD1_READY);
  TXD1_DATA = bTxdData;
}

/*
TXD32Dex() change data to decimal number system
*/
void TxD32Dec(long lLong)
{
  byte bCount, bPrinted;
  long lTmp, lDigit;
  bPrinted = 0;
  if(lLong < 0)
  {
    lLong = -lLong;
    TxD8('-');
  }
  lDigit = 1000000000L;
  for(bCount = 0; bCount < 9; bCount++)
  {
    lTmp = (byte)(lLong/lDigit);
    if(lTmp)
    {
      TxD8(((byte)lTmp)+'0');
      bPrinted = 1;
    }
    else if(bPrinted) TxD8(((byte)lTmp)+'0');
    lLong -= ((long)lTmp)*lDigit;
    lDigit = lDigit/10;
  }
  lTmp = (byte)(lLong/lDigit);
  /*if(lTmp)*/ TxD8(((byte)lTmp)+'0');
}

/*
TxDString() prints data in ACSII code.

*/
void TxDString(byte *bData)
{
  while(*bData)
  {
    TxD8(*bData++);
  }
}
/*
RxD81() read data from Port 1.
RxD81() return Read data.

*/
byte RxD81(void)
{
  while(!RXD1_READY);
  RXD1_RESET;
  return(RXD1_DATA);
}

/*
SIGNAL() UART0 Rx Interrupt - write data to buffer

*/
SIGNAL (SIG_UART0_RECV)
{
  gbpRxInterruptBuffer[(gbRxBufferWritePointer++)] = RXD0_DATA;
}
```

C Language Example : Dinamixel access with Am188ER CPU

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <mem.h>

#define MCS80
#include "..\..\base.h"
#include "..\lib188es.c"

#define RS485_DIRECTION_BIT 0x2000
#define RS485_TXD (SET_PORT1(RS485_DIRECTION_BIT))
#define RS485_RXD (RESET_PORT1(RS485_DIRECTION_BIT))

#define BROADCASTING_ID 0xfe
#define MEMORY_SPARE 10

#define ADDRESS_TORQUE_ENABLE    20
#define ADDRESS_OPERATING_MODE   19
#define ADDRESS_ID                3
#define ADDRESS_GOAL_POSITION    26

#define INST_PING          0x01
#define INST_READ          0x02
#define INST_WRITE         0x03
#define INST_SET_SCHEDULE  0x04
#define INST_DO_SCHEDULE   0x05
#define INST_RESET         0x06

#define DIGITAL_MODE 0
```
Definition area

```
//Gloval variable number
byte gbpInterruptRxBuffer[256+MEMORY_SPARE];   //485 RxD Data Buffer
byte gbRxBufferReadPointer,gbRxBufferWritePointer; //Pointers for access the gbpInterruptRxBuffer

void static interrupt far Serial0Interrupt(void);
void PrintBuffer(byte *bpPrintBuffer, byte bLength);
byte TxPacket(byte *bpTxBuffer, byte bID, byte bInstruction, byte *bpParameter, byte bParameterLength);
byte RxPacket(byte *bpRxBuffer);
```
Major
Function

```
void main(void)
{
    byte bID,bPacketLength;
    byte bpTxBuffer[20+MEMORY_SPARE];
    byte bpRxBuffer[256+MEMORY_SPARE];
    byte bpParameter[256+MEMORY_SPARE];

    CLI; //Disable Interrupt
    //PortInitialize();
    InitPort(OUT, PDATA1, 0xe000); //Set Out(Port30,31:LED,Port29:485Direction)
    InitPort(IN,   PDATA1, 0x0004); //Set In(Port2:Push SW)
    InitPort(NORMAL_USE,   PDATA1, 0x00c0); //
    RS485_RXD;     //Set 485 Direction Select Port to 0

    //UartInitialize();
    outpw(SP0BAUD,5); //.2MBPS = 16MHz/16/5
    outpw(SP1BAUD,17); //57600
    outpw(SP0STS,0);
    outpw(SP1STS,0);

    //InterruptInitialize();
```
CPU dependent Initialize

```
    SetInterrupt(INUM_SERIAL0,Serial0Interrupt,INT_ENABLE|INT_RX, 7/*Priority*/);

    //Memory Initialize
    gbRxBufferReadPointer = gbRxBufferWritePointer = 0;
    STI; //Interrupt Enable

    /*
     *
     *   Example For Driving Dynamixel DX-116
     *
     */
    TxDString("\r\n\n Dynamixel Driving Sample Program");


    //Set ID to 3
    bpParameter[0] = ADDRESS_ID;
    bpParameter[1] = 3;
    bPacketLength = TxPacket(bpTxBuffer, BROADCASTING_ID, INST_WRITE, bpParameter, 2/*Length of Parameter*/);
    bID = 3;
    TxDString("\r\n ->[Dynamixel]: "); PrintBuffer(bpTxBuffer,bPacketLength);
    bPacketLength = RxPacket(bpRxBuffer);
    TxDString("\r\n <-[Dynamixel]: "); PrintBuffer(bpRxBuffer,bPacketLength);

    //Set Motor Torque Enable
    bpParameter[0] = ADDRESS_TORQUE_ENABLE;
    bpParameter[1] = 1;
    bPacketLength = TxPacket(bpTxBuffer, bID, INST_WRITE, bpParameter, 2/*Length of Parameter*/);
    TxDString("\r\n ->[Dynamixel]: "); PrintBuffer(bpTxBuffer,bPacketLength);
    bPacketLength = RxPacket(bpRxBuffer);
    TxDString("\r\n <-[Dynamixel]: "); PrintBuffer(bpRxBuffer,bPacketLength);

    //Move to Position 0x0100 <-> 0x300
    while(1)
    {
        bpParameter[0] = ADDRESS_GOAL_POSITION;
        bpParameter[1] = 0x00; bpParameter[2] = 0x01;
        bPacketLength = TxPacket(bpTxBuffer, bID, INST_WRITE, bpParameter, 3/*Length of Parameter*/);
        TxDString("\r\n ->[Dynamixel]: "); PrintBuffer(bpTxBuffer,bPacketLength);
        bPacketLength = RxPacket(bpRxBuffer);
        TxDString("\r\n <-[Dynamixel]: "); PrintBuffer(bpRxBuffer,bPacketLength);
        MiliSec(1000);

        bpParameter[0] = ADDRESS_GOAL_POSITION;
        bpParameter[1] = 0x00; bpParameter[2] = 0x03;
        bPacketLength = TxPacket(bpTxBuffer, bID, INST_WRITE, bpParameter, 3/*Length of Parameter*/);
        TxDString("\r\n ->[Dynamixel]: "); PrintBuffer(bpTxBuffer,bPacketLength);
        bPacketLength = RxPacket(bpRxBuffer);
        TxDString("\r\n <-[Dynamixel]: "); PrintBuffer(bpRxBuffer,bPacketLength);
        MiliSec(1000);
    }

    //while(1);
}

void static interrupt far Serial0Interrupt(void)      //Serial RxD Interrupt routine
{
    STI; //Enable Interrupt
    gbpInterruptRxBuffer[gbRxBufferWritePointer++] = RXD_DATA0;   //Reading Arrival Data
    outpw(EOI, 0x14); //End of Interrupt
}


byte RxPacket(byte *bpRxBuffer)
{
```

```
        #define RX_TIMEOUT_COUNT2     10000L   //10mSec
        #define RX_TIMEOUT_COUNT1   (RX_TIMEOUT_COUNT2*10L)   //1Sec

        unsigned long ulCounter;
        byte bCount;

        ulCounter = 0;
        while(gbRxBufferReadPointer == gbRxBufferWritePointer)
        {
            if(ulCounter++ > RX_TIMEOUT_COUNT1)
            {
                return 0;
            }
        }
        bCount = 0;
        for(bCount = 0; bCount < 254; bCount++) //Maximum Data Length Limit : 255
        {
            ulCounter = 0;
            while(gbRxBufferReadPointer == gbRxBufferWritePointer)
            {
                if(ulCounter++ > RX_TIMEOUT_COUNT2)
                {
                    return bCount;
                }
            }
            bpRxBuffer[bCount] = gbpInterruptRxBuffer[gbRxBufferReadPointer++];
        }
        return bCount;
}

byte TxPacket(byte *bpTxBuffer, byte bID, byte bInstruction, byte *bpParameter, byte bParameterLength)
{
        byte bCount,bCheckSum,bPacketLength;

        bpTxBuffer[0] = 0xff;
        bpTxBuffer[1] = 0xff;
        bpTxBuffer[2] = bID;
        bpTxBuffer[3] = bParameterLength+2; //Length(Paramter,Instruction,Checksum)
        bpTxBuffer[4] = bInstruction;
        for(bCount = 0; bCount < bParameterLength; bCount++)
        {
            bpTxBuffer[bCount+5] = bpParameter[bCount];
        }
        bCheckSum = 0;
        bPacketLength = bParameterLength+4+2;
        for(bCount = 2; bCount < bPacketLength-1; bCount++) //except 0xff,checksum
        {
            bCheckSum += bpTxBuffer[bCount];
        }
        bpTxBuffer[bCount] = ~bCheckSum; //Writing Checksum with Bit Inversion

        RS485_TXD; //Change 485 Direction to Transmission
        for(bCount = 0; bCount < bPacketLength; bCount++)
        {
            TxD80(bpTxBuffer[bCount]);
        }
        while(!TXD_FINISH0); //Wait until TXD Shift register empty  <---
        RS485_RXD;
        return(bPacketLength);
}

void PrintBuffer(byte *bpPrintBuffer, byte bLength)
{
        byte bCount;
        for(bCount = 0; bCount < bLength; bCount++)
```

> Should wait until last data bit transmission is completed.
> Note.: 'Shift register empty' is differ from 'Tx Ready'. Tx Ready just means you can load the data to CPU UART TxD Register. There can be several Tx Buffering registers as what kind of CPU

```
    {
        TxD8Hex(bpPrintBuffer[bCount]);
        TxD8(' ');
    }
}
```

Result
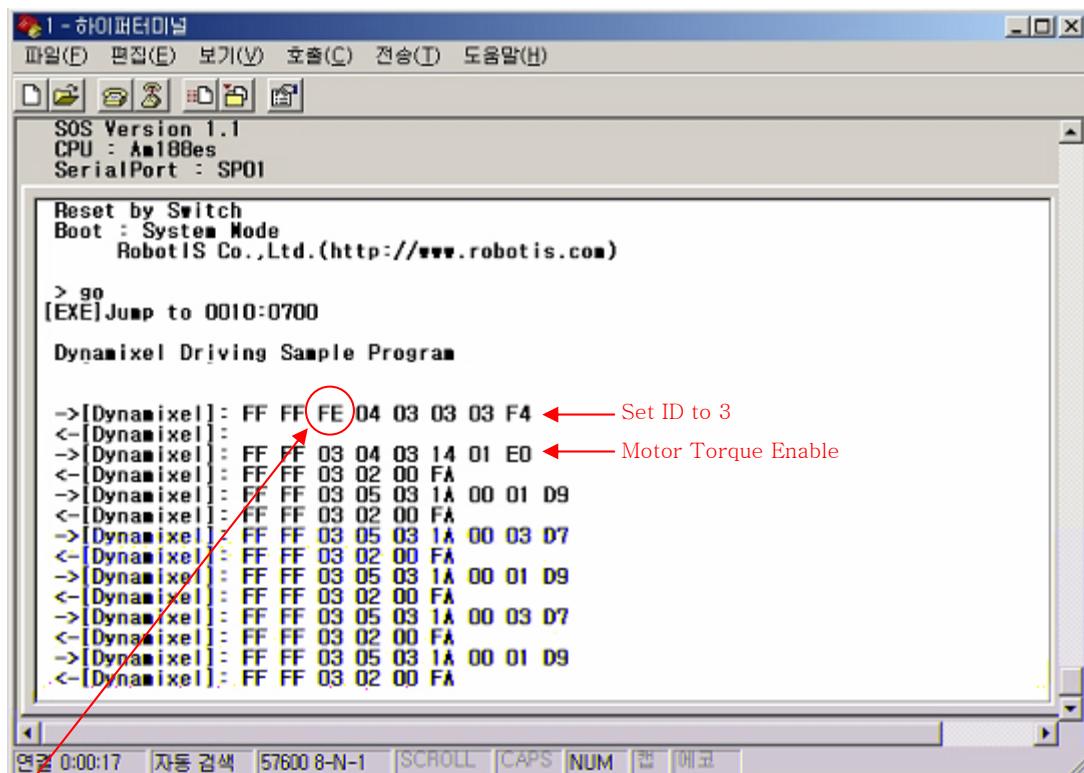
0xFE is BROADCAST_ID, so Dynamixel does not return status packet.(First 2 Instruction Packet)

Connector

Company Name : Molex

Pin Number: 4

Model Number

|	Molex Part Number	Old Part Number
Male	22-03-5045	5267-04
Female	50-37-5043	5264-04

Temperature range : -40°C to +105°C

Contact Insertion Force-max : 14.7N (3.30 lb)

Contact Retention Force-min : 14.7N (3.30 lb)

www.molex.com or www.molex.co.jp for more detail information

Female Connector

Male Connector

Pin No.1
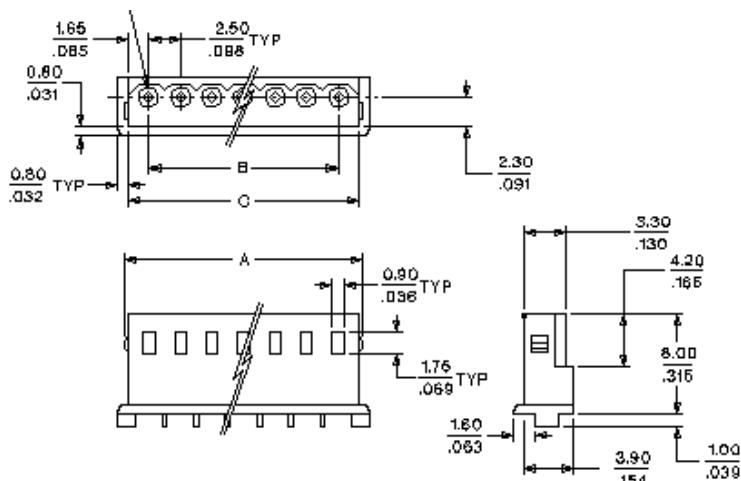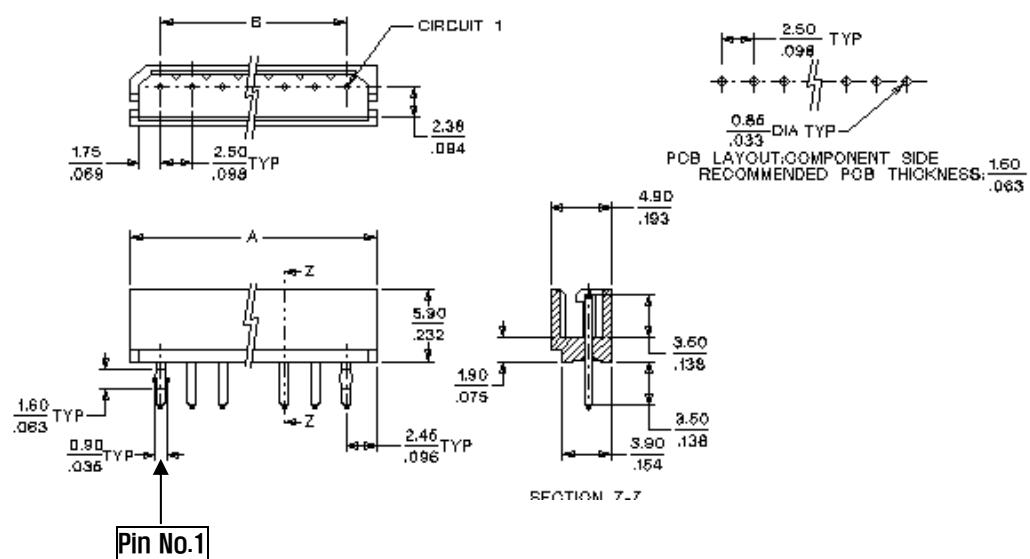
Dimension
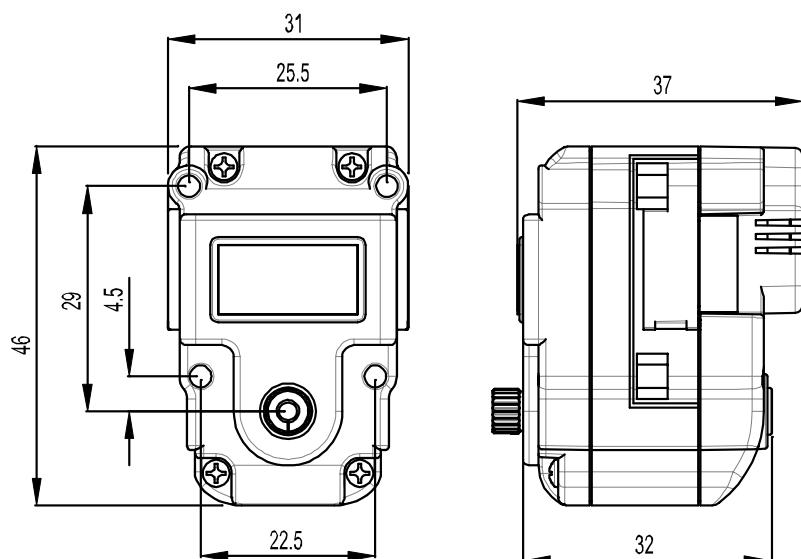
Motor Curve(No reduction gear state)
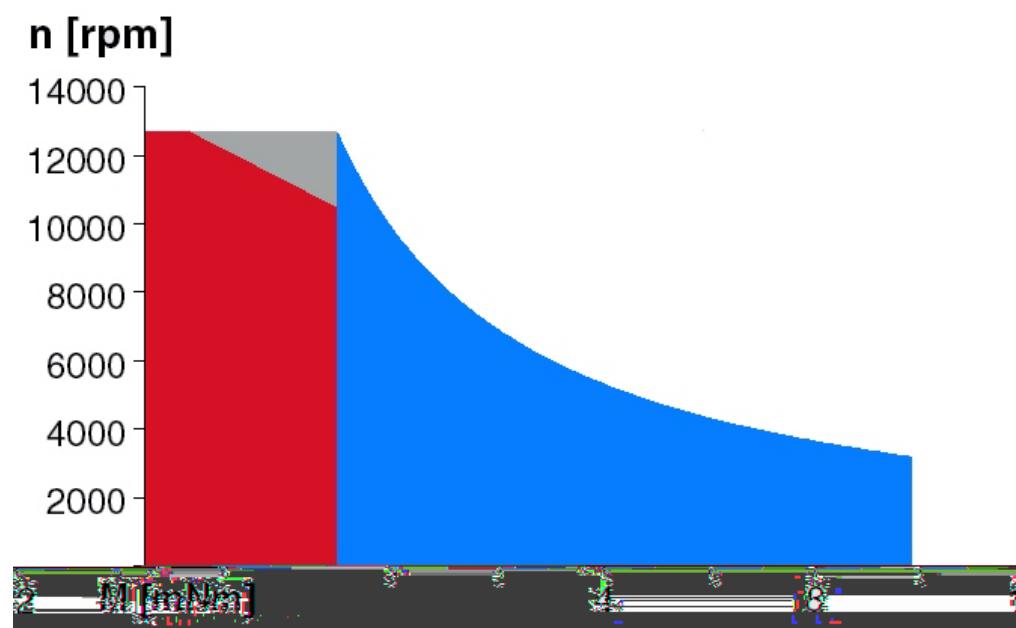
## Optional Frame Application Example

OF116H
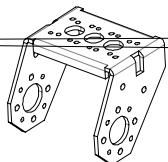
OF116S
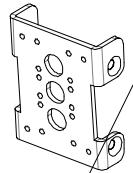
OF116B
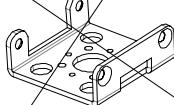
Body to Body Mount
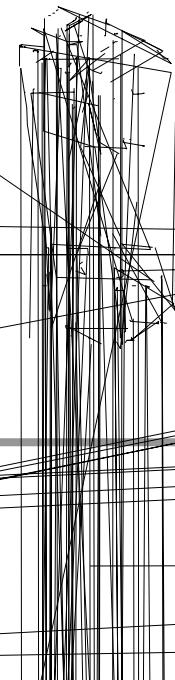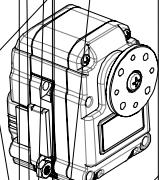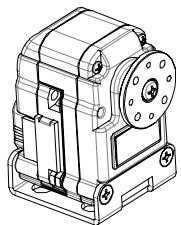
Dynamixel Application Example

CYCLOID Ⅱ