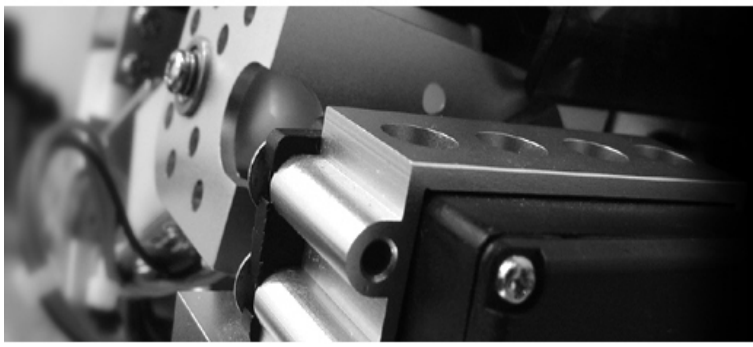
User's Manual 2006-06-14



Closer to Real, ROBOTIS

Sensor Module

Dynamixel AX-S1

Contents

1. Dynamixel AX-S1

1-1. Overview and Characteristics of AX-S1

Dynamixel AX-S1 Dynamixel Sensor Module 'AX-S1' is a Smart Sensor Module that integrates the functions of sound sensor, infrared remote control receiver, infrared distance sensor, light sensor, buzzer, as well as the driver, control unit and network. Compact in size, AX-S1 has various functions and it is made up of special materials that can withstand even the extreme external force. In addition, it can readily recognize subtle changes such as internal temperature, service voltage and other internal conditions and has built-in capability to resolve the situations at hand. Followings are the strengths of the Dynamixel Sensor Module AX-S1.

Precision Control Capability to read sensor that has been detected through 1024 steps resolution

Feedback Feedback capabilities for the values of infrared distance sensor, light sensor, sound sensor.

Alarm System Alarm system that detects out of the range values of internal temperature, torque, service voltage were preset by users (Alarming)

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

Distributed Control Position, velocity, compliance, and torque 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.

Frames Hinge and side mount frame are included as basics. AX-S1 is compatible with AX-12 frames 100%, making it possible to use in various ways. Be cautious as unlike AX-12, Horn part of AX-S1 does not turn, so assemble frame in correct angle with the usage purpose in mind.

Infra-red Sensor It is embedded with three directions infrared sensor, making it possible to detect left/center/right distance angle as well as the light.

Remocon Sensor It has built-in remote control sensor in center, making it possible to transmit and receive infrared data between sensor modules.

Internal Mic It has built-in micro internal microphone, making it possible not only to detect current sound level and maximum loudness but also an ability to count the number of sounds, for instance, the numbers of handclapping

Buzzer Built-in buzzer allows the playback of musical notes and other special note effects.

1-2. Main Specifications

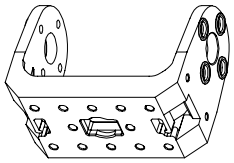
Dynamixel
Networked Sensor Module AX-S1 for Robot Application

Weight	37g
Resolution	10bit (1024)
Voltage	7V~10V (Recommended voltage: 9.6V)
Supply Current	40mA
Operate Temperature	-5℃ ~ +85℃
Command Signal	Digital Packet
Protocol Type	Half duplex Asynchronous Serial Communication (8bit,1stop,No Parity)
Link (Physical)	TTL Level Multi Drop (daisy chain type Connector)
ID	254 ID (0~253)
Communication Speed	7343bps ~ 1 Mbps
Feedback	Infra-red Sensor, Internal Mic, Temperature, Input Voltage, IR Remocon Tx/Rx Data, etc.
Material	Engineering Plastic

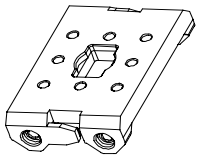
2. Dynamixel Operation

2-1. Mechanical Assembly

Frames Provided The two frames provided with AX-S1 are shown below.

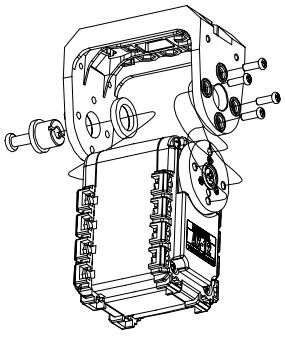


OF-12SH

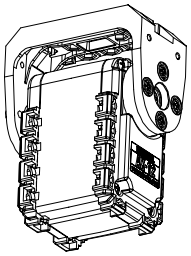


OF-12S

OF-12SH Installation The OF-12SH (hinge frame) can be installed on the AX-12 as the following.



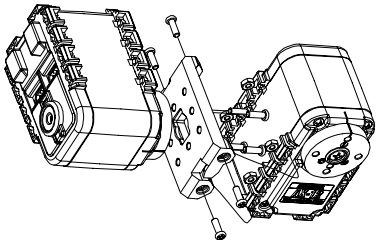
Exploded view



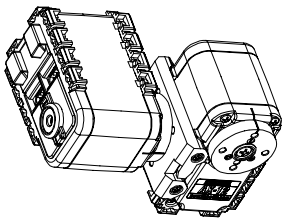
Assembled

OF-12S Installation The OF-12S (side mount frame) can be installed on the AX-12 as the following. The OF-12S can be mounted on any of the three faces (left, right, or under side) of the AX-12 body as needed.

Horn2Body

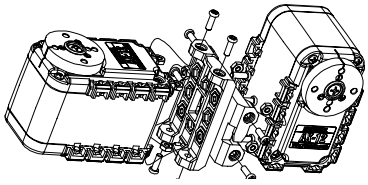


Exploded view

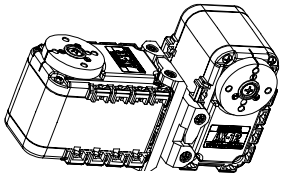


Assembled

Body2Body



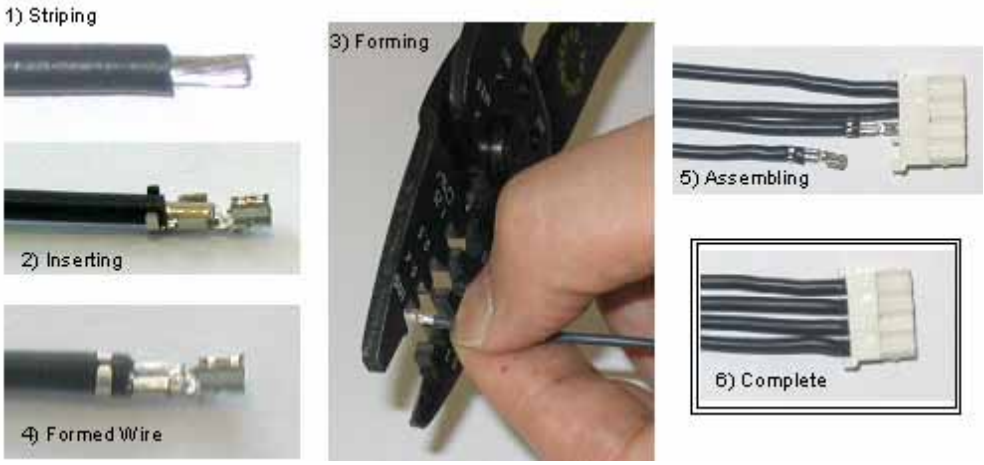
Exploded view



Assembled

2-2. Connector Assembly

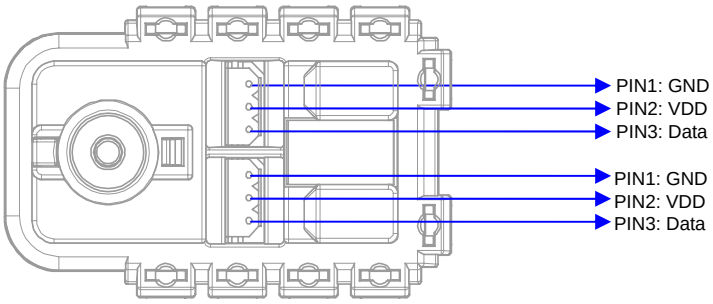
Assemble the connectors as shown below. Attach the wires to the terminals using the correct crimping tool. If you do not have access to a crimping tool, solder the terminals to the wires to ensure that they do not become loose during operation.



2-3. Dynamixel Wiring

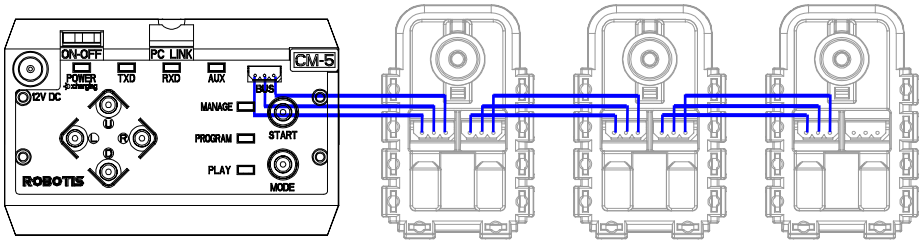
Pin Assignment

The connector pin assignments are as the following. The two connectors on the Dynamixel are connected pin to pin, thus the AX-S1 can be operated with only one connector attached.



Wiring

Connect the AX-2 actuators pin to pin as shown below. Many AX-S1 andAX-12 actuators can be controlled with a single bus in this manner.



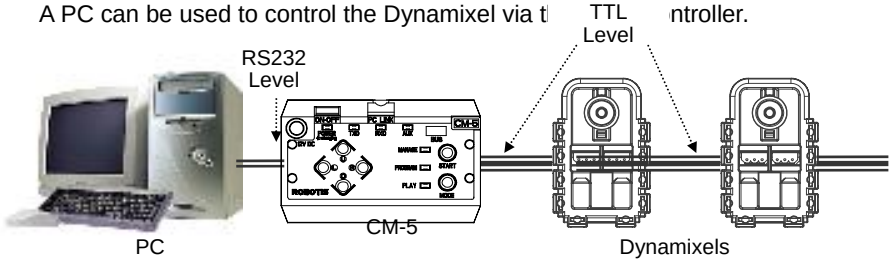
Control Box “CM-5”

Main Controller

To operate the Dynamixel actuators, the main controller must support TTL level half duplex UART. A proprietary controller can be used, but the use of the Dynamixel controller CM-5 is recommended.

PC LINK

A PC can be used to control the Dynamixel via t ntroller.



Bioloid

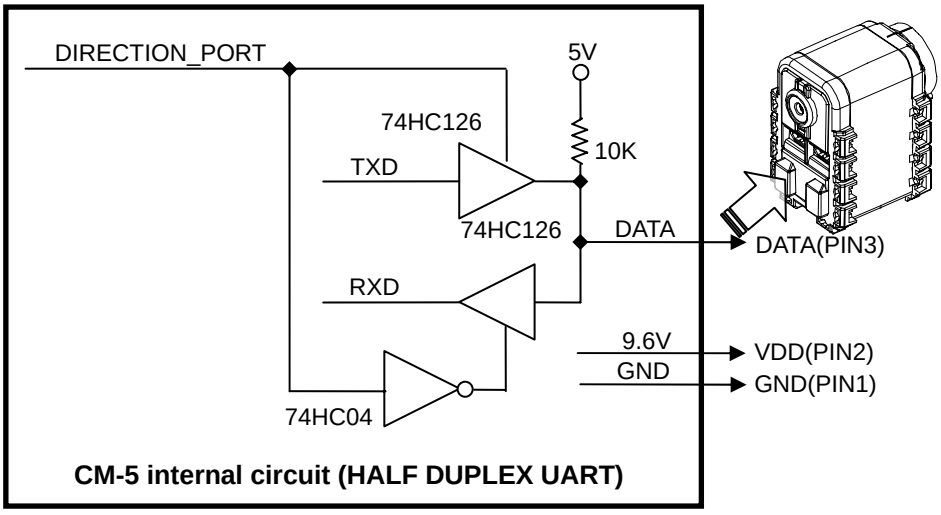
A robot can be built using only the CM-5 controller, a number of AX-12 actuators and AX-S1. An edutainment robotic kit named “Bioloid” is available which is based on the CM-5 controller, the AX-12 actuators and AX-S1

An example of a robot built with Bioloid

For details, please refer to the Bioloid manual.

Connection to UART

To control the Dynamixel actuators, the main controller needs to convert its UART signals to the half duplex type. The recommended circuit diagram for this is shown below.



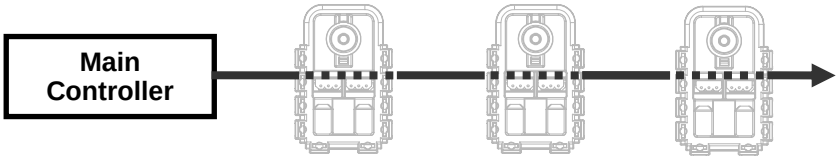
CM-5 internal circuit (HALF DUPLEX UART)

The power is supplied to the Dynamixel actuator from the main controller through Pin 1 and Pin 2 of the Molex3P connector. (The circuit shown above is presented only to explain the use of half duplex UART. The CM-5 controller already has the above circuitry built in, thus the Dynamixel actuators can be directly connected to it)

The direction of data signals on the TTL level TxD and RxD depends on the DIRECTION_PORT level as the following.

• When the DIRECTION_PORT level is High: the signal TxD is output as Data
• When the DIRECTION_PORT level is Low: the signal Data is input as RxD

Half Duplex UART A multi-drop method of connecting multiple Dynamixel actuators to a single node is possible by using the half duplex UART. Thus a protocol that does not allow multiple transmissions at the same time should be maintained when controlling the Dynamixel actuators.



[Multi Drop Link]

Caution Please ensure that the pin assignments are correct when connecting the Dynamixel actuators. Check the current consumption when powering on. 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.

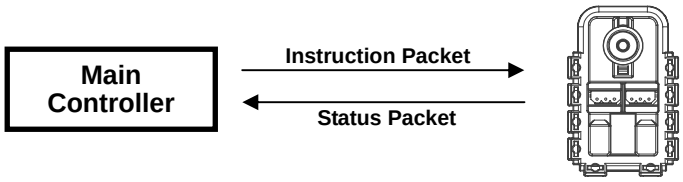
Inspection If the above operation was not successful, then check the connector pin assignment and the voltage/current limit of the power supply.

3. Communication Protocol

3-1. Communication Overview

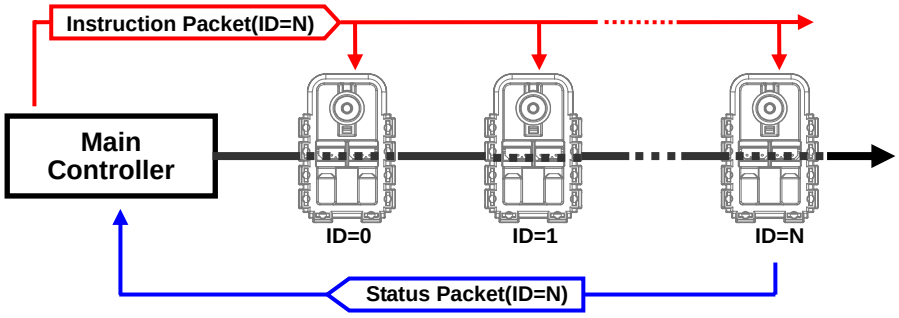
Packet

The main controller communicates with the Dynamixel units by sending and receiving data packets. There are two types of packets; the “Instruction Packet” (sent from the main controller to the Dynamixel actuators) and the “Status Packet” (sent from the Dynamixel actuators to the 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 unit with this ID value will return its respective status packet and perform the required instruction



Unique ID

If multiple Dynamixel units have the same ID value, multiple packets sent simultaneously collide, resulting in communication problems. Thus, it is imperative that no Dynamixel units share the same ID in a network node.

Protocol

The Dynamixel actuators communicate through asynchronous serial communication with 8 bit, 1 stop bit and no parity.

3-2. Instruction Packet

The Instruction Packet is the packet sent by the main controller to the Dynamixel units to send commands. The structure of the Instruction Packet is as the following.

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

The meanings of each packet byte definition are as the following.

0XFF 0XFF The two 0XFF bytes indicate the start of an incoming packet.

ID The unique ID of a Dynamixel unit. There are 254 available ID values, ranging from 0X00 to 0XFD.

Broadcasting ID ID 0XFE is the Broadcasting ID which indicates all of the connected Dynamixel units. Packets sent with this ID apply to all Dynamixel units on the network. Thus packets sent with a broadcasting ID will not return any status packets.

LENGTH The length of the packet where its value is “Number of parameters (N) + 2”

INSTRUCTION The instruction for the Dynamixel actuator to perform.

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

CHECK SUM The computation method for the ‘Check Sum’ is as the following.

Check Sum = ~ (ID + Length + Instruction + Parameter1 + ... Parameter N)
If the calculated value is larger than 255, the lower byte is defined as the checksum value.
~ represents the NOT logic operation.

3-3. Status Packet(Return Packet)

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

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

The meanings of each packet byte definition are as the following.

0XFF 0XFF The two 0XFF bytes indicate the start of the packet.

ID The unique ID of the Dynamixel unit returning the packet.

LENGTH The length of the packet where its value is “Number of parameters (N) + 2”

ERROR The byte representing ERROR sent from the Dynamixel unit. The meaning of each bit is as the following.

Bit	Name	Details
Bit 7	0	-
Bit 6	Instruction Error	Set to 1 if an undefined instruction is sent or an action instruction is sent without a Reg_Write instruction.
Bit 5	0	
Bit 4	Checksum Error	Set to 1 if the checksum of the instruction packet is incorrect
Bit 3	Range Error	Set to 1 if the instruction sent is out of the defined range
Bit 2	Overheating Error	Set to 1 if the internal temperature of the Dynamixel unit is above the operating temperature range as defined 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 operating voltage range as defined in the control table.

PARAMETER0…N Used if additional information is needed

CHECK SUM

The computation method for the 'Check Sum' is as the following.

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

If the calculated value is larger than 255, the lower byte is defined as the checksum value. ~ represents the NOT logic operation.

3-4. Control Table

	Address	Item	Access	Initial Value
EEPROM Area	0(0X00)	Model Number(L)	RD	13(0x0D)
	1(0X01)	Model Number(H)	RD	0(0x00)
	2(0X02)	Version of Firmware	RD	?
	3(0X03)	ID	RD,WR	100(0x64)
	4(0X04)	Baud Rate	RD,WR	1(0x01)
	5(0X05)	Return Delay Time	RD,WR	250(0xFA)
	6(0X06)	(Reserved)	RD,WR	255(0xFF)
	7(0X07)	(Reserved)	RD,WR	3(0x03)
	8(0X08)	(Reserved)	RD,WR	255(0xFF)
	9(0X09)	(Reserved)	RD,WR	3(0x03)
	10(0x0A)	(Reserved)	-	0(0x00)
	11(0X0B)	the Highest Limit Temperature	RD,WR	100(0x64)
	12(0X0C)	the Lowest Limit Voltage	RD,WR	60(0X3C)
	13(0X0D)	the Highest Limit Voltage	RD,WR	190(0xBE)
	14(0X0E)	(Reserved)	RD,WR	255(0XFF)
	15(0X0F)	(Reserved)	RD,WR	3(0x03)
	16(0X10)	Status Return Level	RD,WR	2(0x02)
	17(0X11)	(Reserved)	RD,WR	4(0x04)
	18(0X12)	(Reserved)	RD,WR	4(0x04)
	19(0X13)	(Reserved)	RD,WR	0(0x00)
	20(0X14)	Obstacle Detected Compare Value	RD,WR	32(0x20)
	21(0X15)	Light Detected Compare Value	RD,WR	32(0x20)
	22(0X16)	(Reserved)	RD,WR	32(0x20)
	23(0X17)	(Reserved)	RD	3(0x03)
RAM Area	24(0X18)	(Reserved)	RD,WR	0(0x00)
	25(0X19)	(Reserved)	RD,WR	0(0x00)
	26(0X1A)	Left IR Sensor Data	RD	?
	27(0X1B)	Center IR Sensor Data	RD	?
	28(0X1C)	Right IR Sensor Data	RD	?
	29(0X1D)	Left Luminosity	RD	?
	30(0X1E)	Center Luminosity	RD	?
	31(0X1F)	Right Luminosity	RD	?
	32(0X20)	Obstacle Detection Flag	RD	?
	33(0X21)	Luminosity Detection Flag	RD	?
	34(0X22)	(Reserved)	RD,WR	0
	35(0X23)	Sound Data	RD,WR	?
	36(0X24)	Sound Data Max Hold	RD,WR	?
	37(0X25)	Sound Detected Count	RD,WR	?
	38(0X26)	Sound Detected Time(L)	RD,WR	?
	39(0X27)	Sound Detected Time(H)	RD,WR	?
	40(0X28)	Buzzer Index	RD,WR	?
	41(0X29)	Buzzer Time	RD,WR	?
	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)	IR Remocon Arrived	RD	0(0x00)
	47(0x2F)	Lock	RD,WR	0(0x00)
	48(0x30)	IR Remocon RX Data 0	RD	?
	49(0x31)	IR Remocon RX Data 1	RD	?
	50(0x32)	IR Remocon TX Data 0	RD,WR	?
	51(0x33)	IR Remocon TX Data 1	RD,WR	?
	52(0x34)	Obstacle Detected Compare	RD,WR	?
	53(0x35)	Light Detected Compare	RD,WR	?

Control Table The Control Table contains information on the status and operation of the Dynamixel actuator. The Dynamixel actuator is operated by writing values to its control table and its status is checked by reading values off its control table.

RAM and EEPROM The data values for the RAM area will be set to the default initial values whenever the power is turned on. However, the data values for the EEPROM area are non-volatile and will still remain even after the power is turned off.

Initial Value The Initial Value column on the right side of the control table shows the Factory Default Values for the case of EEPROM area data, and shows the initial value when the power is turned on for the case of RAM area data.

The following explains the meaning of data stored in each of the addresses in the control table.

Address 0x00,0x01 **Model Number**. For AX-S1, the value is 0X000D(13).

Address 0x02 **Firmware Version**.

Address 0x03 **ID**. The unique ID number assigned to each Dynamixel actuators for identifying them. Different IDs are required for each Dynamixel actuators that are on the same network.

Address 0x04 **Baud Rate.** Determines the communication speed. The computation is done by the following formula.

Speed (BPS) = 2000000 / (Address4 + 1)

Data Value for each Major Baud Rate

Address4	Set BPS	Goal 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 tolerance of UART communication.

Address 0x05 **Return Delay Time.** The time it takes for the Status Packet to return after the Instruction Packet is sent. The delay time is given by 2uSec * Address5 value.

Address 0x0B **the Highest Limit Temperature.** The upper limit of the Dynamixel actuator's operating temperature. If the internal temperature of the Dynamixel actuator gets higher than this value, the Over Heating Error Bit (Bit 2 of the Status Packet) will return the value 1, and an alarm will be set by Address 17, 18. The values are in Degrees Celsius

Address 0x0C,0x0D **the Lowest (Highest) Limit Voltage.** The upper and lower limits of the Dynamixel actuator's operating voltage. If the present voltage (Address 42) is out of the specified range, a Voltage Range Error Bit (Bit 0 of the Status Packet) will return the value 1, and an alarm will be set by Address 17, 18. The values are 10 times the actual voltage value. For example, if the Address 12 value is 80, then the lower voltage limit is set to 8V.

Address 0X10 **Status Return Level.** Determines whether the Dynamixel actuator will return a Status Packet after receiving an Instruction Packet.

Address16	Returning the Status Packet
0	Do not respond to any instructions
1	Respond only to READ_DATA instructions
2	Respond to all instructions

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

Address 0x14 **Obstacle Detected Compare Value** Dynamixel Sensor Module sets the standard value for the object detection that is in the direct line of object sensor parameter. If the infrared sensor value is greater than a standard value, as it indicates an obstacle within the set distance, the bit is set to a value of "1" in corresponding to sensor of IR Obstacle Detected, Address 0x20, and conversely, when the sensor value is lower than a standard value, it is set to "0."

The Obstacle Detected Compare Value is allocated in the ROM (Address 0x14) and RAM (Address 0x34) and when the power switched on, the value of EEPROM is copied to RAM.

Address 0x15 **Light Detected Compare Value** Dynamixel Sensor Module sets the standard value for the light detection that is in the direct line of infrared sensor parameter. If the light sensor value is greater than a standard value, as it indicates a light that is brighter than set light parameter, the bit is set to a value of “1” in corresponding to sensor of Light Detected, and conversely, it is set to “0” when it is lower than a standard value.

The Light Detected Compare Value is allocated in the ROM (Address 0x15) and RAM (Address 0x35) and when the power switched on, the value of EEPROM is copied to RAM.

Subsequent Address 0x18 is in RAM domain.

Address 0x1A~0x1C **Infrared Sensor Data (Left/Center/Right)** It is the infrared sensor value of the Dynamixel Sensor Module for measuring distance. The infrared sensor of AX-S1 emits high frequency Infrared and the emitted ray bounces off an object or wall to return to the IR sensor. The Infrared receiver of AX-S1 measures amount of infrared returned. High value will be acquired when an object or wall is near the sensor. Measured value ranges from 0~255. Only 255 will be acquired until a certain distance. Due to the innate properties of infrared measurement method, value of reflected Infrared ray amount might differ depending on the color of an object or surface texture.

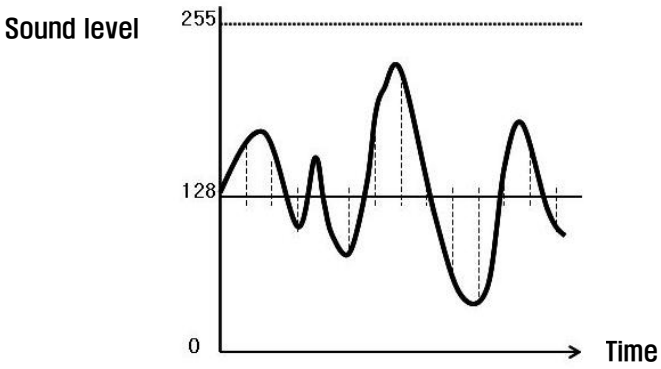
Address 0x1D~0x1F **Luminosity (Left/Center/Right)** It is the light sensor value of the Dynamixel Sensor Module. The technological concept is similar to the infrared sensor. However, this sensor only measures amount of infrared ray emitted from source of illumination. Therefore, light sensor value can be measured from illuminations, such as incandescent bulb, emitting large amount of infrared. Lighter or candle light can be measured from short distance as well. Measured value ranges from 0~255.

Address 0x20 **Obstacle Detection Flag** When the value of infrared distance sensor becomes larger than the Obstacle Detected Compare Value, the AX-S1 recognizes existence of an object and sets object detection bit to 1. Refer to the below table for bit representation of each sensor.

Bit	Representation
Bit 2	An object is detected on the Right Sensor /Light Detected
Bit 1	An object is detected on the Center Sensor /Light Detected
Bit 0	An object is detected on the Left Sensor /Light Detected

Address 0x21 **Luminosity Detection Flag** When the value of light sensor becomes larger than the light detected compare value, the AX-S1 recognizes existence of source of illumination and sets luminosity detection flag bit to 1. Bit representation of each sensor is the same with bit representation of object detection flag setting. (Refer to Address 0x20)

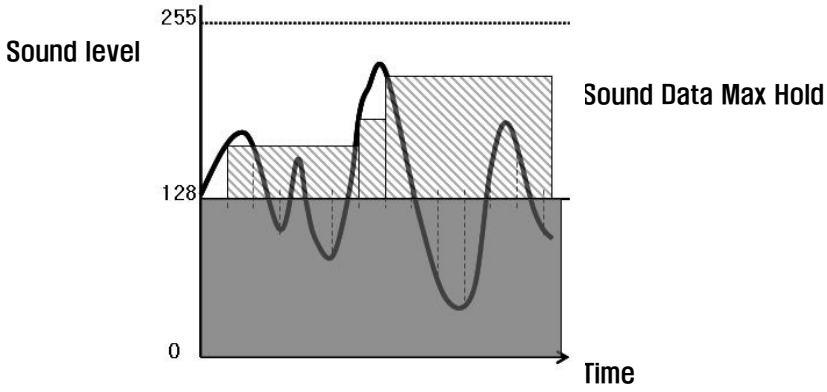
Address 0x23 **Sound Data** It represents intensity of sound waves detected through the microphone of AX-S1. As shown in the illustration below, the magnitude of sound wave fluctuates. Value measured during noiseless state is around 127~128 (0x7F~0x80) and value ranging from 0 to 255 (0xFF) will be measured for noisy state. Sound wave will be measured at the frequency of 3800 input per second.



Address 0x24

Sound Data Max Hold AX-S1 has put aside a value for loudest sound. That is, when the present sound data exceeds the Sound Data Max Hold value, the present sound data will be copied as the Sound Data Max Hold.

Therefore, sound data less than 128 will be ignored and loudest sound intensity will be updated. Below illustration explains the details.



Be cautious as the Sound Data Max Hold is 255 (0xFF) and there is no value that can represent intensity of loudness greater than the optimal loudness, and thus, 255 (0xFF) will be maintained as the Sound Data Max Hold.

Therefore, value of the Sound Data Max Hold should be set at “0” for measuring the value of maximum loudness,

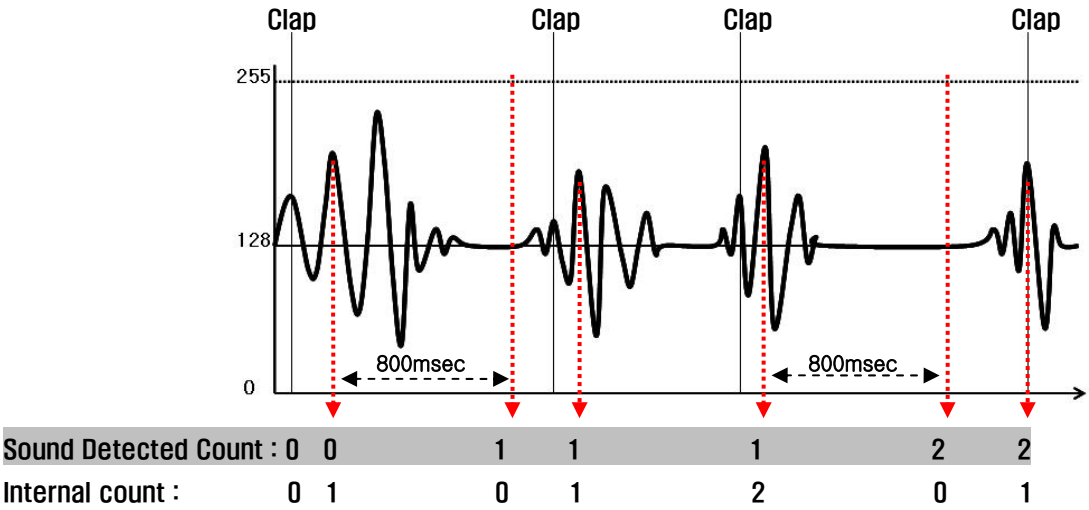
Address 0x25

Sound Detected Count AX-S1 has a counter that counts occurrence of loud sound

exceeding standard level. As an example, number of handclap can be counted by using this.

However, the counter will not count for next 80msec after counting once to prevent a single handclap to be recognized as multiple claps. 800msec after the last count, the value of sound detection frequency counter will be saved.

Timeline of sound detection frequency will be counted internally and then the value of sound detection frequency will be saved after 800msec. After saving, the sound detection frequency value will reset to 0. Below illustration explains the details.

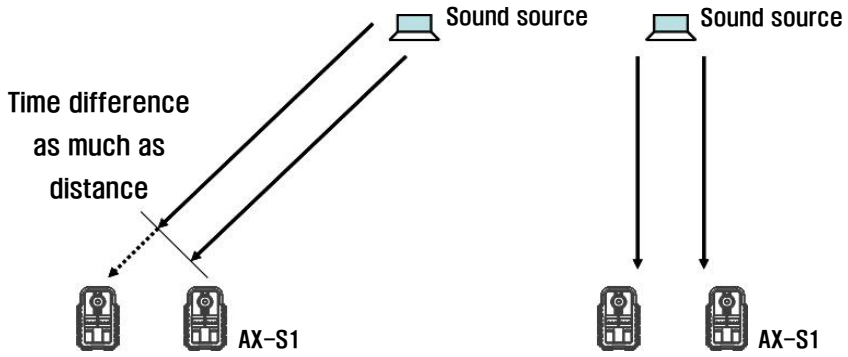


Address 0x26, 0x27 **Sound Detected Time** Anytime Sensor Module AX-S1 counts of sound detection, it saves the time of sound occurrences. This function exists to detect the direction of sound, and thus, it needs at least two AX-S1s; and by using speed of sound (around 343m/sec in 20℃) it uses the time differences of sound arrival in microphone of two AX-S1s.

Sound Detected Time is internally counted (counts 0~65535 repeatedly) and anytime Sound Detected Count is increased, it saves the counted value. Therefore, by placing the two AX-S1s in appropriate distance, and by simultaneously using broadcasting command and initializing to 0 value, the time differenced sound occurs corresponding to sound direction.

If the placement is face to face, the time differences will be almost simultaneous,

however, for the placement that has been set in side angle, the time differences will be influenced by the distances of AX-S1s. With this concept, it can estimate the direction of the sound. Below are detailed illustrations.



(1) Sound from angle (2) Sound from front

It counts completely every 4.096msec and it recounts again from 0. Therefore, in calculating the sound of speed, for every count, sound moves 0.02mm and two AX-S1's distance must be within 70cm.

For example, when two AX-S1s is 10cm apart, by using above method estimation, two AX-S1s' sound detected time difference can be maximum of 5,000. (If it is 5,000, it signifies that sound source is completely from the 90 angle or from the right side.)

Address 0x28 **Buzzer Index** All AX-S1 has built-in buzzer and thus, can playback the simple notes. Buzzer can play up to 52 notes and as it has whole and semitone in each octave, it can playback various melody sounds. The buzzer index value is assigned as follows.

Buzzer index	Melody notes	Buzzer index	Melody notes	Buzzer index	Melody notes	Buzzer index	Melody notes
0	la	13	la#	26	si	39	do
1	la#	14	si	27	do	40	do#
2	si	15	do	28	do#	41	re
3	do	16	do#	29	re	42	re#
4	do#	17	re	30	re#	43	mi
5	re	18	re#	31	mi	44	fa
6	re#	19	mi	32	fa	45	fa#
7	mi	20	fa	33	fa#	46	sol
8	fa	21	fa#	34	sol	47	sol#
9	fa#	22	sol	35	sol#	48	la

10	sol	23	sol#	36	la	49	la#
11	sol#	24	la	37	la#	50	si
12	la	25	la#	38	si	51	do

Address 0x29 **Buzzer Time** AX-S1 has a capability that controls the time interval of buzzer sound. Controllable within 0.1 second unit, the minimum length of time is 0.3 second and the maximum length of time is 5.0 seconds. That is, if user inputs the value of 0~3, the buzzer goes off in 0.3 second, whereas, if the input value is 50 or above, it goes off in 5 second. When the buzzer sound completes, the value automatically initializes back to 0. There are two special features of AX-S1 buzzer time.

First is the function that sets the buzzer to sound constantly. If user inputs value of 254 on buzzer time and input the melody note number on buzzer index, the buzzer sounds the note constantly. To stop the buzzer, input 0 on buzzer time.

The second function plays back the special notes. If user inputs value of 255 on buzzer time and value between 0~26 on buzzer index, 27 various melodies is replayed corresponding to each number. When the melody playback is finished, the value automatically initializes back to 0.

Address 0x2A **Present Voltage.** Currently authorized voltage of Dynamixel AX-S1. It reality, it is multiple of 10 of actual voltage. That is, if 10V, it is read as 100(0x64).

Address 0x2B **Present Temperature.** Inner Celsius temperature of Dynamixel AX-S1

Address 0x2C **Registered Instruction.** If it is registered by the command of REG_WRITE, it is set to 1, and if it is registered by Action command, it is changed to 0 after command is completed.

Address 0x2E **IR Remocon Arrived** **AX-**S1 Sensor Module has infrared sensor module built-in in center and thus, it allows infrared remocon communication between AX-S1's. 2 byte transmission is possible.

Be cautious, however, as the infrared emitter is built into left/center/right, it can transmit infrared remocon in all directions, but, as infrared remocon sensor is built in only in center, its remocon data transmission is limited to certain angle.

I When Infrared remocon data is received by sensor, IR Remocon Arrived value changes to 2, signaling 2 byte transmission. If you read IR Remocon RX data, the IR Remocon

and automatically initializes back to 0.

Address 0x2F **Lock.** If the setting is set to 1, it can only write in range from Address 0X18 to Address0x23 and writing to other ranges is forbidden. Once it is locked, it can be unlocked only after power off. (power down)

Address 0x30,0x31 **IR Remocon RX Data** Address where data from infrared remocon sensor is saved. It reads the value and the IR Remocon Arrived value automatically initializes back to 0.

Address 0x32,0x33 **IR Remocon TX Data** Address where remocon data that will be transmitted via infrared emitter is written to. Upon writing of 2 byte value, remocon data is immediately transmitted.

Address 0x34 **Obstacle Detected Compare Value** Control Table RAM Range where obstacle detected compare value of Address 0x14 is saved.

The IR sensors of AX-S1 emit powerful infrared rays to detect an object at a long distance. It is impossible to detect an object in a short distance around 5cm since it always has maximum value in short distance

To prevent this, AX-S1 support low sensitive mode to detect precise value in a short distance. If the Obstacle Detected Compare Value is 0, it converts to low sensitive mode. The low sensitive mode has very weak long-distance sensing capability but it is possible to detect precise and sensitive short-distance detection not to saturate maximum value.

Address 0x35 **Light Detected Compare Value** Control Table RAM Range where light detected compare value of Address 0x15 is saved

Range

Each data has set value where their valid range is defined. Outside of this range, their write command will return Error. Below table indicates the length for writing and its range. 16 bit data is indicated (L) and (H) and as 2 byte. This 2 byte must be written as one in instruction packet.

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)
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)
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
20(0X14)	Obstacle Detected Compare	1	0	255(0xff)
21(0X15)	Light Detected Compare	1	0	255(0xff)
36(0X24)	Sound Data Max Hold	1	0	255(0xff)
37(0X25)	Sound Detected Count	1	0	255(0xff)
38(0X26)	Sound Detected Time	2	0	65535(0xffff)
40(0X28)	Buzzer Index	1	0	255(0xff)
41(0X29)	Buzzer Time	1	0	255(0xff)
44(0X2C)	Registered Instruction	1	0	1
47(0X2F)	Lock	1	1	1
50(0X32)	IR Remocon TX Data	2	0	65535(0xffff)

[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 for obtaining a Status Packet	0x01	0
READ DATA	Reading values in the Control Table	0x02	2
WRITE DATA	Writing values to the Control Table	0x03	2 ~
REG WRITE	Similar to WRITE_DATA, but stays in standby mode until the ACION instruction is given	0x04	2 ~
ACTION	Triggers the action registered by the REG_WRITE instruction	0x05	0
RESET	Changes the control table values of the Dynamixel actuator to the Factory Default Value settings	0x06	0
SYNC WRITE	Used for controlling many Dynamixel actuators at the same time	0x83	4~

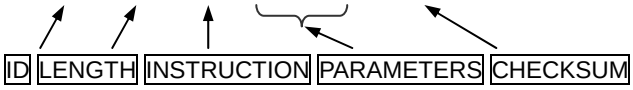
4-1. WRITE_DATA

Function To write data into the control table of the Dynamixel actuator
Length N+3 (N is the number of data to be written)
Instruction 0X03
Parameter1 Starting address of the location where the data is to be written
Parameter2 1st data to be written
Parameter3 2nd data to be written
Parameter N+1 Nth data to be written

Example 1 **Setting the ID of a connected Dynamixel actuator to 1**

Write 1 to address 3 of the control table. The ID is transmitted using the Broadcasting ID (0xFE).

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



Because it was transmitted with a Broadcast ID (0XFE), no status packets are returned.

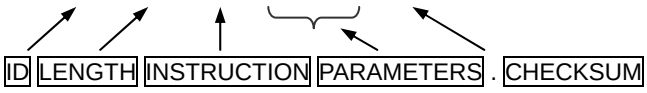
4-2. READ_DATA

Function	Read data from the control table of a Dynamixel actuator
Length	0X04
Instruction	0X02
Parameter1	Starting address of the location where the data is to be read
Parameter2	Length of the data to be read

Example 2 **Reading the internal temperature of the Dynamixel actuator with an ID of 1**

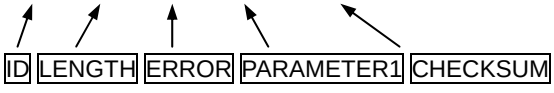
Read 1 byte from address 0x2B of the control table.

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



The returned Status Packet will be as the following.

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



The data read is 0x20. Thus the current internal temperature of the Dynamixel actuator is approximately 32°C (0X20).

4-3. REG_WRITE과 ACTION

4-3-1. REG_WRITE

Function The 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 stored in the Buffer and the Write instruction is under a standby status. At this time, the Registered Instruction register (Address 0x2C) is set to 1. After the Action Instruction Packet is received, the registered Write instruction is finally executed.

Length N+3 (N is the number of data to be written)
Instruction 0X04
Parameter1 Starting address of the location where the data is to be written
Parameter2 1st data to be written
Parameter3 2nd data to be written
Parameter N+1 Nth data to be written

4-3-2. ACTION

Function Triggers the action registered by the REG_WRITE instruction
Length 0X02
Instruction 0X05
Parameter NONE

The ACTION instruction is useful when multiple Dynamixel actuators need to move simultaneously. When controlling multiple Dynamixel actuator units, slight time delays can occur between the 1st and last units to receive an instruction. The Dynamixel actuator handles this problem by using the ACTION instruction.

Broadcasting The Broadcast ID (0XFE) is used when sending ACTION instructions to more than two Dynamixel actuators. Note that no packets are returned by this operation.

4-4. PING

Function Does not command any operations. Used for requesting a status packet or to check the existence of a Dynamixel actuator with a specific ID.

Length 0X02

Instruction 0X01

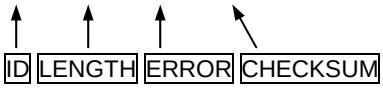
Parameter NONE

Example 3 **Obtaining the status packet of the Dynamixel actuator with an ID of 1**

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

The returned Status Packet is as the following

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



Regardless of whether the Broadcasting ID is used or the Status Return Level (Address 16) is 0, a Status Packet is always returned by the PING instruction.

4-5. RESET

Function Changes the control table values of the Dynamixel actuator to the Factory Default Value settings

Length 0X02

Instruction 0X06

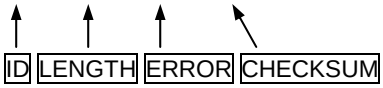
Parameter NONE

Resetting the Dynamixel actuator with an ID of 0

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

The returned Status Packet is as the following

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



Note the ID of this Dynamixel actuator is now changed to 1 after the RESET instruction

4-6. SYNC WRITE

Function Used for controlling many Dynamixel actuators at the same time. The communication time decreases by the Synch Write instruction since many instructions can be transmitted by a single instruction. However, you can use this instruction only when the lengths and addresses of the control table to be written to are the same. Also, the broadcasting ID needs to be used for transmitting.

ID	0XFE
Length	(L + 1) * N + 4 (L: Data length for each Dynamixel actuator, N: The number of Dynamixel actuators)
Instruction	0X83
Parameter1	Starting address of the location where the data is to be written
Parameter2	The length of the data to be written (L)
Parameter3	The ID of the 1st Dynamixel actuator
Parameter4	The 1st data for the 1st Dynamixel actuator
Parameter5	The 2nd data for the 1st Dynamixel actuator
…	
Parameter L+3	The Lth data for the 1st Dynamixel actuator
Parameter L+4	The ID of the 2nd Dynamixel actuator
Parameter L+5	The 1st data for the 2nd Dynamixel actuator
Parameter L+6	The 2nd data for the 2nd Dynamixel actuator
…	
Parameter 2L+4	The Lth data for the 2nd Dynamixel actuator

Data for the 1st Dynamixel actuator

Data for the 2nd Dynamixel actuator

Example 5 **Setting the following positions and velocities for 4 Dynamixel actuators**

Dynamixel actuator with an ID of 0: to position 0X010 with a speed of 0X150
Dynamixel actuator with an ID of 1: to position 0X220 with a speed of 0X360
Dynamixel actuator with an ID of 2: to position 0X030 with a speed of 0X170
Dynamixel actuator with an ID of 0: to position 0X220 with a speed of 0X380

Instruction Packet : 0XFF 0XFF 0XFE 0X18 0X83 0X1E 0X04 0X00 0X10 0X00 0X50 0X01 0X01 0X20 0X02 0X60 0X03 0X02 0X30 0X00 0X70 0X01 0X03 0X20 0X02 0X80 0X03 0X12
No status packets are returned since the Broadcasting ID was used.

5. Example

We will give an example of Dynamixel AX-S1 with following setup parameter. Reset state ID=100, Baudrate = 1MBPS

.

Example 6 Dynamixel AX-S1 that has ID 100 reads the Model Number and Firmware Version

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

Communication ->[Dynamixel]:FF FF 64 04 02 00 03 95 (LEN:008)
<-[Dynamixel]:FF FF 64 05 00 0D 00 12 77 (LEN:009)

Status Packet Result Model Number = 13(0x0D)(in case of AX-S1) Firmware Version = 0x12

Example 7 Dynamixel AX-S1 that has ID 100 changes ID to 0.

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

Communication ->[Dynamixel]:FF FF 64 04 03 03 00 91 (LEN:008)
<-[Dynamixel]:FF FF 64 02 00 99 (LEN:006)

Status Packet Result NO ERROR

Example 8 Change the Baud Rate of Dynamixel to 57600 bps.

Instruction Packet Instruction = WRITE_DATA, Address = 0x04, DATA = 0x22

Communication ->[Dynamixel]:FF FF 64 04 03 04 22 6E (LEN:008)
<-[Dynamixel]:FF FF 64 02 00 99 (LEN:006)

Status Packet Result NO ERROR

Example 9 Dynamixel that has ID 100 resets the Return Delay Time to 4uSec

Return Delay Time Value of 1 is applicable to 2uSec.

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

Communication ->[Dynamixel]:FF FF 64 04 03 05 02 8D (LEN:008)
<-[Dynamixel]:FF FF 64 02 00 99 (LEN:006)

Status Packet Result NO ERROR

It is good idea to set the Return Delay Time to minimum value within allowable range in the main controller.

Example 10 Dynamixel that has ID 100 resets the distance sensor standard value to 60.

Instruction Packet Instruction = WRITE_DATA, Address = 0x34, DATA = 0x3C

Communication ->[Dynamixel]:FF FF 64 04 03 34 3C 24 (LEN:008)
<-[Dynamixel]:FF FF 64 02 00 99 (LEN:006)

Status Packet Result NO ERROR

Example 11 Dynamixel that has ID 100 resets the maximum value of temperature to 80°

Instruction Packet Instruction = WRITE_DATA, Address = 0x0B, DATA = 0x50

Communication ->[Dynamixel]:FF FF 64 04 03 0B 50 39 (LEN:008)
<-[Dynamixel]:FF FF 64 02 00 99 (LEN:006)

Status Packet Result NO ERROR

Example 12 Dynamixel that has ID 100 sets the voltage to 10V ~ 17V.

10V is represented by 100 (0x64), and 17V by 170 (0xAA).

Instruction Packet Instruction = WRITE_DATA, Address = 0x0C, DATA = 0x64, 0xAA

Communication ->[Dynamixel]:FF FF 64 05 03 0C 64 AA 79 (LEN:009)
<-[Dynamixel]:FF FF 64 02 00 99 (LEN:006)

Status Packet Result NO ERROR

Example 13 Dynamixel that has ID 100 changes the light sensor standard value to 10..

Instruction Packet Instruction = WRITE_DATA, Address = 0x35, DATA = 0x0A

Communication ->[Dynamixel]:FF FF 64 04 03 35 0A 55 (LEN:08)
<-[Dynamixel]:FF FF 64 02 00 99 (LEN:006)

Status Packet Result NO ERROR

Example 14 Dynamixel that has ID 100 sets the parameter so that status packet is never returned.

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

Communication ->[Dynamixel]:FF FF 64 04 03 10 00 84 (LEN:008)
<-[Dynamixel]:FF FF 64 02 00 99 (LEN:006)

Status Packet Result NO ERROR
Status packet is not returned from next instruction.

Example 15 Dynamixel AX-S1 that has ID100 reads the right distance sensor value

Instruction Packet Instruction = READ_DATA, Address = 0x1C, DATA = 0x01

Communication ->[Dynamixel]:FF FF 64 04 02 1C 01 78 (LEN:008)
<-[Dynamixel]:FF FF 64 03 00 21 77 (LEN:007)

Status Packet Result NO ERROR
The right distance sensor value is 0x21

Example 16 Dynamixel AX-S1 that has ID 100 reads the center light sensor value

Instruction Packet Instruction = READ_DATA, Address = 0x1E, DATA = 0x01

Communication ->[Dynamixel]:FF FF 64 04 02 1E 01 76 (LEN:008)
<-[Dynamixel]:FF FF 64 03 00 00 98 (LEN:007)

Status Packet Result NO ERROR
The center light sensor value is 0x00

Example 17 Dynamixel AX-S1 that has ID 100 reads the sound loudness

Instruction Packet Instruction = READ_DATA, Address = 0x23, DATA = 0x01

Communication ->[Dynamixel]:FF FF 64 04 02 23 01 71 (LEN:08)
<-[Dynamixel]:FF FF 64 03 00 7E 1A (LEN:007)

Status Packet Result NO ERROR
The sound loudness value is 0x7E (126)

Example 18 Dynamixel AX-S1 that has ID 100 reads the numbers of sound detect frequency

Instruction Packet Instruction = READ_DATA, Address = 0x25, DATA = 0x01

Communication ->[Dynamixel]:FF FF 64 04 02 25 01 6F (LEN:008)
<-[Dynamixel]:FF FF 64 03 00 02 96 (LEN:007)

Status Packet Result NO ERROR
The number of sound detect frequency is 2.

Example 19 Dynamixel AX-S1 that has ID 100 playbacks special melody 5 times through buzzer

.

Case 1. After writing 0xFF(255) on buzzer sound interval, it writes No. 5 on buzzer note melody.

Instruction Packet ID=100, Instruction = WRITE_DATA, Address = 0x29, DATA = 0xFF

ID=100, Instruction = WRITE_DATA, Address = 0x28, DATA = 0x05

Communication

->[Dynamixel]:FF FF 64 04 03 29 FF 6C (LEN:008)
<-[Dynamixel]:FF FF 64 02 00 99 (LEN:006)
->[Dynamixel]:FF FF 64 04 03 28 05 67 (LEN:008)
<-[Dynamixel]:FF FF 64 02 00 99 (LEN:006)

Status Packet Result NO ERROR

Case 2. Writes buzzer note and buzzer sound interval simultaneously

Instruction Packet ID=100, Instruction = WRITE_DATA, Address = 0x28, DATA = 0x05, 0xFF

Communication

->[Dynamixel]:FF FF 64 05 03 28 05 FF 67 (LEN:009)
<-[Dynamixel]:FF FF 64 02 00 99 (LEN:006)

Status Packet Result NO ERROR

Example 20 Dynamixel that has ID 0 sets the parameter so that it cannot write anywhere except in Address0x18 ~ Address0x23

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

Once locked, the only way to unlock it is to remove the power.

If an attempt is made to access any locked data, an error is returned.

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

Example 21 Dynamixel that has ID 0 sets the minimum output value (punch) to 0x40

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

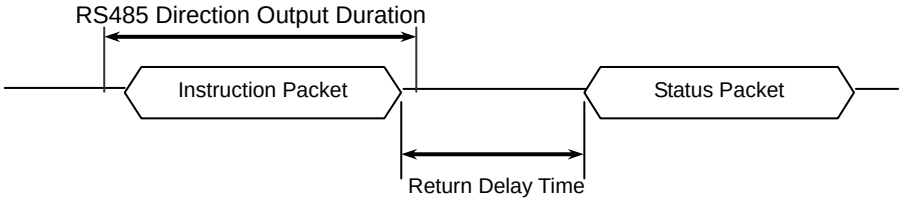
Communication ->[Dynamixel]:FF FF 00 05 03 30 40 00 87 (LEN:009)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR

Appendix

Half duplex UART

Half duplex UART is a serial communication protocol where both TxD and RxD cannot be used at the same time. This method is generally used when many devices need to be connected to a single bus. Since more than one device are connected to the same bus, all the other devices need to be in input mode while one device is transmitting. The Main Controller that controllers the Dynamixel actuators sets the communication direction to input mode, and only when it is transmitting an Instruction Packet, it changes the direction to 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 Direction Port to input mode during the Return Delay Time after sending an instruction packet.

Tx,Rx Direction

For Half Duplex UART, the transmission ending timing is important to change the direction to receiving mode. 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
}
```

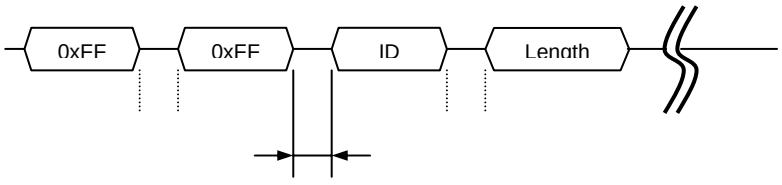
When changing the direction, the TXD_SHIFT_REGISTER_EMPTY_BIT must be checked.

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

```
LINE 1    DIRECTION_PORT = 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   DIRECTION_PORT = RX_DIRECTION; //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 : 2005.5.11
 * Author : BS KIM
 */

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

#define cbi(REG8,BITNUM) REG8 &= ~(_BV(BITNUM))
#define sbi(REG8,BITNUM) REG8 |= _BV(BITNUM)


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

////// For CM-5
#define    RS485_TXD    PORTE    &=    ~_BV(PE3),PORTE    |=    _BV(PE2)
                        //PORT_485_DIRECTION = 1
#define    RS485_RXD    PORTE    &=    ~_BV(PE2),PORTE    |=    _BV(PE3)
                        //PORT_485_DIRECTION = 0
/*
////// For CM-2
#define RS485_TXD PORTE |= _BV(PE2); //_485_DIRECTION = 1
#define RS485_RXD PORTE &= ~_BV(PE2);//PORT_485_DIRECTION = 0
*/
//#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 BIT_RS485_DIRECTION0   0x08   //Port E
#define BIT_RS485_DIRECTION1   0x04   //Port E

#define BIT_ZIGBEE_RESET                PD4   //out : default 1 //PORTD
#define BIT_ENABLE_RXD_LINK_PC          PD5   //out : default 1
#define BIT_ENABLE_RXD_LINK_ZIGBEE      PD6   //out : default 0
#define BIT_LINK_PLUGIN                 PD7   //in, no pull up

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 gbRxBufferReadPointer;
byte gbpRxBuffer[128];
byte gbpTxBuffer[128];
volatile byte gbRxBufferWritePointer;

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

   PortInitialize(); //Port In/Out Direction Definition
   RS485_RXD; //Set RS485 Direction to Input State.
   SerialInitialize(SERIAL_PORT0,1,RX_INTERRUPT);//RS485
                        Initializing(RxInterrupt)
   SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,0);          //RS232
```

```
                    Initializing(None Interrupt)

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

  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);
  bRxPacketLength                                                    =
                    RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpP
                    arameter[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+gbpP
                    arameter[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);
}

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
  DDRE  |=  (BIT_RS485_DIRECTION0|BIT_RS485_DIRECTION1);  //set
                    output the bit RS485direction

  DDRD                                                               |=
                    (BIT_ZIGBEE_RESET|BIT_ENABLE_RXD_LINK_PC|
                    BIT_ENABLE_RXD_LINK_ZIGBEE);

  PORTD &= ~_BV(BIT_LINK_PLUGIN); // no pull up
  PORTD |= _BV(BIT_ZIGBEE_RESET);
  PORTD |= _BV(BIT_ENABLE_RXD_LINK_PC);
  PORTD |= _BV(BIT_ENABLE_RXD_LINK_ZIGBEE);
}

/*
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);
}

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

byte RxPacket(byte bRxPacketLength)
{
#define RX_TIMEOUT_COUNT2     3000L
#define RX_TIMEOUT_COUNT1    (RX_TIMEOUT_COUNT2*10L)
  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_DATA               (UDR1)
#define RXD1_READY              bit_is_set(UCSR1A,7)
#define RXD1_DATA               (UDR1)

#define TXD0_READY              bit_is_set(UCSR0A,5)
#define TXD0_DATA               (UDR0)
#define RXD0_READY              bit_is_set(UCSR0A,7)
#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 UART1.
RxD81() return Read data.
*/
byte RxD81(void)
{
    while(!RXD1_READY);
    return(RXD1_DATA);
}

/*
SIGNAL() UART0 Rx Interrupt - write data to buffer
*/
SIGNAL (SIG_UART0_RECV)
{
    gbpRxInterruptBuffer[(gbRxBufferWritePointer++)] = RXD0_DATA;
}
```

Connector

Company Name : Molex

Pin Number: 3

Model Number

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

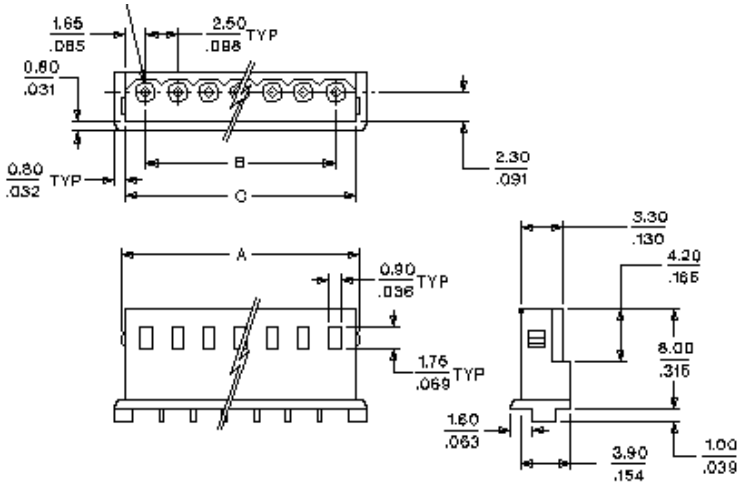
Temperature range : -40°C to +105°C

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

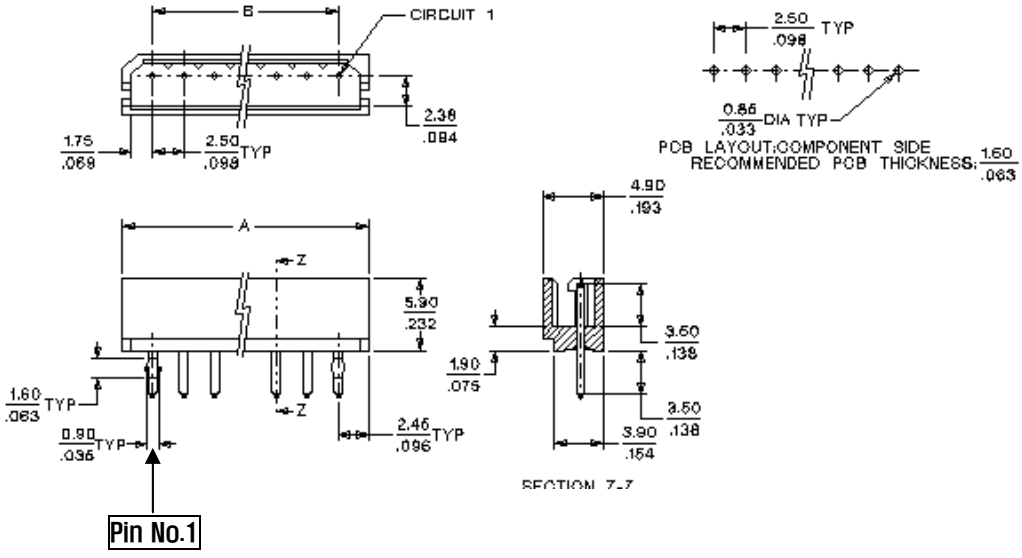
Contact Retention Force-min : 14.7N (3.30 lb)

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

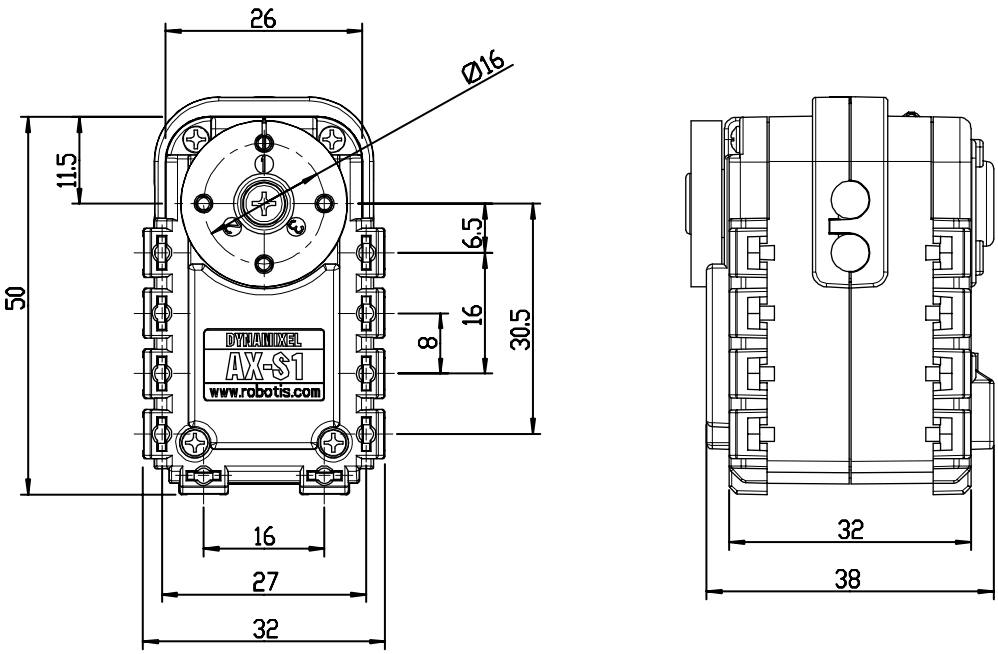
Female Connector



Male Connector



Dimension



CM-5

Dedicated AX-12, AX-S1 control box. Able to control 30 AX-12 actuators, 10 AX-S1.
6 push buttons (5 for selection, 1 for reset)
Optional installable wireless devices available
Battery compartment (AA x 8) with recharging capability (when connected to an external SMPS)