Chapter 1

HTTP API

EiT API

Device status

Resource for storing and fetching the status of a device with a given id.

Get device status

0.1 GET /status/{device}

REQUEST *raw*

RESPONSE

```
200 (OK)
Content-Type: application/json
```

```
    {
        "timestamp": "Timestamp in milliseconds when the server received the last status
update",
        "_id": "Database id, not needed for anything",
        "device_id": "The same as the {device}-part of the request",
        "data1": "3.141529",
        "data_2": "2.71828",
        "and so on...": "any data the device has sent to the server",
        ...
    }
```

Set device status

0.2 POST /status/{device}

REQUEST *raw*

```
Content-Type: application/json
```

```
{
    "data1": "3.141529",
    "data_2": "2.71828",
    "and so on...": "any data here will be stored by the server",
    ...
}
```

Response

```
200 (OK)
Content-Type: application/json
```

```
    Will return the same as a GET request to [/command/{device}]
```

Manage sensor data for a single sensor

Resource for storing and fetching sensor data for a given sensor for a given device.

Get sensor data

0.3 GET /data/{device}/{sensor}

Request *raw*

```

```

```

```

Response

```
200 (OK)
Content-Type: application/json
```

```
    {
        "timestamp": "Timestamp in milliseconds when the server received the last status
update",
        "_id": "Database id, not needed for anything",
        "device_id": "The same as the {device}-part of the request",
        "sensor": "The same as the {sensor}-part of the request",
        "any_key": "data specified by the device when updating the sensor data",
        ...
    }
```

Set sensor data

0.4 POST /data/{device}/{sensor}

Request *raw*

```
Content-Type: application/json
```

```
{
    "any_key": "data specified by the device when updating the sensor data",
    ...
}
```

Response

```
200 (OK)
Content-Type: application/json
```

```
    Will return the same as a GET request to [/command/{device}]
```

Manage sensor data for multiple sensors

Resource for storing and fetching sensor data for all sensors for a given device.

Get the data from all the device's sensors

0.5 GET /data/{device}

Request *raw*

Response

```
200 (OK)
Content-Type: application/json
```

```
    [
        {
            "timestamp": "Timestamp in milliseconds when the server received the last status
update",
            "_id": "Database id, not needed for anything",
            "device_id": "The same as the {device}-part of the request",
            "sensor": "The id of this sensor",
            "any_key": "data specified by the device when updating the sensor data",
            ...
        },
        {
            "timestamp": "Timestamp in milliseconds when the server received the last status
update",
            "_id": "Database id, not needed for anything",
            "device_id": "The same as the {device}-part of the request",
            "sensor": "The id of this sensor",
            "any_key": "data specified by the device when updating the sensor data",
            ...
        },
        ...
    ]
```

Set the data for several of the device's sensors

0.6 POST /data/{device}

Request *raw*

```
Content-Type: application/json
```

```
[
    {
        "sensor": "The id of this sensor",
        ...
    },
    {
        "sensor": "The id of this sensor",
        ...
    },
    ...
]
```

Response

```
200 (OK)
Content-Type: application/json
```

```
    Will return the same as a GET request to [/command/{device}]
```

Manage a device's command queue

Resource for adding commands to a device's command queue and retrieving the command queue.

Get the device's command queue and flush it

0.7 GET /command/{device}

Request *raw*

Response

```
200 (OK)
Content-Type: application/json
```

```
    [
        {
            "timestamp": "Timestamp in milliseconds when the server received the last status
update",
            "any_key": "Any data can go here",
            ...
        },
        ...
    ]
```

Add a command to the device's command queue

0.8 POST /command/{device}

Request *raw*

```
Content-Type: application/json
```

```
{
    "any_key": "Any data can go here",
    ...
}
```

Response

```
200 (OK)
Content-Type: application/json
```

```
    {}
```

Chapter 2

Agent code

2.1 car.h

```cpp
#ifndef CAR_H_
#define CAR_H_

#include "motor.h"
#include <pthread.h>

#define NO_TURN    0
#define LEFT_TURN 1
#define RIGHT_TURN   2

#define TURN_MAGNITUDE   0.5f

class Car{

public:
  Car(int FR,int FL,int BR,int BL) : frontRightWheel(FR, WHEELMODE), frontLeftWheel(FL, WHEELMODE),
  backRightWheel(BR, WHEELMODE), backLeftWheel(BL, WHEELMODE){turn = NO_TURN; speed = 0; direction = 0; mode = IDLE_MODE;};
  void setSpeed(int, bool);
  int getSpeed();
  void turnCar(int);
  void setMode(int);
  int getMode();
  void startPing();
private:
  int direction;
  int speed;
  int turn;
  int mode;
  Motor frontRightWheel;
  Motor frontLeftWheel;
  Motor backRightWheel;
  Motor backLeftWheel;
  pthread_t thread_car;
  static void * staticEntryPoint(void * c);
  void ping();
};

#endif
```

include/car.h

2.2 car.cpp

```cpp
#include "car.h"
#include <stdio.h>
#include <unistd.h>

pthread_mutex_t mutex_car = PTHREAD_MUTEX_INITIALIZER;
```

```

void Car::setSpeed(int theSpeed, bool dir){

  if(getMode() == FAILSAFE_MODE)
    return;

  try{
    switch(turn)
    {
    case NO_TURN:
      //set all wheels same speed
      frontLeftWheel.setSpeed(theSpeed, !dir);
      backLeftWheel.setSpeed(theSpeed, !dir);
      frontRightWheel.setSpeed(theSpeed, dir);
      backRightWheel.setSpeed(theSpeed, dir);
      break;
    case LEFT_TURN:
      //set left wheels TURN_MAGNITUDE of right wheels
      frontLeftWheel.setSpeed(theSpeed*TURN_MAGNITUDE, !dir);
      backLeftWheel.setSpeed(theSpeed*TURN_MAGNITUDE, !dir);
      frontRightWheel.setSpeed(theSpeed, dir);
      backRightWheel.setSpeed(theSpeed, dir);
      break;
    case RIGHT_TURN:
      //set right wheels TURN_MAGNITUDE of left wheels
      frontLeftWheel.setSpeed(theSpeed, !dir);
      backLeftWheel.setSpeed(theSpeed, !dir);
      frontRightWheel.setSpeed(theSpeed*TURN_MAGNITUDE, dir);
      backRightWheel.setSpeed(theSpeed*TURN_MAGNITUDE, dir);
      break;
    }
    speed = theSpeed;
    direction = dir;
  }
  catch(MotorException e) {
    printf("ID: %d lost\n",e.ID);
    printError(e.status);
    setMode(FAILSAFE_MODE);
    printf("Wheels lost!\n");
    startPing();
  }

}

void Car::turnCar(int theTurn){

  if(getMode() == FAILSAFE_MODE)
    return;

  try{
    turn = theTurn;
    if(speed != 0){
      setSpeed(speed, direction);
      return;
    }
    if(turn == NO_TURN){
      setSpeed(0,1);
      return;
    }
    bool dir;
    if(turn == LEFT_TURN)
      dir = 1;
    if(turn == RIGHT_TURN)
      dir = 0;

    printf("direction %d\n",direction);
    frontLeftWheel.setSpeed(1023*TURN_MAGNITUDE, dir);
    backLeftWheel.setSpeed(1023*TURN_MAGNITUDE, dir);
    frontRightWheel.setSpeed(1023*TURN_MAGNITUDE, dir);
    backRightWheel.setSpeed(1023*TURN_MAGNITUDE, dir);
  }
  catch(MotorException e) {
    printf("ID: %d lost\n",e.ID);
    printError(e.status);
    setMode(FAILSAFE_MODE);
    printf("Wheels lost!\n");
    startPing();
  }

}

void Car::setMode(int theMode){
  pthread_mutex_lock( &mutex_car );
  mode = theMode;
  pthread_mutex_unlock( &mutex_car );
}

int Car::getMode(){
  pthread_mutex_lock( &mutex_car );
  int temp = mode;
  pthread_mutex_unlock( &mutex_car );
  return temp;
}

void Car::ping(){
  printf("Ping Car\n");
  while(1){
    int count = 0;
    count += frontLeftWheel.ping();
    count += backLeftWheel.ping();
    count += frontRightWheel.ping();
    count += backRightWheel.ping();

    if(count == 4){
      printf("All wheels active!\n");
      setMode(IDLE_MODE);
      return;
    }
  }
}

void Car::startPing(){
  pthread_create(&thread_car, NULL, Car::staticEntryPoint, this);
}

void * Car::staticEntryPoint(void * c)
{
    ((Car *) c)->ping();
    return NULL;
}
```

src/car.cpp

2.3 motor.h

```
#ifndef MOTOR_H_
#define MOTOR_H_

#include <dynamixel.h>
#include <pthread.h>

// Control table address
#define CW_ANGLE_LIMIT_L  6
#define CW_ANGLE_LIMIT_H  7
#define CCW_ANGLE_LIMIT_L 8
#define CCW_ANGLE_LIMIT_H 9
#define MAX_TORQUE_L     14
#define MAX_TORQUE_H     15
#define HIGH_LIMIT_VOLTAGE  13
#define GOAL_POSITION_L    30
#define GOAL_POSITION_H    31
#define MOVING_SPEED_L     32
#define MOVING_SPEED_H     33
#define PRESENT_POSITION_L  36
#define PRESENT_POSITION_H  37
#define PRESENT_SPEED_L    38
#define PRESENT_SPEED_H    39
#define MOVING        46

#define WHEELMODE    0
#define SERVOMODE    1

#define CW       1
#define CCW      0

#define IDLE_MODE    0
#define FAILSAFE_MODE    1


class MotorException{
public:
  MotorException(int theID, int theStatus) : ID(theID), status(theStatus){};
  int ID;
  int status;
};

class Motor{
public:
  Motor(int, int);
  int getMode();
  int getPosition();
  int getSpeed();
  void setGoalPosition(int);
  void setSpeed(int,bool);
  void setMode(int);
  void setRotateDirection(int);
  void printErrorCode(void);
  void checkStatus();
  int ping();
private:
  int position;
  int speed;
  int mode;
  int ID;
  int commStatus;
  int rotateDirection;
};

void pingAll();
void printError(int status);

#endif
```

include/motor.h

2.4 motor.cpp

```
#include ”motor.h”
#include ”dynamixel.h”
#include ”stdio.h”
#include ”communication.h”

Motor::Motor(int theID, int theMode){
  ID = theID;
  mode = theMode;
  commStatus = COMM_RXSUCCESS;
  setMode(mode);
}

int Motor::getMode(){
  return mode;
}

int Motor::getPosition(){

  int temp = readWord( ID, PRESENT_POSITION_L );
  commStatus = getResult();
  if(commStatus != COMM_RXSUCCESS)
    throw MotorException(ID,commStatus);
  printErrorCode();
  position = temp;
  return position;
}

int Motor::getSpeed(){

  unsigned short temp = readWord( ID, PRESENT_SPEED_L );
  commStatus = getResult();
  if(commStatus != COMM_RXSUCCESS)
    throw MotorException(ID,commStatus);
  printErrorCode();
  speed = temp & 1023;
  return speed;
}

void Motor::setGoalPosition(int thePosition){

  writeWord( ID, GOAL_POSITION_L, thePosition );
  commStatus = getResult();
  if(commStatus != COMM_RXSUCCESS)
    throw MotorException(ID,commStatus);
  printErrorCode();
}


void Motor::setMode(int theMode){

  switch(theMode)
  {
  case WHEELMODE:
    writeWord( ID, CW_ANGLE_LIMIT_L, 0 );
    writeWord( ID, CCW_ANGLE_LIMIT_L, 0 );
    break;
  case SERVOMODE:
    writeWord( ID, CW_ANGLE_LIMIT_L, 0 );
    writeWord( ID, CCW_ANGLE_LIMIT_L, 1023 );
    break;
  default:
    printf(”unknown mode: %d\n”, theMode);
    return;
  }
  mode = theMode;
}

void Motor::setSpeed(int theSpeed, bool theDirection){

  writeWord( ID, MOVING_SPEED_L, theSpeed | (theDirection<<10) );
  commStatus = getResult();
  if(commStatus != COMM_RXSUCCESS)
    throw MotorException(ID,commStatus);
  printErrorCode();
}

void Motor::setRotateDirection(int direction){

  switch(direction)
  {
  case CW:
    writeWord(ID, MOVING_SPEED_L, 1024);
    break;
  case CCW:
    writeWord(ID, MOVING_SPEED_L, 0);
    break;
  default:
    printf("invalid input: %d\n", direction);
    return;
  }
  commStatus = getResult();
  if(commStatus != COMM_RXSUCCESS)
    throw MotorException(ID,commStatus);
  printErrorCode();

  rotateDirection = direction;
}

// Print error bit of status packet
void Motor::printErrorCode()
{
  if(getRXpacketError(ERRBIT_VOLTAGE) == 1)
    printf("Input voltage error!\n");

  if(getRXpacketError(ERRBIT_ANGLE) == 1)
    printf("Angle limit error!\n");

  if(getRXpacketError(ERRBIT_OVERHEAT) == 1)
    printf("Overheat error!\n");

  if(getRXpacketError(ERRBIT_RANGE) == 1)
    printf("Out of range error!\n");

  if(getRXpacketError(ERRBIT_CHECKSUM) == 1)
    printf("Checksum error!\n");

  if(getRXpacketError(ERRBIT_OVERLOAD) == 1)
    printf("Overload error!\n");

  if(getRXpacketError(ERRBIT_INSTRUCTION) == 1)
    printf("Instruction code error!\n");
}

void Motor::checkStatus(){

  unsigned char temp;
  for(int i = 0; i<50; i++)
  {
    if(i == 10 || i == 45)
      continue;
    temp = readByte( ID, i );
    printf("%d:\t%d\t%d\n", ID, i, temp);
  }
  printf("\n");
}

int Motor::ping(){
  pingID(ID);
  commStatus = getResult();
  if( commStatus == COMM_RXSUCCESS )
  {
    //printf("Motor ID: %d active!\n",ID);
    return 1;
  }
  //printf("Motor ID: %d NOT active!\n",ID);
  return 0;
}

void pingAll(){
  for(int i = 0; i<254; i++){
    dxl_ping(i);
    if( dxl_get_result( ) == COMM_RXSUCCESS )
    {
      printf("ID: %d active!\n",i);
    }
  }
}

void printError(int status){
  switch(status)
  {
  case COMM_TXFAIL:

    printf("COMM_TXFAIL: Failed transmit instruction packet!\n");
    break;

  case COMM_TXERROR:
    printf("COMM_TXERROR: Incorrect instruction packet!\n");
    break;

  case COMM_RXFAIL:
    printf("COMM_RXFAIL: Failed get status packet from device!\n");
    break;

  case COMM_RXWAITING:
    printf("COMM_RXWAITING: Now recieving status packet!\n");
    break;

  case COMM_RXTIMEOUT:
    printf("COMM_RXTIMEOUT: There is no status packet!\n");
    break;

  case COMM_RXCORRUPT:
    printf("COMM_RXCORRUPT: Incorrect status packet!\n");
    break;

  default:
    printf("This is unknown error code!\n");
    break;
  }
}
```

src/motor.cpp

2.5 manipulator.h

```
#ifndef MANIPULATOR_H_
#define MANIPULATOR_H_

#include "motor.h"
#include <pthread.h>

#define PI 3.14159265

#define XSTART    0
#define YSTART    155
#define ZSTART    77

class Manipulator{
public:
  Manipulator(int IDOne ,int IDTwo,int IDThree, int IDGrip_left, int IDGrip_right ) :
  one(IDOne, SERVOMODE), two(IDTwo, SERVOMODE), three(IDThree, SERVOMODE),
  grip_left(IDGrip_left, SERVOMODE), grip_right(IDGrip_right, SERVOMODE) {theta1 = 0; theta2 = 0; theta3 = 0; mode =
       IDLE_MODE;};
  void goToPosition(int, int, int);
  void setAngles(float, float, float);
  void setGripper(bool);
  void drawLine(int, int, int, int, int);
  void drawCircle(int, int, int, int, float, float);
  void setMode(int);
  int getMode();
  void startPing();
private:
  float theta1;
  float theta2;
  float theta3;
  int mode;
  Motor one;
  Motor two;
  Motor three;
  Motor grip_left;
  Motor grip_right;
  pthread_t thread;
  static void * staticEntryPoint(void * c);
  void ping();
};

#endif
```

include/manipulator.h

2.6 manipulator.cpp

```
#include <stdio.h>
#include <unistd.h>
#include <math.h>
#include "manipulator.h"

using namespace std;

#define D2  77  //length of first arm in mm
#define D3  155 //length of second arm in mm

#define ANGLE_TO_VALUE  (float)511*6/(5*PI)

#define GRIPPER_LEFT_ZERO 511-140
#define GRIPPER_RIGHT_ZERO  511+140
#define MAX_COUNT   5

pthread_mutex_t mutex_man = PTHREAD_MUTEX_INITIALIZER;

void Manipulator::goToPosition(int x, int y, int z){

  //return error if beyond max
//  if((x*x+y*y+z*z) > (D2+D3)*(D2+D3))
//  {
//    printf("invalid position!\n");
//    return;
//  }

  if(getMode() == FAILSAFE_MODE)
    return;

  float s3, c3, l;

  l = sqrt(x*x+y*y);
  c3 = (z*z + l*l - D2*D2 - D3*D3)/(2*D2*D3);
  s3 = sqrt(1-c3*c3);

  theta3 = atan2(s3,c3);
  theta2 = PI/2 - atan2(D3*s3, D2+D3*c3)-atan2(z,l);
  theta1 = atan2(x,y);

  setAngles(theta1, theta2, theta3);
}

void Manipulator::setAngles(float t1, float t2, float t3){

  if(getMode() == FAILSAFE_MODE)
    return;

  try{
    int dummy;

    if(t1 != t1)
      printf("nan theta 1\n");
    else if(t1 > 5*PI/6){
      one.setGoalPosition(1023);
      printf("Theta 1 too high\n");
    }
    else if(t1 < -5*PI/6){
      one.setGoalPosition(0);
      printf("Theta 1 too low\n");
    }
    else{
      dummy = (float)(t1*ANGLE_TO_VALUE+511);
      one.setGoalPosition(dummy);
      //printf("one: %d\n",dummy);
    }

    if(t2 != t2)
      printf("nan theta 2\n");
    else if(t2 > 5*PI/6){
      two.setGoalPosition(1023);
      printf("Theta 2 too high\n");
    }
    else if(t2 < 0){
      two.setGoalPosition(511);
      printf("Theta 2 too low\n");
    }
    else{
      dummy = (float)(t2*ANGLE_TO_VALUE+511);
      two.setGoalPosition(dummy);
      //printf("two: %d\n",dummy);
    }

    if(t3 != t3)
      printf("nan theta 3\n");
    else if(t3 > 0.78*PI){
      three.setGoalPosition(989);
      printf("Theta 3 too high\n");
    }
    else if(t3 < -0.5*PI){
      three.setGoalPosition(51);
      printf("Theta 3 too low\n");
    }
    else{
      dummy = (float)(t3*ANGLE_TO_VALUE+511);
      three.setGoalPosition(dummy);
      //printf("three: %d\n",dummy);
    }
  }
  catch(MotorException e) {
    printf("ID: %d lost\n",e.ID);
    printError(e.status);
    setMode(FAILSAFE_MODE);
    printf("Manipulator lost!\n");
    startPing();
  }
}

void Manipulator::setGripper(bool on){

  if(getMode() == FAILSAFE_MODE)
    return;

  try{
    if(!on){
      grip_left.setGoalPosition(511-50);
      grip_right.setGoalPosition(511+50);
      return;
    }

    int positionL, positionR, lastPositionL, lastPositionR;
    int counter = 0;
    //put servo set point to zero degrees
    grip_left.setGoalPosition(GRIPPER_LEFT_ZERO);
    grip_right.setGoalPosition(GRIPPER_RIGHT_ZERO);
    lastPositionR = grip_right.getPosition();
    lastPositionL = grip_left.getPosition();
    while(1){
      positionL = grip_left.getPosition();
      positionR = grip_right.getPosition();
      printf("left: %d\tright: %d\n",positionL,positionR);

      if(lastPositionL == positionL || lastPositionR == positionR)
        counter++;
      else
        counter = 0;
      if(counter == MAX_COUNT)
        return;
      lastPositionL = positionL;
      lastPositionR = positionR;
      usleep(10000);
    }
  }
  catch(MotorException e) {
    printf("ID: %d lost\n",e.ID);
    printError(e.status);
    setMode(FAILSAFE_MODE);
    printf("Manipulator lost!\n");
    startPing();
  }
}

void Manipulator::drawLine(int xstart, int ystart, int xend, int yend, int z){

  if(getMode() == FAILSAFE_MODE)
    return;

  try{
    goToPosition(xstart,ystart,z+50);
    sleep(1);
    goToPosition(xstart,ystart,z);
    usleep(100000);
    int x = xend-xstart;
    int y = yend-ystart;
    int length = sqrt(x*x+y*y);
    x /= length;  //normalize
    y /= length;  //normalize
    for(int i = 0; i<length; i++){
      printf("x: %d\ty: %d\n",xstart+i*x, ystart+i*y);
      goToPosition(xstart+i*x, ystart+i*y, z);
      usleep(10000);
    }
  }
  catch(MotorException e) {
    printf("ID: %d lost\n",e.ID);
    printError(e.status);
    setMode(FAILSAFE_MODE);
    printf("Manipulator lost!\n");
    startPing();
  }
}

void Manipulator::drawCircle(int xcenter, int ycenter, int z, int radius, float startAngle, float endAngle){

  if(getMode() == FAILSAFE_MODE)
    return;

  try{
    float t = startAngle;
    float stepSize = 0.01;
    while(t <= endAngle){
      goToPosition(radius*sin(t) + xcenter, radius*cos(t) + ycenter, z);
      t += stepSize;
      usleep(10000);
    }
  }
  catch(MotorException e) {
    printf("ID: %d lost\n",e.ID);
    printError(e.status);
    setMode(FAILSAFE_MODE);
    printf("Manipulator lost!\n");
    startPing();

  }
}

void Manipulator::setMode(int theMode){
  pthread_mutex_lock( &mutex_man );
  mode = theMode;
  pthread_mutex_unlock( &mutex_man );
}
int Manipulator::getMode(){
  pthread_mutex_lock( &mutex_man );
  int temp = mode;
  pthread_mutex_unlock( &mutex_man );
  return temp;
}

void Manipulator::ping(){
  printf("Ping Manipulators\n");
  while(1){
    int count = 0;
    count += one.ping();
    count += two.ping();
    count += three.ping();
    count += grip_left.ping();
    count += grip_right.ping();

    if(count == 5){
      printf("All manipulator motors active!\n");
      setMode(IDLE_MODE);
      //printf("Returning to start position\n");
      //goToPosition(XSTART,YSTART,ZSTART);
      //setGripper(0);
      return;
    }
  }
}

void Manipulator::startPing(){

  pthread_create(&thread, NULL, Manipulator::staticEntryPoint, this);
}

void * Manipulator::staticEntryPoint(void * c)
{
    ((Manipulator *) c)->ping();
    return NULL;
}
```

src/manipulator.cpp

2.7 sensor.h

```
#ifndef SENSOR_H_
#define SENSOR_H_

#include <dynamixel.h>

//control table adress
#define IR_LEFT_FIRE_DATA 26
#define IR_CENTER_FIRE_DATA 27
#define IR_RIGHT_FIRE_DATA  28
#define LIGHT_LEFT_DATA   29
#define LIGHT_CENTER_DATA 30
#define LIGHT_RIGHT_DATA  31
#define IR_OBSTACLE_DETECTED  32
#define LIGHT_DETECTED    33
#define SOUND_DATA    35
#define BUZZER_DATA_NOTE  40
#define BUZZER_DATA_TIME  41

#define LEFT      0
#define CENTER      1
#define RIGHT     2

/*melody:
  0: Rising
  1: Falling
  2: Fight
  4: Fail
  5: sad
  6: bip bip
  7: sad 2
  10: whistle rise
  11: bip bop
  15: bip bip 2
  16: phone
  21: whistle
  24: rtrtrrtrt
*/

class Sensor{
public:
  Sensor(int);
  int getIR(int);
  int getLight(int);  //only infrared light
  void playMelody(int); //input range 0-26
  void playMelody(unsigned char*, int); //play from arrays in songs.h
  void ping();
  void setMode(int);
  int getMode();
private:
  int ID;
  int commStatus;
  int mode;
};

#endif
```

include/sensor.h

2.8 sensor.cpp

```
#include "motor.h"
#include "sensor.h"
#include "stdio.h"
#include <unistd.h>
#include "communication.h"

Sensor::Sensor(int theID){
  ID = theID;
  commStatus = COMM_RXSUCCESS;
  mode = IDLE_MODE;
}

int Sensor::getLight(int pos){

  int data = readByte( ID, LIGHT_LEFT_DATA + pos );
  commStatus = getResult();
  if(commStatus != COMM_RXSUCCESS)
  {
    mode = FAILSAFE_MODE;
     printf("sensor lost\n");
  }
  return data;
}

```

```
int Sensor::getIR(int pos){

  int data = readByte( ID, IR_LEFT_FIRE_DATA + pos );
  commStatus = getResult();
  if(commStatus != COMM_RXSUCCESS)
  {
    mode = FAILSAFE_MODE;
    printf("sensor lost\n");
  }


  return data;
}

void Sensor::playMelody(int song){

  if(song < 0 || song > 26){
    printf("invalid input\n");
    return;
  }
  writeByte(ID, BUZZER_DATA_TIME, 255);
  commStatus = getResult();
  if(commStatus != COMM_RXSUCCESS)
  {
    mode = FAILSAFE_MODE;
    printf("sensor lost\n");
  }
  writeByte(ID, BUZZER_DATA_NOTE, song);
  commStatus = getResult();
  if(commStatus != COMM_RXSUCCESS)
  {
    mode = FAILSAFE_MODE;
    printf("sensor lost\n");
  }
}

void Sensor::playMelody(unsigned char* song, int length){



  for(int i = 0; i<length; i+=2)
  {

    if(song[i+1] != 100)
    {
      writeByte(ID, BUZZER_DATA_TIME, 254);
      writeByte(ID, BUZZER_DATA_NOTE, song[i+1]);
      usleep(40000*song[i]);
    }
    else
    {
      writeByte(ID, BUZZER_DATA_TIME, 0);
      usleep(40000*song[i]);
    }


  }
  writeByte(ID, BUZZER_DATA_TIME, 0);

}

void Sensor::ping(){
  pingID(ID);
  commStatus = getResult();
  if( commStatus == COMM_RXSUCCESS )
  {
    printf("Sensor ID: %d active!\n",ID);
    setMode(IDLE_MODE);
  }
  else{
    setMode(FAILSAFE_MODE);
  }
}

void Sensor::setMode(int theMode){
  mode = theMode;
}

int Sensor::getMode(){
  return mode;
}
```

src/sensor.cpp

2.9 interface.h

```
#ifndef INTERFACE_H_
#define INTERFACE_H_

#include "manipulator.h"
#include "car.h"

void windowInit();
void checkEvent(Manipulator *, Car *);

#endif
```

include/interface.h

2.10 interface.cpp

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <stdio.h>
#include <stdlib.h>
#include "interface.h"
#include "manipulator.h"

#define KEYMASK ButtonPressMask | KeyPressMask | KeyReleaseMask | ButtonReleaseMask | PointerMotionMask

#define FORWARD 25
#define BACKWARD 39
#define LEFT 38
#define RIGHT 40

#define LEFT_MOUSE_BUTTON    1
#define RIGHT_MOUSE_BUTTON   3
#define MOUSE_WHEEL    2
#define MOUSE_WHEEL_FORWARD 4
#define MOUSE_WHEEL_BACKWARD  5

Display *display;
Window window;
XEvent event;
bool button = 0;
bool buttonR = 0;
int xpos = XSTART;
int ypos = YSTART;
int zpos = ZSTART;
int xzero = 0;
int yzero = 0;

void windowInit()
{
    int s;
    /* open connection with the server */
    display = XOpenDisplay(NULL);
    if (display == NULL)
    {
        fprintf(stderr, "Cannot open display\n");
        exit(1);
    }

    s = DefaultScreen(display);

    /* create window */
    window = XCreateSimpleWindow(display, RootWindow(display, s), 10, 10, 500, 500, 1,
                                 BlackPixel(display, s), WhitePixel(display, s));

    /* select kind of events we are interested in */
    XSelectInput(display, window, KEYMASK);

    /* map (show) the window */
    XMapWindow(display, window);

    //do not detect autorepeating events from keyboard
    XAutoRepeatOff(display);
    printf("Display open\n");
}
void checkEvent(Manipulator *man, Car *car){
        XNextEvent (display, &event);
        switch(event.type){
    case MotionNotify:
      if(button){
        xpos -= event.xmotion.x - xzero;
        ypos -= event.xmotion.y - yzero;
        xzero = event.xmotion.x;
        yzero = event.xmotion.y;
        //printf("xpos: %d\t ypos: %d\n", xpos, ypos);
        man->goToPosition(xpos,ypos,zpos);
      }
      break;
    case ButtonPress:
      if(event.xkey.keycode == LEFT_MOUSE_BUTTON)
      {
        button = 1;
        xzero = event.xbutton.x;
        yzero = event.xbutton.y;
      }
      if(event.xkey.keycode == RIGHT_MOUSE_BUTTON)
      {
        buttonR ^= 1;
        man->setGripper(buttonR);
      }
      if(event.xkey.keycode == MOUSE_WHEEL_FORWARD)
      {
        zpos-=10;
        man->goToPosition(xpos,ypos,zpos);
      }
      if(event.xkey.keycode == MOUSE_WHEEL_BACKWARD)
      {
        zpos+=10;
        man->goToPosition(xpos,ypos,zpos);
      }

      printf( "KeyPress: %d\n", event.xkey.keycode );
      break;
    case ButtonRelease:
      if(event.xkey.keycode == LEFT_MOUSE_BUTTON)
        button = 0;
      break;
    case KeyPress:
      //printf( "KeyPress: %d\n", e.xkey.keycode );
      switch(event.xkey.keycode){
        case FORWARD:
          printf("forward\n");
          car->setSpeed(1023,1);
          break;
        case BACKWARD:
          car->setSpeed(1023,0);
          printf("backward\n");
          break;
        case RIGHT:
          car->turnCar(RIGHT_TURN);
          printf("right\n");
          break;
        case LEFT:
          car->turnCar(LEFT_TURN);
          printf("left\n");
          break;
        default:
          printf("unknown:%d\n",event.xkey.keycode);
      }
      break;
    case KeyRelease:
      //printf( "KeyRelease: %d\n", e.xkey.keycode );
      switch(event.xkey.keycode){
        case FORWARD:
          car->setSpeed(0,1);
          printf("forward released\n");
          break;
        case BACKWARD:
          car->setSpeed(0,1);
          printf("backward released\n");
          break;
        case RIGHT:
          car->turnCar(NO_TURN);
          printf("right released\n");
          break;
        case LEFT:
          car->turnCar(NO_TURN);
          printf("left released\n");
          break;
        default:
          printf("unknown:%d\n",event.xkey.keycode);
      }
      break;
  }
}
```

src/interface.cpp

2.11 dynamixel.h

```
#ifndef _DYNAMIXEL_HEADER
#define _DYNAMIXEL_HEADER

#ifdef __cplusplus
extern "C" {
#endif


///////////// device control methods ////////////////////////
int dxl_initialize(int deviceIndex, int baudnum );
void dxl_terminate();


///////////// set/get packet methods //////////////////////////
#define MAXNUM_TXPARAM      (150)
#define MAXNUM_RXPARAM      (60)

void dxl_set_txpacket_id(int id);
#define BROADCAST_ID      (254)

void dxl_set_txpacket_instruction(int instruction);
#define INST_PING       (1)
#define INST_READ       (2)
#define INST_WRITE        (3)
#define INST_REG_WRITE      (4)
#define INST_ACTION       (5)
#define INST_RESET        (6)
#define INST_SYNC_WRITE     (131)

void dxl_set_txpacket_parameter(int index, int value);
void dxl_set_txpacket_length(int length);

int dxl_get_rxpacket_error(int errbit);
#define ERRBIT_VOLTAGE      (1)
#define ERRBIT_ANGLE      (2)
#define ERRBIT_OVERHEAT     (4)
#define ERRBIT_RANGE      (8)
#define ERRBIT_CHECKSUM     (16)
#define ERRBIT_OVERLOAD     (32)
#define ERRBIT_INSTRUCTION    (64)

int dxl_get_rxpacket_length(void);
int dxl_get_rxpacket_parameter(int index);


// utility for value
int dxl_makeword(int lowbyte, int highbyte);
int dxl_get_lowbyte(int word);
int dxl_get_highbyte(int word);


////////// packet communication methods ///////////////////////
void dxl_tx_packet(void);
void dxl_rx_packet(void);
void dxl_txrx_packet(void);

int dxl_get_result(void);
#define COMM_TXSUCCESS      (0)
#define COMM_RXSUCCESS      (1)
#define COMM_TXFAIL     (2)
#define COMM_RXFAIL     (3)
#define COMM_TXERROR      (4)
#define COMM_RXWAITING      (5)
#define COMM_RXTIMEOUT      (6)
#define COMM_RXCORRUPT      (7)


//////////// high communication methods ///////////////////////
void dxl_ping(int id);
int dxl_read_byte(int id, int address);
void dxl_write_byte(int id, int address, int value);
int dxl_read_word(int id, int address);
void dxl_write_word(int id, int address, int value);


#ifdef __cplusplus
}
#endif

#endif
```

include/dynamixel.h

2.12 dynamixel.c

```
#include ”dxl_hal.h”
#include ”dynamixel.h”

#define ID                  (2)
#define LENGTH              (3)
#define INSTRUCTION         (4)
#define ERRBIT              (4)
#define PARAMETER           (5)
#define DEFAULT_BAUDNUMBER  (1)

unsigned char gbInstructionPacket[MAXNUM_TXPARAM+10] = {0};
unsigned char gbStatusPacket[MAXNUM_RXPARAM+10] = {0};
unsigned char gbRxPacketLength = 0;
unsigned char gbRxGetLength = 0;
int gbCommStatus = COMM_RXSUCCESS;
int giBusUsing = 0;


int dxl_initialize(int deviceIndex, int baudnum )
{
  float baudrate;
  baudrate = 2000000.0f / (float)(baudnum + 1);

  if( dxl_hal_open(deviceIndex, baudrate) == 0 )
    return 0;

  gbCommStatus = COMM_RXSUCCESS;
  giBusUsing = 0;
  return 1;
}

void dxl_terminate(void)
{
  dxl_hal_close();
}

void dxl_tx_packet(void)
{
  unsigned char i;
  unsigned char TxNumByte, RealTxNumByte;
  unsigned char checksum = 0;

  if( giBusUsing == 1 )
    return;

  giBusUsing = 1;

  if( gbInstructionPacket[LENGTH] > (MAXNUM_TXPARAM+2) )
  {
    gbCommStatus = COMM_TXERROR;
    giBusUsing = 0;
    return;
  }

  if( gbInstructionPacket[INSTRUCTION] != INST_PING
    && gbInstructionPacket[INSTRUCTION] != INST_READ
    && gbInstructionPacket[INSTRUCTION] != INST_WRITE
    && gbInstructionPacket[INSTRUCTION] != INST_REG_WRITE
    && gbInstructionPacket[INSTRUCTION] != INST_ACTION
    && gbInstructionPacket[INSTRUCTION] != INST_RESET
    && gbInstructionPacket[INSTRUCTION] != INST_SYNC_WRITE )
  {
    gbCommStatus = COMM_TXERROR;
    giBusUsing = 0;
    return;
  }

  gbInstructionPacket[0] = 0xff;
  gbInstructionPacket[1] = 0xff;
  for( i=0; i<(gbInstructionPacket[LENGTH]+1); i++ )
    checksum += gbInstructionPacket[i+2];
  gbInstructionPacket[gbInstructionPacket[LENGTH]+3] = ~checksum;

  if( gbCommStatus == COMM_RXTIMEOUT || gbCommStatus == COMM_RXCORRUPT )
    dxl_hal_clear();

  TxNumByte = gbInstructionPacket[LENGTH] + 4;
  RealTxNumByte = dxl_hal_tx( (unsigned char*)gbInstructionPacket, TxNumByte );

  if( TxNumByte != RealTxNumByte )
  {
    gbCommStatus = COMM_TXFAIL;
    giBusUsing = 0;
    return;
  }

  if( gbInstructionPacket[INSTRUCTION] == INST_READ )
    dxl_hal_set_timeout( gbInstructionPacket[PARAMETER+1] + 6 );
  else
    dxl_hal_set_timeout( 6 );

```

```c
  gbCommStatus = COMM_TXSUCCESS;
}

void dxl_rx_packet(void)
{
  unsigned char i, j, nRead;
  unsigned char checksum = 0;

  if( giBusUsing == 0 )
    return;

  if( gbInstructionPacket[ID] == BROADCAST_ID )
  {
    gbCommStatus = COMM_RXSUCCESS;
    giBusUsing = 0;
    return;
  }

  if( gbCommStatus == COMM_TXSUCCESS )
  {
    gbRxGetLength = 0;
    gbRxPacketLength = 6;
  }

  nRead = dxl_hal_rx( (unsigned char*)&gbStatusPacket[gbRxGetLength], gbRxPacketLength - gbRxGetLength );
  gbRxGetLength += nRead;
  if( gbRxGetLength < gbRxPacketLength )
  {
    if( dxl_hal_timeout() == 1 )
    {
      if(gbRxGetLength == 0)
        gbCommStatus = COMM_RXTIMEOUT;
      else
        gbCommStatus = COMM_RXCORRUPT;
      giBusUsing = 0;
      return;
    }
  }

  // Find packet header
  for( i=0; i<(gbRxGetLength-1); i++ )
  {
    if( gbStatusPacket[i] == 0xff && gbStatusPacket[i+1] == 0xff )
    {
      break;
    }
    else if( i == gbRxGetLength-2 && gbStatusPacket[gbRxGetLength-1] == 0xff )
    {
      break;
    }
  }
  if( i > 0 )
  {
    for( j=0; j<(gbRxGetLength-i); j++ )
      gbStatusPacket[j] = gbStatusPacket[j + i];

    gbRxGetLength -= i;
  }

  if( gbRxGetLength < gbRxPacketLength )
  {
    gbCommStatus = COMM_RXWAITING;
    return;
  }

  // Check id pairing
  if( gbInstructionPacket[ID] != gbStatusPacket[ID])
  {
    gbCommStatus = COMM_RXCORRUPT;
    giBusUsing = 0;
    return;
  }

  gbRxPacketLength = gbStatusPacket[LENGTH] + 4;
  if( gbRxGetLength < gbRxPacketLength )
  {
    nRead = dxl_hal_rx( (unsigned char*)&gbStatusPacket[gbRxGetLength], gbRxPacketLength - gbRxGetLength );
    gbRxGetLength += nRead;
    if( gbRxGetLength < gbRxPacketLength )
    {
      gbCommStatus = COMM_RXWAITING;
      return;
    }
  }

  // Check checksum
  for( i=0; i<(gbStatusPacket[LENGTH]+1); i++ )
    checksum += gbStatusPacket[i+2];
  checksum = ~checksum;

  if( gbStatusPacket[gbStatusPacket[LENGTH]+3] != checksum )
  {
    gbCommStatus = COMM_RXCORRUPT;
    giBusUsing = 0;
    return;
  }

  gbCommStatus = COMM_RXSUCCESS;
  giBusUsing = 0;
}

void dxl_txrx_packet(void)
{
  dxl_tx_packet();

  if( gbCommStatus != COMM_TXSUCCESS )
    return;

  do{
    dxl_rx_packet();
  }while( gbCommStatus == COMM_RXWAITING );
}

int dxl_get_result(void)
{
  return gbCommStatus;
}

void dxl_set_txpacket_id( int id )
{
  gbInstructionPacket[ID] = (unsigned char)id;
}

void dxl_set_txpacket_instruction( int instruction )
{
  gbInstructionPacket[INSTRUCTION] = (unsigned char)instruction;
}

void dxl_set_txpacket_parameter( int index, int value )
{
  gbInstructionPacket[PARAMETER+index] = (unsigned char)value;
}

void dxl_set_txpacket_length( int length )
{
  gbInstructionPacket[LENGTH] = (unsigned char)length;
}

int dxl_get_rxpacket_error( int errbit )
{
  if( gbStatusPacket[ERRBIT] & (unsigned char)errbit )
    return 1;

  return 0;
}

int dxl_get_rxpacket_length(void)
{
  return (int)gbStatusPacket[LENGTH];
}

int dxl_get_rxpacket_parameter( int index )
{
  return (int)gbStatusPacket[PARAMETER+index];
}

int dxl_makeword( int lowbyte, int highbyte )
{
  unsigned short word;

  word = highbyte;
  word = word << 8;
  word = word + lowbyte;
  return (int)word;
}

int dxl_get_lowbyte( int word )
{
  unsigned short temp;

  temp = word & 0xff;
  return (int)temp;
}

int dxl_get_highbyte( int word )
{
  unsigned short temp;

  temp = word & 0xff00;
  temp = temp >> 8;
  return (int)temp;
}

void dxl_ping( int id )
```

```
{
  while(giBusUsing);

  gbInstructionPacket[ID] = (unsigned char)id;
  gbInstructionPacket[INSTRUCTION] = INST_PING;
  gbInstructionPacket[LENGTH] = 2;

  dxl_txrx_packet();
}

int dxl_read_byte( int id, int address )
{
  while(giBusUsing);

  gbInstructionPacket[ID] = (unsigned char)id;
  gbInstructionPacket[INSTRUCTION] = INST_READ;
  gbInstructionPacket[PARAMETER] = (unsigned char)address;
  gbInstructionPacket[PARAMETER+1] = 1;
  gbInstructionPacket[LENGTH] = 4;

  dxl_txrx_packet();

  return (int)gbStatusPacket[PARAMETER];
}

void dxl_write_byte( int id, int address, int value )
{
  while(giBusUsing);

  gbInstructionPacket[ID] = (unsigned char)id;
  gbInstructionPacket[INSTRUCTION] = INST_WRITE;
  gbInstructionPacket[PARAMETER] = (unsigned char)address;
  gbInstructionPacket[PARAMETER+1] = (unsigned char)value;
  gbInstructionPacket[LENGTH] = 4;

  dxl_txrx_packet();
}

int dxl_read_word( int id, int address )
{
  while(giBusUsing);

  gbInstructionPacket[ID] = (unsigned char)id;
  gbInstructionPacket[INSTRUCTION] = INST_READ;
  gbInstructionPacket[PARAMETER] = (unsigned char)address;
  gbInstructionPacket[PARAMETER+1] = 2;
  gbInstructionPacket[LENGTH] = 4;

  dxl_txrx_packet();

  return dxl_makeword((int)gbStatusPacket[PARAMETER], (int)gbStatusPacket[PARAMETER+1]);
}

void dxl_write_word( int id, int address, int value )
{
  while(giBusUsing);

  gbInstructionPacket[ID] = (unsigned char)id;
  gbInstructionPacket[INSTRUCTION] = INST_WRITE;
  gbInstructionPacket[PARAMETER] = (unsigned char)address;
  gbInstructionPacket[PARAMETER+1] = (unsigned char)dxl_get_lowbyte(value);
  gbInstructionPacket[PARAMETER+2] = (unsigned char)dxl_get_highbyte(value);
  gbInstructionPacket[LENGTH] = 5;

  dxl_txrx_packet();
}
```

src/dynamixel.c

2.13 dxl_ hal.h

```
#ifndef _DYNAMIXEL_HAL_HEADER
#define _DYNAMIXEL_HAL_HEADER


#ifdef __cplusplus
extern "C" {
#endif


int dxl_hal_open(int deviceIndex, float baudrate);
void dxl_hal_close();
int dxl_hal_set_baud( float baudrate );
void dxl_hal_clear();
int dxl_hal_tx( unsigned char *pPacket, int numPacket );
int dxl_hal_rx( unsigned char *pPacket, int numPacket );
void dxl_hal_set_timeout( int NumRcvByte );
int dxl_hal_timeout();



#ifdef __cplusplus
}
#endif

#endif
```

src/dxl_hal.h

2.14 dxl_ hal.c

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>
#include <linux/serial.h>
#include <sys/ioctl.h>
#include <sys/time.h>

#include "dxl_hal.h"

int gSocket_fd  = -1;
long  glStartTime = 0;
float gfRcvWaitTime = 0.0f;
float gfByteTransTime = 0.0f;

char  gDeviceName[20];

int dxl_hal_open(int deviceIndex, float baudrate)
{
  struct termios newtio;
  struct serial_struct serinfo;
  char dev_name[100] = {0, };

  sprintf(dev_name, "/dev/ttyUSB%d", deviceIndex);

  strcpy(gDeviceName, dev_name);
  memset(&newtio, 0, sizeof(newtio));
  dxl_hal_close();

  if((gSocket_fd = open(gDeviceName, O_RDWR|O_NOCTTY|O_NONBLOCK)) < 0) {
    fprintf(stderr, "device open error: %s\n", dev_name);
    goto DXL_HAL_OPEN_ERROR;
  }

  newtio.c_cflag    = B38400|CS8|CLOCAL|CREAD;
  newtio.c_iflag    = IGNPAR;
  newtio.c_oflag    = 0;
  newtio.c_lflag    = 0;
  newtio.c_cc[VTIME]  = 0;  // time-out     (TIME * 0.1    ) 0 : disable
  newtio.c_cc[VMIN] = 0;  // MIN     read     return

  tcflush(gSocket_fd, TCIFLUSH);
  tcsetattr(gSocket_fd, TCSANOW, &newtio);

  if(gSocket_fd == -1)
    return 0;

  if(ioctl(gSocket_fd, TIOCGSERIAL, &serinfo) < 0) {
    fprintf(stderr, "Cannot get serial info\n");
    return 0;
  }

  serinfo.flags &= ~ASYNC_SPD_MASK;
  serinfo.flags |= ASYNC_SPD_CUST;
  serinfo.custom_divisor = serinfo.baud_base / baudrate;

  if(ioctl(gSocket_fd, TIOCSSERIAL, &serinfo) < 0) {
    fprintf(stderr, "Cannot set serial info\n");
    return 0;
  }

  dxl_hal_close();

  gfByteTransTime = (float)((1000.0f / baudrate) * 12.0f);

  strcpy(gDeviceName, dev_name);
```

```c
  memset(&newtio, 0, sizeof(newtio));
  dxl_hal_close();

  if((gSocket_fd = open(gDeviceName, O_RDWR|O_NOCTTY|O_NONBLOCK)) < 0) {
    fprintf(stderr, "device open error: %s\n", dev_name);
    goto DXL_HAL_OPEN_ERROR;
  }

  newtio.c_cflag    = B38400|CS8|CLOCAL|CREAD;
  newtio.c_iflag    = IGNPAR;
  newtio.c_oflag    = 0;
  newtio.c_lflag    = 0;
  newtio.c_cc[VTIME]  = 0;  // time-out     (TIME * 0.1    ) 0 : disable
  newtio.c_cc[VMIN] = 0;  // MIN     read     return

  tcflush(gSocket_fd, TCIFLUSH);
  tcsetattr(gSocket_fd, TCSANOW, &newtio);

  return 1;

DXL_HAL_OPEN_ERROR:
  dxl_hal_close();
  return 0;
}

void dxl_hal_close()
{
  if(gSocket_fd != -1)
    close(gSocket_fd);
  gSocket_fd = -1;
}

int dxl_hal_set_baud( float baudrate )
{
  struct serial_struct serinfo;

  if(gSocket_fd == -1)
    return 0;

  if(ioctl(gSocket_fd, TIOCGSERIAL, &serinfo) < 0) {
    fprintf(stderr, "Cannot get serial info\n");
    return 0;
  }

  serinfo.flags &= ~ASYNC_SPD_MASK;
  serinfo.flags |= ASYNC_SPD_CUST;
  serinfo.custom_divisor = serinfo.baud_base / baudrate;

  if(ioctl(gSocket_fd, TIOCSSERIAL, &serinfo) < 0) {
    fprintf(stderr, "Cannot set serial info\n");
    return 0;
  }

  //dxl_hal_close();
  //dxl_hal_open(gDeviceName, baudrate);

  gfByteTransTime = (float)((1000.0f / baudrate) * 12.0f);
  return 1;
}

void dxl_hal_clear(void)
{
  tcflush(gSocket_fd, TCIFLUSH);
}

int dxl_hal_tx( unsigned char *pPacket, int numPacket )
{
  return write(gSocket_fd, pPacket, numPacket);
}

int dxl_hal_rx( unsigned char *pPacket, int numPacket )
{
  memset(pPacket, 0, numPacket);
  return read(gSocket_fd, pPacket, numPacket);
}

static inline long myclock()
{
  struct timeval tv;
  gettimeofday (&tv, NULL);
  return (tv.tv_sec * 1000 + tv.tv_usec / 1000);
}

void dxl_hal_set_timeout( int NumRcvByte )
{
  glStartTime = myclock();
  gfRcvWaitTime = (float)(gfByteTransTime*(float)NumRcvByte + 5.0f);
}

int dxl_hal_timeout(void)
{
  long time;

  time = myclock() - glStartTime;

  if(time > gfRcvWaitTime)
    return 1;
  else if(time < 0)
    glStartTime = myclock();

  return 0;
}
```

src/dxl_hal.c

2.15 communication.h

```
#ifndef COMMUNICATION_H_
#define COMMUNICATION_H_

int readWord(int, int);
int readByte(int, int);
int getResult();
int getRXpacketError(int);
void writeWord(int,int,int);
void writeByte(int,int,int);
void pingID(int);

#endif
```

include/communication.h

2.16 communication.cpp

```
#include <dynamixel.h>
#include <pthread.h>

//Mutex is used for multiple access from threads
//Best way would be to make communication atomic
//such that the communication would finnish without
//being interrupted. That way yould could avoid timeout error
pthread_mutex_t mutex_comm = PTHREAD_MUTEX_INITIALIZER;

int readWord(int id, int adress){
  pthread_mutex_lock( &mutex_comm );
  int temp = dxl_read_word(id, adress);
  pthread_mutex_unlock( &mutex_comm );
  return temp;
}

int readByte(int id, int adress){
  pthread_mutex_lock( &mutex_comm );
  int temp = dxl_read_byte(id, adress);
  pthread_mutex_unlock( &mutex_comm );
  return temp;
}

int getResult(){
  pthread_mutex_lock( &mutex_comm );
  int temp = dxl_get_result();
  pthread_mutex_unlock( &mutex_comm );
  return temp;
}

int getRXpacketError(int errbit){
  pthread_mutex_lock( &mutex_comm );
  int temp = dxl_get_rxpacket_error(errbit);
  pthread_mutex_unlock( &mutex_comm );
  return temp;
}

void writeWord(int id, int adress, int value){
  pthread_mutex_lock( &mutex_comm );
  dxl_write_word(id, adress, value);
  pthread_mutex_unlock( &mutex_comm );
}

void writeByte(int id,int adress,int value){
  pthread_mutex_lock( &mutex_comm );
  dxl_write_byte(id, adress, value);
  pthread_mutex_unlock( &mutex_comm );
}

void pingID(int id){
  pthread_mutex_lock( &mutex_comm );
  dxl_ping(id);
  pthread_mutex_unlock( &mutex_comm );
}
```

src/communication.cpp

2.17 json_ processing.h

```

//defines
#define BUFFER_SIZE  (256 * 1024)  /* 256 KB */
#define URL_FORMAT   "https://wodinaz.com/%s"
#define URL_SIZE     256

//includes
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <string>
#include <vector>
#include <map>

using namespace std;

//functions
void json_test_function();
//example code that uses the four basic functions to communicate with the server

void debug_print_map(map<string,double> mymap);
// a debug function used to print maps received from the server

void debug_print_vector(vector<string> myvector);
//debug function used to print vectors


void json_send_data(map<string,double> mymap);
// Uploads the provided map of sensor values to the server

map<string,double> json_get_data(int id);
// Downloads sensor data from the server. The user must choose which agent (id) to receive from

void json_send_command(string cmd,int id);
// Uploads a command to the server.
//The agent with the corresponding id will download this command

vector<string> json_get_commands(int id);
//Download commands from the server.
```

include/json_processing.h

2.18 json_ processing.cpp

```
/*
 * Copyright (c) 2009-2013 Petri Lehtinen <petri@digip.org>
 *
 * Jansson is free software; you can redistribute it and/or modify
 * it under the terms of the MIT license. See LICENSE for details.
 */

#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#include <jansson.h>

#include "http_functions.h"

#define BUFFER_SIZE  (256 * 1024)  /* 256 KB */

```

```
#define URL_FORMAT   "https://wodinaz.com/%s"
#define URL_SIZE     256
int i=0;

//URL's
#define PATH_CONNECT "connect"
#define PATH_DATA "data/"
#define PATH_COMMAND "command/"

//C++ stuff
#include <string>
#include <iostream>
#include <ostream>
#include <sstream>
#include <vector>
#include <map>
using namespace std;

int myID=0;
int testID=0;

//functions



void debug_print_map(map<string,double> mymap){
    for (map<string,double>::iterator it=mymap.begin(); it!=mymap.end(); ++it)
    {
      string key = it->first;
      double value = it->second;
      printf ("sensor %s has value %f\n",key.c_str(),value);
    }
}

void debug_print_vector(vector<string> myvector){
    for (vector<string>::iterator it=myvector.begin(); it!=myvector.end(); ++it)
    {
      string command = *it;
      printf ("command: %s\n",command.c_str());
    }
}

string convertIntToString(int number)
{
    if (number == 0)
        return "0";
    string temp="";
    string returnvalue="";
    while (number>0)
    {
        temp+=number%10+48;
        number/=10;
    }
    for (int i=0;i<temp.length();i++)
        returnvalue+=temp[temp.length()-i-1];
    return returnvalue;
}
int convertStringToInt(string inputString){
    return atoi(inputString.c_str());
}
double convertStringToDouble(string inputString){
    stringstream ss(inputString);
    double result;
    return ss >> result ? result : 0;

}
string convertDoubleToString(double number){
    ostringstream convert;   // stream used for the conversion

    convert << number;      // insert the textual representation of 'Number' in the characters in the stream

    return convert.str(); // set 'Result' to the contents of the stream

}

map<string,double> json_get_data(int id){
    printf("starting get_data\n");
    map<string,double> data_map;
    int root_length=0;
    char *text_response;
    char url[URL_SIZE];
    string id_path=PATH_DATA;

    string id_string = "client_"+convertIntToString(id);
    id_path.append(id_string);
    snprintf(url, URL_SIZE, URL_FORMAT, id_path.c_str());
    printf("url:%s\n",url);

    text_response = http_request(url);
    printf("response:%s\n",text_response);
    json_t *root;
    json_error_t error;
    root = json_loads(text_response, 0, &error);
    free(text_response);

    if(!root)
    {
        fprintf(stderr, "error: on line %d: %s\n", error.line, error.text);
        throw 202;
    }

    if(!json_is_array(root))
    {
        fprintf(stderr, "error: root is not an object\n");
        json_decref(root);
        root_length=1;
    }

    root_length=json_array_size(root);
    printf("root_length:%d\n",root_length );
    //getting the actual data
    json_t *data, *time_stamp, *entry_id, *sensor, *sensor_value, *device_id;
    double timeStamp,entryID,sensorValue, deviceID;
    string sensor_name;
    for (i=0;i<root_length;i++){ //DEBUG i<root_length
        data = json_array_get(root, i);
        if(!json_is_object(data))
        {
            fprintf(stderr, "error: commit data %d is not an object\n", i + 1);
            json_decref(root);
            throw 202;
        }

        time_stamp = json_object_get(data,"timestamp");
        if (!json_is_string(time_stamp)){
            printf("throwing jsonException\n");
            throw 202;
        }
        else {

            timeStamp = convertStringToDouble(json_string_value(time_stamp));
            printf("timeStamp:%f\n",timeStamp );
        }

        entry_id = json_object_get(data,"_id");
        if (!json_is_string(entry_id)){
            printf("throwing jsonException\n");
            throw 202;
        }
        else {
            entryID =convertStringToDouble(json_string_value(entry_id));
        }

        sensor= json_object_get(data,"sensor");
        if (!json_is_string(sensor)){
            printf("throwing jsonException\n");
            throw 202;
        }
        else {
            sensor_name = json_string_value(sensor);
            printf("sensor_name:%s\n",sensor_name.c_str() );
        }

        const char* snsr_name = sensor_name.c_str();
        sensor_value = json_object_get(data,snsr_name);
        if (!json_is_string(sensor_value)){
            printf("throwing jsonException at sensor_value\n");
            throw 202;
        }
        else {
            sensorValue= convertStringToDouble(json_string_value(sensor_value));
            printf("sensor_value:%f\n",sensorValue);
        }

        device_id = json_object_get(data,"device_id");
        if (!json_is_string(device_id)){
            printf("throwing jsonException at device id\n");
            throw 202;
        }
        else {
            deviceID = convertStringToDouble(json_string_value(device_id));
            printf("deviceID:%f\n",deviceID);
        }
        //put stuff in returning map
        data_map[sensor_name]=sensorValue;
    }
    return data_map;
}

void json_send_data(map<string,double> mymap){
    //printf("starting send_data\n");

    char url[URL_SIZE];


    string id_string = convertIntToString(myID);
    string http_path=PATH_DATA;
    http_path.append("client_"+id_string);
    string sensor_name;
    string key;
    double value;
    string value_string;
    string json_string;
    for (map<string,double>::iterator it=mymap.begin(); it!=mymap.end(); ++it)
    {
        key = it->first;
        value = it->second;
        value_string=convertDoubleToString(value);
        sensor_name=key;

        string http_path=PATH_DATA;
        http_path.append("client_"+id_string);
        http_path.append("/");
        http_path.append(sensor_name);
        json_string="{";
        json_string.append("\"");

        json_string.append(sensor_name);
        json_string.append("\"");
        json_string.append(":");
        json_string.append(" ");
        json_string.append("\""+value_string+"\""+"}");
        snprintf(url, URL_SIZE, URL_FORMAT, http_path.c_str());
        //printf("url:%s\n",url);
        //printf("json_string:%s\n",json_string.c_str());

        char *json_cstring = new char[json_string.length() + 1];
        strcpy(json_cstring, json_string.c_str());
        // do stuff

        http_post(url,json_cstring);
        free(json_cstring);
    }
}

void json_send_command(string cmd,int id){
    printf("starting send_commands\n");

    char url[URL_SIZE];
    string command=cmd;
    string http_path=PATH_COMMAND;
    string id_string = convertIntToString(id);
    http_path.append("client_"+id_string);
    string json_string;
    http_path=PATH_COMMAND;
    http_path.append("client_"+id_string);
    json_string="{";
    json_string.append("\"");

    json_string.append("command");
    json_string.append("\"");
    json_string.append(":");
    json_string.append(" ");
    json_string.append("\""+command+"\""+"}");
    snprintf(url, URL_SIZE, URL_FORMAT, http_path.c_str());
    printf("url:%s\n",url);
    printf("json_string:%s\n",json_string.c_str());

    char *json_cstring = new char[json_string.length() + 1];
    strcpy(json_cstring, json_string.c_str());
    // do stuff

    http_post(url,json_cstring);
    free(json_cstring);
}

vector<string> json_get_commands(int id){
    //printf("starting get_commands\n");
    vector<string> commands_vector;
    int root_length=0;
    char *text_response;
    char url[URL_SIZE];
    string id_path=PATH_COMMAND;

    string id_string = "client_"+convertIntToString(id);
    id_path.append(id_string);
    snprintf(url, URL_SIZE, URL_FORMAT, id_path.c_str());
    //printf("url:%s\n",url);

    text_response = http_request(url);
    //printf("response:%s\n",text_response);
    json_t *root;
    json_error_t error;
    root = json_loads(text_response, 0, &error);
    free(text_response);

    if(!root)
    {
        fprintf(stderr, "error: on line %d: %s\n", error.line, error.text);
        throw 202;
    }

    if(!json_is_array(root))
    {
        fprintf(stderr, "error: root is not an array\n");
        json_decref(root);
        root_length=1;
    }

    root_length=json_array_size(root);
    //printf("root_length:%d\n",root_length );
    //getting the actual data
    json_t *data, *time_stamp, *iterator;
    double timeStamp;
    string command="";
    for (i=0;i<root_length;i++){ //DEBUG i<root_length
        data = json_array_get(root, i);
        if(!json_is_object(data))
        {
            fprintf(stderr, "error: commit data %d is not an object\n", i + 1);
            json_decref(root);
            throw 202;
        }

        time_stamp = json_object_get(data,"timestamp");
        if (!json_is_string(time_stamp)){
            printf("throwing jsonException\n");
            throw 202;
        }
        else {

            timeStamp = convertStringToDouble(json_string_value(time_stamp));
            printf("timeStamp:%f\n",timeStamp );
        }
        iterator =json_object_get(data,"command");
        if (!json_is_string(iterator)){
            printf("throwing jsonException\n");
            throw 202;
        }
        else {
            command = json_string_value(iterator);
            //printf("command:%s\n",command.c_str());
        }
        commands_vector.push_back(command);
    }
    return commands_vector;
}


void json_test_function(){
    map<string,double> debug_map;
    debug_map["test1"]=8.9;
    debug_map["test2"]=5678.456;
    printf("Sending data\n");
    json_send_data(debug_map);
    printf("printing data\n");
    debug_print_map(json_get_data(testID));


    string command1="command_one";
    string command2="command two";
    printf("sending commands\n");
    json_send_command(command1,testID);
    json_send_command(command2,testID);
    printf("printing commands\n");
    debug_print_vector(json_get_commands(testID));
}
```

src/json_processing.cpp

2.19 http_ functions.h

```
#ifndef HTTP_FUNCTIONS
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

// make HTTP request to url
char* http_request(char *url);

```

```
//make a HTTP post to url
void http_post(char* url, char* json_string);
#endif
```

include/http_functions.h

2.20 http_ functions.cpp

```

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <curl/curl.h>
using namespace std;

#define BUFFER_SIZE  (256 * 1024)  /* 256 KB */

#define URL_FORMAT   "https://wodinaz.com/%s"
#define URL_SIZE     256

struct write_result
{
    char *data;
    int pos;
};

static size_t write_response(void *ptr, size_t size, size_t nmemb, void *stream)
{
    struct write_result *result = (struct write_result *)stream;

    if(result->pos + size * nmemb >= BUFFER_SIZE - 1)
    {
        fprintf(stderr, "error: too small buffer\n");
        return 0;
    }

    memcpy(result->data + result->pos, ptr, size * nmemb);
    result->pos += size * nmemb;

    return size * nmemb;
}

// make HTTP request to url
char* http_request(char *url)
{
    CURL *curl = NULL;
    CURLcode status;
    struct curl_slist *headers = NULL;
    char *data = NULL;
    long code;

    curl_global_init(CURL_GLOBAL_ALL);
    curl = curl_easy_init();
    if(!curl)
        goto error;

    data = (char*)malloc(BUFFER_SIZE);
    if(!data)
        goto error;

    struct write_result write_result;
    write_result.data=data;
    write_result.pos=0;

    curl_easy_setopt(curl, CURLOPT_URL, url);

    curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);

    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, write_response);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, &write_result);

    status = curl_easy_perform(curl);
    if(status != 0)
    {
        fprintf(stderr, "error: unable to request data from %s:\n", url);
        fprintf(stderr, "%s\n", curl_easy_strerror(status));
        goto error;
    }

    curl_easy_getinfo(curl, CURLINFO_RESPONSE_CODE, &code);
    if(code != 200)
    {
        fprintf(stderr, "error: server responded with code %ld\n", code);
        goto error;
    }

    curl_easy_cleanup(curl);
    curl_slist_free_all(headers);
    curl_global_cleanup();

    /* zero-terminate the result */
    data[write_result.pos] = '\0';

    return data;

error:
    if(data)
        free(data);
    if(curl)
        curl_easy_cleanup(curl);
    if(headers)
        curl_slist_free_all(headers);
    curl_global_cleanup();
    return NULL;
}

//post to server
void http_post(char* url, char* json_string){
     CURL *curl;
     CURLcode res;

      /* In windows, this will init the winsock stuff */
      curl_global_init(CURL_GLOBAL_ALL);
      /* get a curl handle */
      curl = curl_easy_init();
      if(curl) {
        /* First set the URL that is about to receive our POST. This URL can
           just as well be a https:// URL if that is what should receive the
           data. */
        curl_easy_setopt(curl, CURLOPT_URL, url);

        /* Now specify the POST data */
        curl_easy_setopt(curl, CURLOPT_POSTFIELDS, json_string);

        /* Perform the request, res will get the return code */
        res = curl_easy_perform(curl);

        /* Check for errors */
        if(res != CURLE_OK)
          fprintf(stderr, "curl_easy_perform() failed: %s\n",
                  curl_easy_strerror(res));


        //printf("return code:%d\n",res );
        /* always cleanup */
        curl_easy_cleanup(curl);
      }
      curl_global_cleanup();
}
```

src/http_functions.cpp

Chapter 3

Example code

3.1 Car

```
#include <stdio.h>
#include <termio.h>
#include <unistd.h>
#include <dynamixel.h>
#include <time.h>
#include "car.h"

using namespace std;

//put ID of the wheels here
#define FRONT_RIGHT_WHEEL 1
#define BACK_RIGHT_WHEEL  3
#define FRONT_LEFT_WHEEL  0
#define BACK_LEFT_WHEEL   2


int main(){

  int deviceIndex = 0;
  int baudnum = 1;

  printf("--------CAR TEST PROGRAM--------\n");

  ///////// Open USB2Dynamixel ////////////
  if( dxl_initialize(deviceIndex, baudnum) == 0 )
  {
    printf( "Failed to open USB2Dynamixel!\n" );
    printf( "Press Enter key to terminate...\n" );
    getchar();
    return 0;
  }
  else
    printf( "Succeed to open USB2Dynamixel!\n" );


  Car car1(FRONT_RIGHT_WHEEL, FRONT_LEFT_WHEEL, BACK_RIGHT_WHEEL, BACK_LEFT_WHEEL);
  sleep(1);

  car1.setSpeed(1023,1);
  sleep(2);
  car1.setSpeed(1023,0);
  sleep(2);
  car1.setSpeed(0,1);

    while(1)
    {


    }

  // Close device
  car1.setSpeed(0,1);
  dxl_terminate();
  return 0;
}
```

example/Car/src/main.cpp

3.2 Interface

```
#include <stdio.h>
#include <termio.h>
#include <unistd.h>
#include <dynamixel.h>
#include <time.h>
#include "car.h"
#include "manipulator.h"
#include "interface.h"

using namespace std;

//put ID of the wheels here
#define FRONT_RIGHT_WHEEL 1
#define BACK_RIGHT_WHEEL  3
#define FRONT_LEFT_WHEEL  0
#define BACK_LEFT_WHEEL   2

#define MAN_ONE     4   //zero at 511
#define MAN_TWO     7   //zero at 511, not allowed to go under
#define MAN_THREE   5   //zero at 511

#define GRIPPER_LEFT    12
#define GRIPPER_RIGHT   6

int main(){

  int deviceIndex = 0;
  int baudnum = 1;

  printf("-------LOCAL INTERFACE TEST PROGRAM-------\n");

  ///////// Open USB2Dynamixel ////////////
  if( dxl_initialize(deviceIndex, baudnum) == 0 )
  {
    printf( "Failed to open USB2Dynamixel!\n" );
    printf( "Press Enter key to terminate...\n" );
    getchar();
    return 0;
  }
  else
    printf( "Succeed to open USB2Dynamixel!\n" );

  windowInit();
  Car car1(FRONT_RIGHT_WHEEL, FRONT_LEFT_WHEEL, BACK_RIGHT_WHEEL, BACK_LEFT_WHEEL);
  Manipulator manipulator1(MAN_ONE, MAN_TWO, MAN_THREE, GRIPPER_LEFT, GRIPPER_RIGHT);
  sleep(1);

  manipulator1.goToPosition(XSTART,YSTART,ZSTART);
  manipulator1.setGripper(0);

    while(1)
    {

      checkEvent(&manipulator1, &car1);

    }



  // Close device
  car1.setSpeed(0,1);
  dxl_terminate();
  return 0;
}
```

example/Interface/src/main.cpp

3.3 Main

```
#include <stdio.h>
```

```
#include <termio.h>
#include <unistd.h>
#include <dynamixel.h>
#include <pthread.h>
#include <vector>
#include <string>
#include <time.h>
#include "car.h"
#include "manipulator.h"
#include "json_processing.h"
#include "sensor.h"

using namespace std;

//ID of wheels
#define FRONT_RIGHT_WHEEL 1
#define BACK_RIGHT_WHEEL  3
#define FRONT_LEFT_WHEEL  0
#define BACK_LEFT_WHEEL   2

//ID of manipulator arm
#define MAN_ONE      4   //zero at 511
#define MAN_TWO      7   //zero at 511, not allowed to go under
#define MAN_THREE    5   //zero at 511

//ID of gripper
#define GRIPPER_LEFT     12
#define GRIPPER_RIGHT    6

//ID of sensor
#define SENSOR       100

void *sendSensorData(void *ptr);

int main(){

  pthread_t thread1;
  int deviceIndex = 0;
  int baudnum = 1;
  string command;
  vector <string> commands;
  string strCheck = "position";

  printf("-------MAIN PROGRAM-------\n");

  //////// Open USB2Dynamixel ///////////
  if( dxl_initialize(deviceIndex, baudnum) == 0 )
  {
    printf( "Failed to open USB2Dynamixel!\n" );
    printf( "Press Enter key to terminate...\n" );
    getchar();
    return 0;
  }
  else
    printf( "Succeed to open USB2Dynamixel!\n" );

  Car car1(FRONT_RIGHT_WHEEL, FRONT_LEFT_WHEEL, BACK_RIGHT_WHEEL, BACK_LEFT_WHEEL);
  Manipulator manipulator1(MAN_ONE, MAN_TWO, MAN_THREE, GRIPPER_LEFT, GRIPPER_RIGHT);
  Sensor sensor1(SENSOR);
  sleep(1);

  sensor1.playMelody(6);
  manipulator1.goToPosition(XSTART,YSTART,ZSTART);
  manipulator1.setGripper(0);

  //get old commands from server and disregard them
  vector <string> dummy = json_get_commands(0);

  //create thread for sending sensor data
  pthread_create( &thread1, NULL, sendSensorData, &sensor1 );

    while(1)
    {

        //get commands
        while(commands.empty())
        {
          commands = json_get_commands(0);
        }

        //execute commands
        while(!commands.empty())
        {
          command = commands.front();
          commands.erase(commands.begin());
          if(command == "forward")
            car1.setSpeed(1023,1);

          else if(command == "backward")
            car1.setSpeed(1023,0);

          else if(command == "stop")
```

```
            car1.setSpeed(0,1);

          else if(command == "leftTurn")
            car1.turnCar(LEFT_TURN);

          else if(command == "rightTurn")
            car1.turnCar(RIGHT_TURN);

          else if(command == "noTurn")
            car1.turnCar(NO_TURN);

          else if(command == "gripClose")
            manipulator1.setGripper(1);

          else if(command == "gripOpen")
            manipulator1.setGripper(0);

          else if(command.find(strCheck) != string::npos){
            size_t found1 = command.find(" ");
            size_t found2 = command.find(" ", found1+1);
            size_t found3 = command.find(" ", found2+1);
            string nr1 = command.substr(found1+1, found2-found1);
            string nr2 = command.substr(found2+1, found3-found2);
            string nr3 = command.substr(found3+1);

            int x = atoi(nr1.c_str());
            int y = atoi(nr2.c_str());
            int z = atoi(nr3.c_str());
            manipulator1.goToPosition(x, y, z);
          }

          else
            printf("Unknown command\n");

          printf("command: %s\n", command.c_str());
        }


    }



  // Close device
  car1.setSpeed(0,1);
  dxl_terminate();
  return 0;
}

//thread function for continously sending data
void *sendSensorData(void *ptr){

  //initialize sensor here?
  Sensor* p = (Sensor*)ptr;
  int data;
  map <string,double> sensorData;
  while(1){
    //sleep for 100ms
    sleep(1);

    if(p->getMode() == FAILSAFE_MODE)
    {
      p->ping();
      continue;
    }
    //get data and put it in the map
    data = p->getIR(CENTER);
    printf("\nIR center: %d\n",data);
    sensorData["IR center"] = data;

    data = p->getIR(LEFT);
    printf("IR left: %d\n",data);
    sensorData["IR left"] = data;

    data = p->getIR(RIGHT);
    printf("IR right: %d\n",data);
    sensorData["IR right"] = data;
    //send data
    json_send_data(sensorData);
    //clear map
    sensorData.clear();
  }
  return NULL;
}
```

example/Main/src/main.cpp

3.4 Manipulator

```cpp
#include <stdio.h>
#include <termio.h>
#include <unistd.h>
#include <dynamixel.h>
#include <time.h>
#include "manipulator.h"

using namespace std;

#define MAN_ONE      4    //zero at 511
#define MAN_TWO      7    //zero at 511, not allowed to go under
#define MAN_THREE    5    //zero at 511

#define GRIPPER_LEFT     12
#define GRIPPER_RIGHT    6

int main(){

  int deviceIndex = 0;
  int baudnum = 1;

  printf("-------MANIPULATOR TEST PROGRAM-------\n");

  ///////// Open USB2Dynamixel ////////////
  if( dxl_initialize(deviceIndex, baudnum) == 0 )
  {
    printf( "Failed to open USB2Dynamixel!\n" );
    printf( "Press Enter key to terminate...\n" );
    getchar();
    return 0;
  }
  else
    printf( "Succeed to open USB2Dynamixel!\n" );

  Manipulator manipulator1(MAN_ONE, MAN_TWO, MAN_THREE, GRIPPER_LEFT, GRIPPER_RIGHT);
  sleep(1);

  manipulator1.setGripper(0);

  //test drawing
//  manipulator1.setGripper(1);
//  manipulator1.drawLine(50,200,50,150,0);
//  manipulator1.drawLine(50,175,25,175,0);
//  manipulator1.drawLine(25,200,25,150,0);

  while(1)
  {

    for(int i = 0; i < 130; i+=1)
    {
      manipulator1.goToPosition(0,170,i);
      usleep(5000);
    }
    for(int i = 130; i > 0; i-=1)
    {
      manipulator1.goToPosition(0,170,i);
      usleep(5000);
    }

    for(int i = 0; i < 100; i+=1)
    {
      manipulator1.goToPosition(i,170,0);
      usleep(5000);
    }
    for(int i = 100; i > -100; i-=1)
    {
      manipulator1.goToPosition(i,170,0);
      usleep(5000);
    }
    for(int i = -100; i < 0; i+=1)
    {
      manipulator1.goToPosition(i,170,0);
      usleep(5000);
    }

  }



  // Close device
  dxl_terminate();
  return 0;
}
```

example/Manipulator/src/main.cpp

3.5 Motor

```cpp
#include <stdio.h>
#include <termio.h>
#include <unistd.h>
#include <dynamixel.h>
#include "motor.h"

using namespace std;

#define MOTOR_ID    1

int main(){

  bool b = 0;
  int deviceIndex = 0;
  int baudnum = 1;

  printf("-------MOTOR TEST PROGRAM-------\n");

  ///////// Open USB2Dynamixel ////////////
  if( dxl_initialize(deviceIndex, baudnum) == 0 )
  {
    printf( "Failed to open USB2Dynamixel!\n" );
    printf( "Press Enter key to terminate...\n" );
    getchar();
    return 0;
  }
  else
    printf( "Succeed to open USB2Dynamixel!\n" );


  Motor motor1(MOTOR_ID, SERVOMODE);

  while(1)
  {
    try{
      printf( "Press Enter key to continue!(press ESC and Enter to quit)\n" );
      if(getchar() == 0x1b)
        break;

      if(b){
        printf("motor1 to 300 degrees\n");
        motor1.setGoalPosition(1023);
      }

      else{
        printf("motor1 to 30 degrees\n");
        motor1.setGoalPosition(0);
      }


      b ^= 1;   //change b
    }
    catch(MotorException e) {
      printf("ID: %d lost\n",e.ID);
      printError(e.status);
      break;
    }

  }



  // Close device
  dxl_terminate();
  return 0;
}
```

example/Motor/src/main.cpp

3.6 ReadWrite

```cpp
//###########################################################
//##                      R O B O T I S                    ##
//##          ReadWrite Example code for Dynamixel.        ##
//##                                           2009.11.10 ##
//###########################################################
#include <stdio.h>
#include <termio.h>
#include <unistd.h>
#include <dynamixel.h>
```

```c

// Control table address
#define P_GOAL_POSITION_L 30
#define P_GOAL_POSITION_H 31
#define P_PRESENT_POSITION_L  36
#define P_PRESENT_POSITION_H  37
#define P_MOVING    46

// Defulat setting
#define DEFAULT_BAUDNUM   1 // 1Mbps
#define DEFAULT_ID    1

void PrintCommStatus(int CommStatus);
void PrintErrorCode(void);

int main()
{
  int baudnum = 1;
  int GoalPos[2] = {0, 1023};
  //int GoalPos[2] = {0, 4095}; // for Ex series
  int index = 0;
  int deviceIndex = 0;
  int Moving, PresentPos;
  int CommStatus;

  printf( "\n\nRead/Write example for Linux\n\n" );
  ///////// Open USB2Dynamixel ////////////
  if( dxl_initialize(deviceIndex, baudnum) == 0 )
  {
    printf( "Failed to open USB2Dynamixel!\n" );
    printf( "Press Enter key to terminate...\n" );
    getchar();
    return 0;
  }
  else
    printf( "Succeed to open USB2Dynamixel!\n" );

  while(1)
  {
    printf( "Press Enter key to continue!(press ESC and Enter to quit)\n" );
    if(getchar() == 0x1b)
      break;

    // Write goal position
    dxl_write_word( DEFAULT_ID, P_GOAL_POSITION_L, GoalPos[index] );
    do
    {
      // Read present position
      PresentPos = dxl_read_word( DEFAULT_ID, P_PRESENT_POSITION_L );
      CommStatus = dxl_get_result();

      if( CommStatus == COMM_RXSUCCESS )
      {
        printf( "%d   %d\n",GoalPos[index], PresentPos );
        PrintErrorCode();
      }
      else
      {
        PrintCommStatus(CommStatus);
        break;
      }

      // Check moving done
      Moving = dxl_read_byte( DEFAULT_ID, P_MOVING );
      CommStatus = dxl_get_result();
      if( CommStatus == COMM_RXSUCCESS )
      {
        if( Moving == 0 )
        {
          // Change goal position
          if( index == 0 )
            index = 1;
          else
            index = 0;
        }

        PrintErrorCode();
      }
      else
      {
        PrintCommStatus(CommStatus);
        break;
      }
    }while(Moving == 1);
  }

  // Close device
  dxl_terminate();
  printf( "Press Enter key to terminate...\n" );
  getchar();
  return 0;
}
```

```
// Print communication result
void PrintCommStatus(int CommStatus)
{
  switch(CommStatus)
  {
  case COMM_TXFAIL:
    printf("COMM_TXFAIL: Failed transmit instruction packet!\n");
    break;

  case COMM_TXERROR:
    printf("COMM_TXERROR: Incorrect instruction packet!\n");
    break;

  case COMM_RXFAIL:
    printf("COMM_RXFAIL: Failed get status packet from device!\n");
    break;

  case COMM_RXWAITING:
    printf("COMM_RXWAITING: Now recieving status packet!\n");
    break;

  case COMM_RXTIMEOUT:
    printf("COMM_RXTIMEOUT: There is no status packet!\n");
    break;

  case COMM_RXCORRUPT:
    printf("COMM_RXCORRUPT: Incorrect status packet!\n");
    break;

  default:
    printf("This is unknown error code!\n");
    break;
  }
}

// Print error bit of status packet
void PrintErrorCode()
{
  if(dxl_get_rxpacket_error(ERRBIT_VOLTAGE) == 1)
    printf("Input voltage error!\n");

  if(dxl_get_rxpacket_error(ERRBIT_ANGLE) == 1)
    printf("Angle limit error!\n");

  if(dxl_get_rxpacket_error(ERRBIT_OVERHEAT) == 1)
    printf("Overheat error!\n");

  if(dxl_get_rxpacket_error(ERRBIT_RANGE) == 1)
    printf("Out of range error!\n");

  if(dxl_get_rxpacket_error(ERRBIT_CHECKSUM) == 1)
    printf("Checksum error!\n");

  if(dxl_get_rxpacket_error(ERRBIT_OVERLOAD) == 1)
    printf("Overload error!\n");

  if(dxl_get_rxpacket_error(ERRBIT_INSTRUCTION) == 1)
    printf("Instruction code error!\n");
}
```

example/ReadWrite/ReadWrite.c

3.7 Sensor

```
#include <stdio.h>
#include <termio.h>
#include <unistd.h>
#include <dynamixel.h>
#include "sensor.h"
#include "songs.h"


using namespace std;

#define SENSOR    100

int main(){

  int deviceIndex = 0;
  int baudnum = 1;

  printf("-------Sensor TEST PROGRAM-------\n");

  ///////// Open USB2Dynamixel ////////////
  if( dxl_initialize(deviceIndex, baudnum) == 0 )
  {
    printf( "Failed to open USB2Dynamixel!\n" );
    printf( "Press Enter key to terminate...\n" );
    getchar();
    return 0;
  }
  else
    printf( "Succeed to open USB2Dynamixel!\n" );


  Sensor sensor1(SENSOR);
  sensor1.playMelody(FurElise,sizeof(FurElise));
  //sensor1.playMelody(Sirene,sizeof(Sirene));
  //sensor1.playMelody(6);

  while(1)
  {

  }

  // Close device
  dxl_terminate();
  return 0;
}
```

example/Sensor/src/main.cpp

3.8 SyncWrite

```
//##########################################################
//##                      R O B O T I S                   ##
//##         SyncWrite Example code for Dynamixel.        ##
//##                                           2009.11.10 ##
//##########################################################
#include <stdio.h>
#include <unistd.h>
#include <math.h>
#include <termio.h>

#include <dynamixel.h>

#define PI   3.141592f
#define NUM_ACTUATOR     3

// Control table address
#define P_GOAL_POSITION_L 30
#define P_GOAL_POSITION_H 31
#define P_GOAL_SPEED_L    32
#define P_GOAL_SPEED_H    33

// Defulat setting
#define DEFAULT_BAUDNUM   1 // 1Mbps
#define NUM_ACTUATOR     3 // Number of actuator
#define STEP_THETA       (PI / 100.0f) // Large value is more fast
#define CONTROL_PERIOD    (10000) // usec (Large value is more slow)

void PrintCommStatus(int CommStatus);
void PrintErrorCode(void);

int main()
{
  int id[NUM_ACTUATOR];
  int baudnum = 1;
  int deviceIndex = 0;
  float phase[NUM_ACTUATOR];
  float theta = 0;
  int AmpPos = 512;
  //int AmpPos = 2048; // for EX series
  int GoalPos;
  int i;
  int CommStatus;
  printf( "\n\nSyncWrite example for Linux\n\n" );

  // Initialize id and phase
  for( i=0; i<NUM_ACTUATOR; i++ )
  {
    id[i] = i+1;
    phase[i] = 2*PI * (float)i / (float)NUM_ACTUATOR;
  }

  ///////// Open USB2Dynamixel ////////////
  if( dxl_initialize(deviceIndex, baudnum) == 0 )
  {
    printf( "Failed to open USB2Dynamixel!\n" );
    printf( "Press Enter key to terminate...\n" );
    getchar();
    return 0;
  }
  else
    printf( "Succeed to open USB2Dynamixel!\n" );

  // Set goal speed
  dxl_write_word( BROADCAST_ID, P_GOAL_SPEED_L, 0 );
  // Set goal position
  dxl_write_word( BROADCAST_ID, P_GOAL_POSITION_L, AmpPos );

  while(1)
  {
    printf( "Press Enter key to continue!(press ESC and Enter to quit)\n" );
    if(getchar() == 0x1b)
      break;

    theta = 0;
    do
    {
      // Make syncwrite packet
      dxl_set_txpacket_id(BROADCAST_ID);
      dxl_set_txpacket_instruction(INST_SYNC_WRITE);
      dxl_set_txpacket_parameter(0, P_GOAL_POSITION_L);
      dxl_set_txpacket_parameter(1, 2);
      for( i=0; i<NUM_ACTUATOR; i++ )
      {
        dxl_set_txpacket_parameter(2+3*i, id[i]);
        GoalPos = (int)((sin(theta+phase[i]) + 1.0) * (double)AmpPos);
        printf( "%d  ", GoalPos );
        dxl_set_txpacket_parameter(2+3*i+1, dxl_get_lowbyte(GoalPos));
        dxl_set_txpacket_parameter(2+3*i+2, dxl_get_highbyte(GoalPos));
      }
      dxl_set_txpacket_length((2+1)*NUM_ACTUATOR+4);


      printf( "\n" );

      dxl_txrx_packet();
      CommStatus = dxl_get_result();
      if( CommStatus == COMM_RXSUCCESS )
      {
        PrintErrorCode();
      }
      else
      {
        PrintCommStatus(CommStatus);
        break;
      }

      theta += STEP_THETA;
      usleep(CONTROL_PERIOD);

    }while(theta < 2*PI);
  }

  dxl_terminate();
  printf( "Press Enter key to terminate...\n" );
  getchar();

  return 0;
}

// Print communication result
void PrintCommStatus(int CommStatus)
{
  switch(CommStatus)
  {
  case COMM_TXFAIL:
    printf("COMM_TXFAIL: Failed transmit instruction packet!\n");
    break;

  case COMM_TXERROR:
    printf("COMM_TXERROR: Incorrect instruction packet!\n");
    break;

  case COMM_RXFAIL:
    printf("COMM_RXFAIL: Failed get status packet from device!\n");
    break;

  case COMM_RXWAITING:
    printf("COMM_RXWAITING: Now recieving status packet!\n");
    break;

  case COMM_RXTIMEOUT:
    printf("COMM_RXTIMEOUT: There is no status packet!\n");
    break;

  case COMM_RXCORRUPT:
    printf("COMM_RXCORRUPT: Incorrect status packet!\n");
```

```c
    break;

  default:
    printf("This is unknown error code!\n");
    break;
  }
}

// Print error bit of status packet
void PrintErrorCode()
{
  if(dxl_get_rxpacket_error(ERRBIT_VOLTAGE) == 1)
    printf("Input voltage error!\n");

  if(dxl_get_rxpacket_error(ERRBIT_ANGLE) == 1)
    printf("Angle limit error!\n");

  if(dxl_get_rxpacket_error(ERRBIT_OVERHEAT) == 1)
    printf("Overheat error!\n");

  if(dxl_get_rxpacket_error(ERRBIT_RANGE) == 1)
    printf("Out of range error!\n");

  if(dxl_get_rxpacket_error(ERRBIT_CHECKSUM) == 1)
    printf("Checksum error!\n");

  if(dxl_get_rxpacket_error(ERRBIT_OVERLOAD) == 1)
    printf("Overload error!\n");

  if(dxl_get_rxpacket_error(ERRBIT_INSTRUCTION) == 1)
    printf("Instruction code error!\n");
}
```

example/SyncWrite/SyncWrite.c

Chapter 4

Server code

4.1 Installation notes

```
Requirements:
MongoDB
Python
    pip (http://www.pip-installer.org/en/latest/)
    virtualenv (http://www.virtualenv.org/en/latest/)

Setup:
In this directory:
# virtualenv --no-site-packages venv
# source venv/bin/activate
# pip install -r requirements.txt
# python server/server.py
```

server/INSTALL

4.2 Utility functions

```python
from flask import Response
from functools import wraps
from helpers import unicode_to_str

def get_str_object_or_404(action):
    @wraps(action)
    def wrapper(*args, **kwargs):
        result = action(*args, **kwargs)
        if not result:
            return {}, 404, {'Access-Control-Allow-Origin': '*', 'Access-Control-Allow-Headers': 'accept, content-type, origin'}
        else:
            return unicode_to_str(result), 200, {'Access-Control-Allow-Origin': '*', 'Access-Control-Allow-Headers': 'accept, content-type, origin'}
    return wrapper
```

server/tools/decorators.py

```python
import time

def unicode_to_str(data):
    if isinstance(data, dict):
        ret = {}
        for key, value in data.iteritems():
            ret[unicode_to_str(key)] = unicode_to_str(value)
        return ret
    elif isinstance(data, list):
        ret = []
        for value in data:
            ret.append(unicode_to_str(value))
        return ret
    else:
        return str(data)

def get_microtime():
    return int(round(time.time() * 1000))
```

server/tools/helpers.py

4.3 Server logic

```python
from flask import request
from flask.ext import restful
from pymongo import MongoClient
from tools.decorators import get_str_object_or_404
from tools.helpers import get_microtime, unicode_to_str

mongodb = MongoClient().db

class OptionsResrouce(restful.Resource):
    def options(self):
        return {'Allow': 'GET,POST'}, 200, {'Access-Control-Allow-Origin': '*', 'Access-Control-Allow-Methods': 'POST,GET', 'Access-Control-Allow-Headers': 'accept, content-type, origin'}

class Status(restful.Resource):
    def __init__(self):
        self.collection = mongodb.status

    @get_str_object_or_404
    def get(self, id):
        return self.collection.find_one({'device_id': id})

    def post(self, id):
        data = request.get_json(force=True, cache=False)
        data["device_id"] = id
        data["timestamp"] = get_microtime()

        self.collection.update({'device_id': id}, data, upsert=True)

        return {"commands": Command().get(id)}

    def options(self, id):
        return {'Allow': 'GET,POST'}, 200, {'Access-Control-Allow-Origin': '*', 'Access-Control-Allow-Methods': 'POST,GET', 'Access-Control-Allow-Headers': 'accept, content-type, origin'}

class StatusOptions(OptionsResrouce):
    pass

class Data(restful.Resource):
    def __init__(self):
        self.collection = mongodb.data

    @get_str_object_or_404
    def get(self, id, sensor):
        return self.collection.find_one({'device_id': id, 'sensor': sensor})

    def post(self, id, sensor):
        data = request.get_json(force=True, cache=False)

        data["device_id"] = id
        data["timestamp"] = get_microtime()
        data["sensor"] = sensor
        self.collection.update({'device_id': id, 'sensor': sensor}, data, upsert=True)

        return {"commands": Command().get(id)}

    def options(self, id):
        return {'Allow': 'GET,POST'}, 200, {'Access-Control-Allow-Origin': '*', 'Access-Control-Allow-Methods': 'POST,GET', 'Access-Control-Allow-Headers': 'accept, content-type, origin'}

class DataOptions(OptionsResrouce):
    pass

class Data_Collection(restful.Resource):
    def __init__(self):
        self.collection = mongodb.data

    @get_str_object_or_404
    def get(self, id):
        return [sensor for sensor in self.collection.find({'device_id': id})]

    def post(self, id):
        data = request.get_json(force=True, cache=False)

        for sensor_data in data:
            sensor_data["device_id"] = id
            sensor_data["timestamp"] = get_microtime()
            self.collection.update({'device_id': id, 'sensor': sensor_data['sensor']}, sensor_data, upsert=True)

        return {"commands": Command().get(id)}

    def options(self, id):
        return {'Allow': 'GET,POST'}, 200, {'Access-Control-Allow-Origin': '*', 'Access-Control-Allow-Methods': 'POST,GET', 'Access-Control-Allow-Headers': 'accept, content-type, origin'}

class Command(restful.Resource):
    def __init__(self):
        self.collection = mongodb.commands
        self.id = hex(id(self))

    def __get_document_lock(self, id):
        document = self.collection.find_one({"device_id": id})

        if not document:
            self.collection.insert({"device_id": id, "state": self.id, "queue": []})
            document = self.collection.find_one({"device_id": id})

        while document["state"] != self.id:
            while document["state"] != "ready":
                document = self.collection.find_one({"device_id": id})
            self.collection.update({"device_id": id, "state": "ready"}, {"$set": {"state": self.id}})
            document = self.collection.find_one({"device_id": id})

    def __free_document_lock(self, id):
        self.collection.update({"device_id": id}, {"$set": {"state": "ready"}})

    def get(self, id):
        self.__get_document_lock(id)

        try:
            document = self.collection.find_one({"device_id": id})
            self.collection.update({"device_id": id}, {"$set": {"queue": []}})
        finally:
            self.__free_document_lock(id)

        return unicode_to_str(document["queue"])

    def post(self, id):
        command = request.get_json(force=True, cache=False)
        command["timestamp"] = get_microtime()

        self.__get_document_lock(id)

        try:
            self.collection.update({"device_id": id}, {"$push": {"queue": command}})
        finally:
            self.__free_document_lock(id)
        return {}, 200, {'Access-Control-Allow-Origin': '*', 'Access-Control-Allow-Headers': 'accept, content-type, origin'}

    def options(self, id):
        return {'Allow': 'GET,POST'}, 200, {'Access-Control-Allow-Origin': '*', 'Access-Control-Allow-Methods': 'POST,GET', 'Access-Control-Allow-Headers': 'accept, content-type, origin'}

class CommandOptions(OptionsResrouce):
    pass
```

server/resources.py

4.4 Main program

```
from flask import Flask
from flask.ext import restful
import resources

app = Flask(__name__)
api = restful.Api(app)

api.add_resource(resources.Status, '/status/<string:id>')
api.add_resource(resources.StatusOptions, '/status')
api.add_resource(resources.Data, '/data/<string:id>/<string:sensor>')
api.add_resource(resources.DataOptions, '/data')
api.add_resource(resources.Data_Collection, '/data/<string:id>')
api.add_resource(resources.Command, '/command/<string:id>')
api.add_resource(resources.CommandOptions, '/command')

if __name__ == '__main__':
    app.run(debug=True)
```

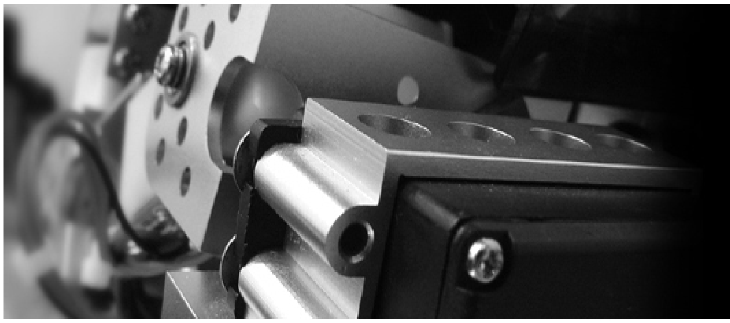
server/server.py

Chapter 5

Datasheets

5.1 AX-12

User's Manual   2006-06-14



Closer to Real, ROBOTIS

Dynamixel AX-12

Contents

1. Dynamixel AX-12

1-1. Overview and Characteristics of AX-12

Dynamixel AX-12 The Dynamixel series robot actuator is a smart, modular actuator that incorporates a gear reducer, a precision DC motor and a control circuitry with networking functionality, all in a single package. Despite its compact size, it can produce high torque and is made with high quality 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. The Dynamixel series robot actuator has 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 handle the problem automatically (e.g. torque off)

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.

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

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

Frames A hinge frame and a side mount frame are included.

1-2. Main Specifications

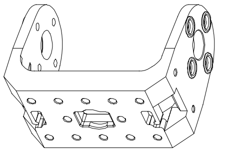
	AX-12	
Weight (g)	**55**	
Gear Reduction Ratio	**1/254**	
Input Voltage (V)	**at 7V**	**at 10V**
Final Max Holding Torque(kgf.cm)	**12**	**16.5**
Sec/60degree	**0.269**	**0.196**

Resolution	0.35°
Operating Angle	300°, Endless Turn
Voltage	7V~10V (Recommended voltage: 9.6V)
Max. Current	900mA
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	Position, Temperature, Load, Input Voltage, etc.
Material	Engineering Plastic

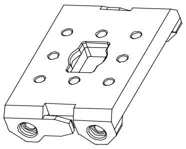
2. Dynamixel Operation

2-1. **Mechanical Assembly**

Frames Provided The two frames provided with AX-12 are shown below.

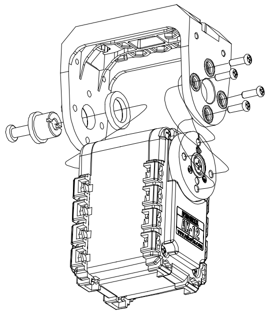


OF-12SH

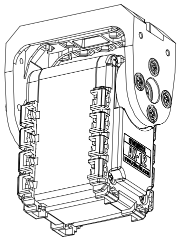


OF-12S

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



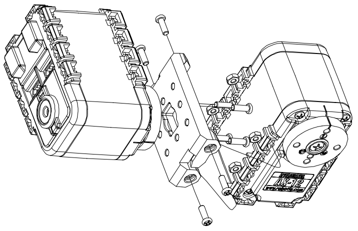
Exploded view



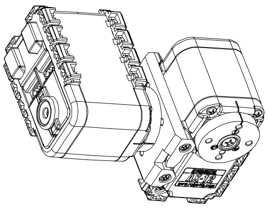
Assembled

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

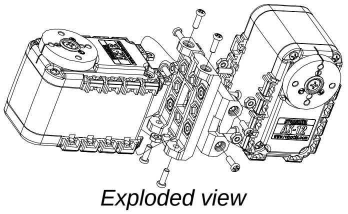


Exploded view

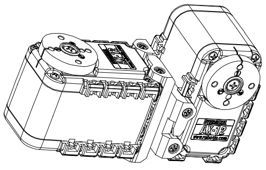


Assembled

Body2Body



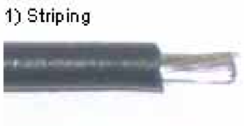
Exploded view

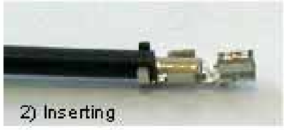


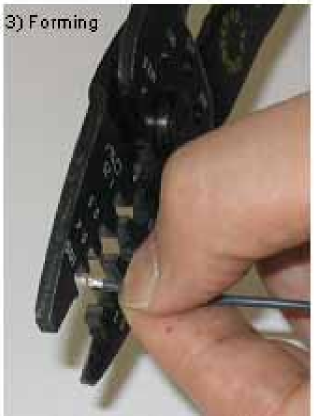
Assembled

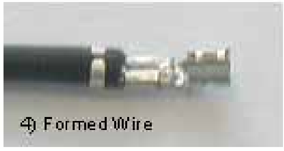
2-2 . Connector Assembly

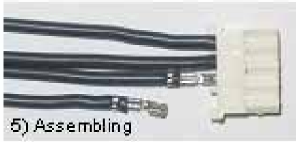
Assemble the connectors as shown below. Attach the wires to the terminals using the correct crimping tool. If you do not have access to a crimping tool, solder the terminals to the wires to ensure that they do not become loose during operation.

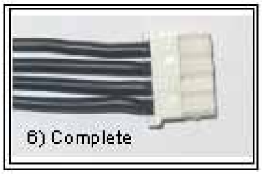








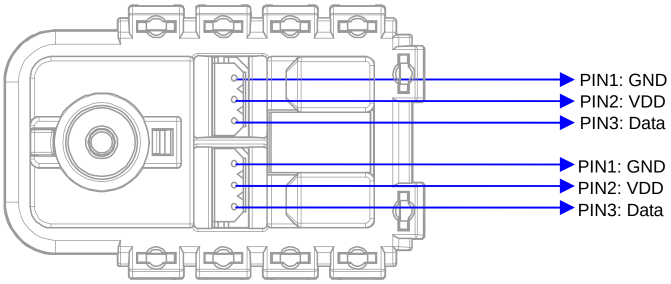




2-3. Dynamixel Wiring

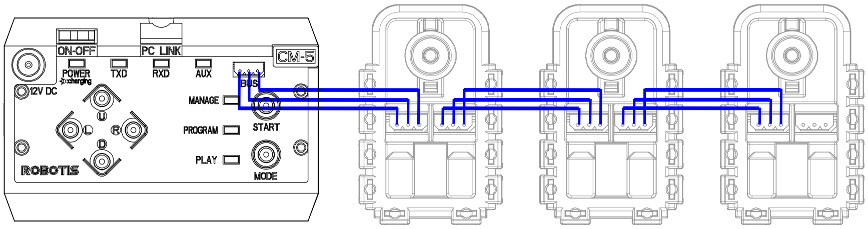
Pin Assignment

The connector pin assignments are as the following. The two connectors on the Dynamixel are connected pin to pin, thus the AX-12 can be operated with only one connector attached.



Wiring

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



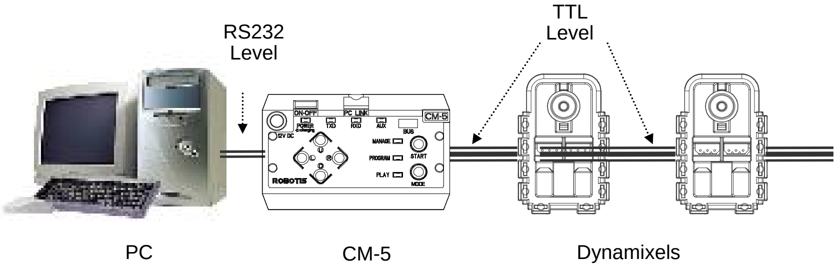
Control Box “CM-5”

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 the CM-5 controller.



bioloid

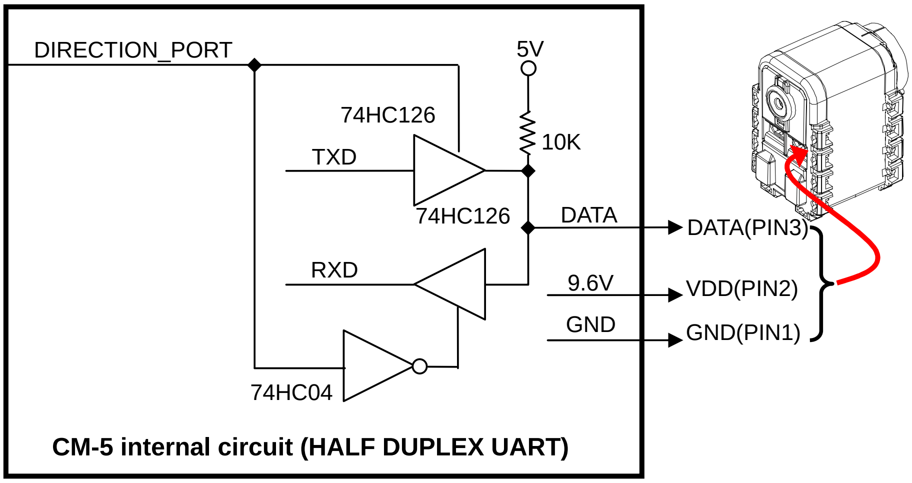
A robot can be built using only the CM-5 controller and a number of AX-12 actuators. An edutainment robotic kit named "Bioloid" is available which is based on the CM-5 controller and the AX-12 actuators.

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.

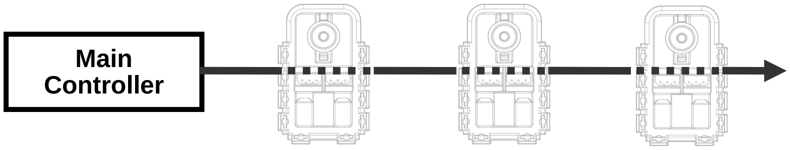


The power is supplied to the Dynamixel actuator from the main controller through Pin 1 and Pin 2 of the Molex3P connector. (The circuit shown above is presented only to explain the use of half duplex UART. The CM-5 controller already has the above circuitry built in, thus the Dynamixel actuators can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

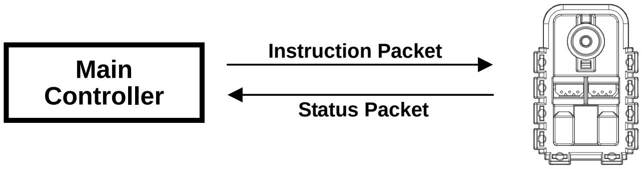
When power is applied to the Dynamixel actuator, the LED blinks twice to confirm its connection.

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

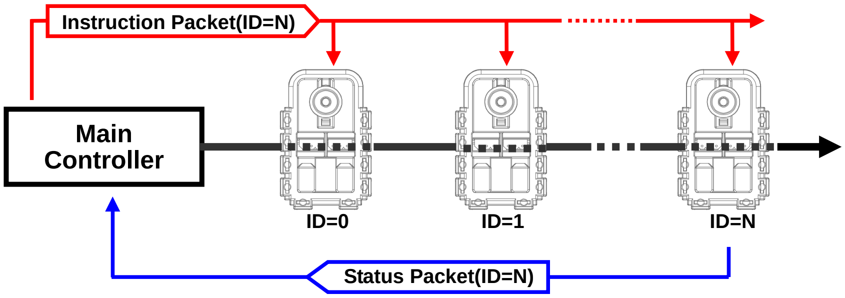
3. Communication Protocol

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. The initial value is set to 1.

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

ERROR The byte representing errors 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	Overload Error	Set to 1 if the specified maximum torque can't control the applied load.
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	12(0x0C)
	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	1(0x01)
	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	?
RAM Area	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)

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-12, this value is 0X000C (12).

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

Adress4	Hex	Set BPS	Goal BPS	Error
1	0X01	1000000.0	1000000.0	0.000%
3	0X03	500000.0	500000.0	0.000%
4	0X04	400000.0	400000.0	0.000%
7	0X07	250000.0	250000.0	0.000%
9	0X09	200000.0	200000.0	0.000%
16	0X10	117647.1	115200.0	-2.124%
34	0X22	57142.9	57600.0	0.794%
103	0X67	19230.8	19200.0	-0.160%
207	0XCF	9615.4	9600.0	-0.160%

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

Caution The initial value of Baudrate is set to 1(1000000bps)

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 0x06,0x07,0x08,0x09

Operating Angle Limit. Sets the Dynamixel actuator's operating angle 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 the Goal Position is set outside this range set by the operating angle limits.

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 0x0E,0x0F, 0x22,0x23

Max Torque. The maximum torque output for the Dynamixel actuator. When this value is set to 0, the Dynamixel actuator enters the Free Run mode. There are two locations where this maximum torque limit is defined; in the EEPROM (Address 0X0E, 0x0F) and in the RAM (Address 0x22, 0x23). When the power is turned on, the maximum torque limit value defined in the EEPROM is copied to the location in the RAM. The torque of the Dynamixel actuator is limited by the values located in the RAM (Address 0x22, 0x23).

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 0X11 **Alarm LED.** If the corresponding Bit is set to 1, the LED blinks when an Error occurs.

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

This function operates following the “OR” logical operation of all bits. For example, if the value is set to 0X05, the LED will blink when an Input 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.** If the corresponding Bit is set to a 1, the Dynamixel actuator’s torque will be turned off when an error occurs.

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

This function operates following the “OR” logical operation of all bits. However, unlike the Alarm LED, after returning to a normal condition, it maintains the torque off status. To recover, the Torque Enable (Address0X18) needs to be reset to 1.

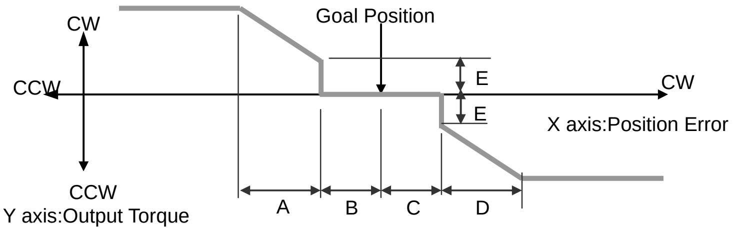
Address 0x14~0x17 **Calibration.** Data used for compensating for the differences between the potentiometers used in the Dynamixel units. The user cannot change this data.

The following (from Address 0x18) is in the RAM area.

Address 0x18 **Torque Enable.** When the power is first turned on, the Dynamixel actuator enters the Torque Free Run condition (zero torque). Setting the value in Address 0x18 to 1 enables the torque.

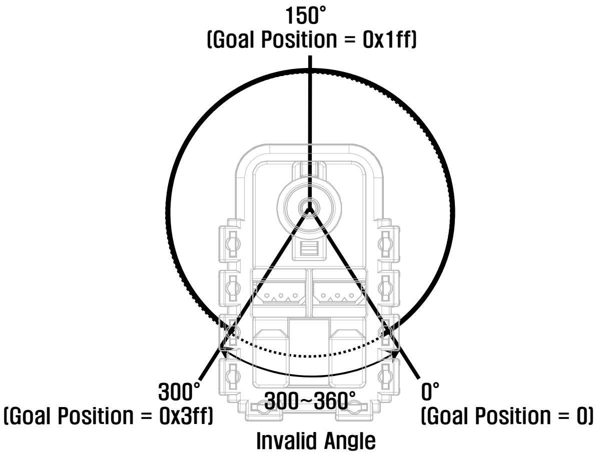
Address 0x19 **LED**. The LED turns on when set to 1 and turns off if set to 0.

Address 0x1A~0x1D **Compliance Margin and Slope.** The compliance of the Dynamixel actuator is defined by setting the compliance Margin and Slope. This feature can be utilized for absorbing shocks at the output shaft. The following graph shows how each compliance value (length of A, B, C & D) is defined by the 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 actuator output to move to. Setting this value to 0x3ff moves the output shaft to the position at 300°.



Address 0x20,0x21 **Moving Speed.** Sets the angular velocity of the output moving to the Goal Position. Setting this value to its maximum value of 0x3ff moves the output with an angular velocity of 114 RPM, provided that there is enough power supplied (The lowest velocity is when this value is set to 1. When set to 0, the velocity is the largest possible for the supplied voltage, e.g. no velocity control is applied.)

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

Address 0x26,0x27 **Present Speed.** Current angular velocity of the Dynamixel actuator output.

Address 0x28,0x29 **Present Load.** The magnitude of the load on the operating Dynamixel actuator. 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 currently applied to the Dynamixel actuator. The value is 10 times the actual voltage. For example, 10V is represented as 100 (0x64).

Address 0x2B **Present Temperature.** The internal temperature of the Dynamixel actuator in Degrees Celsius.

Address 0x2C **Registered Instruction.** Set to 1 when an instruction is assigned by the REG_WRITE command. Set to 0 after it completes the assigned instruction by the Action command.

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

Address 0x2F **Lock.** If set to 1, only Address 0x18 to 0x23 can be written to and other areas cannot. Once locked, it can only be unlocked by turning the power off.

Address 0x30,0x31 **Punch.** The minimum current supplied to the motor during operation. The initial value is set to 0x20 and its maximum value is 0x3ff.

Endless Turn If both values for the CW Angle Limit and the CCW Angle Limit are set to 0, an Endless Turn mode can be implemented by setting the Goal Speed. This feature can be used for implementing a continuously rotating wheel.

Goal Speed Setting

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

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

Range

Each data has a valid minimum and maximum values. Write instructions made outside of these valid ranges will return an error. The following table summarizes the data range for each register. 16 bit data registers are indicated with two bytes (L) and (H). Both bytes need to be written at the same time as one 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)
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 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`

ID LENGTH INSTRUCTION PARAMETERS CHECKSUM

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`

ID LENGTH INSTRUCTION PARAMETERS . CHECKSUM

The returned Status Packet will be as the following.
Status Packet : 0XFF 0XFF 0X01 0X03 0X00 0X20 0XDB

ID LENGTH ERROR PARAMETER1 CHECKSUM

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`

ID LENGTH INSTRUCTION CHECKSUM

The returned Status Packet is as the following

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

ID LENGTH ERROR CHECKSUM

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

Example 4 **Resetting the Dynamixel actuator with an ID of 0**

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

ID LENGTH INSTRUCTION CHECKSUM

The returned Status Packet is as the following

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

ID LENGTH ERROR CHECKSUM

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	Data for 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	Data for 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	
….		

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

For the following examples, we assume a Dynamixel actuator with an ID of 1 in Reset status and that the Baud rate is 57142 BPS.

Example 6 **Reading the Model Number and Firmware Version of the Dynamixel actuator with an ID of 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 74 00 08 7D (LEN:009)

Status Packet Result Model Number = 116 (0x74) (for the case of DX-116) Firmware Version = 0x08

Example 7 **Changing the ID to 0 for a Dynamixel actuator with an ID of 1**

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

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

Status Packet Result NO ERROR

Example 8 **Changing the Baud Rate of a Dynamixel actuator to 1M bps**

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

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

Status Packet Result NO ERROR

Example 9 **Resetting the Return Delay Time to 4 uSec for a Dynamixel actuator with an ID of 0**

A Return Delay Time Value of 1 corresponds to 2uSec.

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

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

Status Packet Result NO ERROR

It is recommended to set the Return Delay Time to the minimum value allowed by the Main Controller.

Example 10 **Limiting the operating angle range to 0°~150° for a Dynamixel actuator with an ID of 0**

Since the CCW Angle Limit of 0x3ff corresponds to 300°, the angle 150° is represented by the value 0x1ff

Instruction Packet Instruction = WRITE_DATA, Address = 0x08, DATA = 0xff, 0x01

Communication ->[Dynamixel]:FF FF 00 05 03 08 FF 01 EF (LEN:009)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR

Example 11 **Resetting the upper limit for the operating temperature to 80°C for a Dynamixel actuator with an ID of 0**

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

Communication ->[Dynamixel]:FF FF 00 04 03 0B 50 9D (LEN:008)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR

Example 12 **Setting the operating voltage to 10V ~ 17V for a Dynamixel actuator with an ID of 0**

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 00 05 03 0C 64 AA DD (LEN:009)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR

Example 13 **Setting the maximum torque to 50% of its maximum possible value for a Dynamixel actuator with an ID of 0**

Set the MAX Torque value located in the ROM area to 0x1ff which is 50% 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

To verify the effect of the adjusted Max Torque value, the power needs to be turned off and then on.

Example 14 **Set the Dynamixel actuator with an ID of 0 to never return 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 is not returned starting with the following instruction.

Example 15 **Set the Alarm to blink the LED and Shutdown (Torque off) the actuator when the operating temperature goes over the set limit**

Since the Overheating Error is Bit 2, 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 and Enable Torque for a Dynamixel actuator with an ID of 0**

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

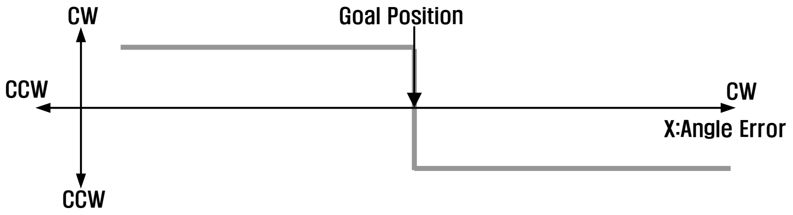
Communication ->[Dynamixel]:FF FF 00 05 03 18 <u>01 01</u> DD (LEN:009)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR

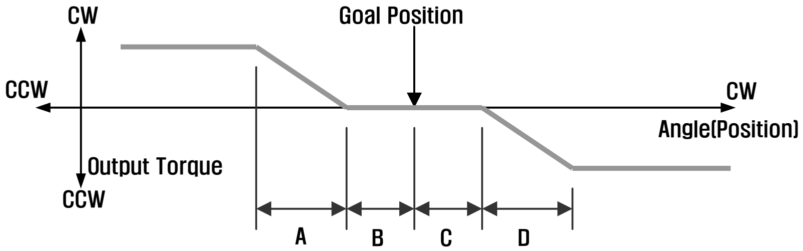
You can verify the Torque Enabled status by trying to move the output of the actuator by hand.

Example 17 **Setting the Compliance Margin to 1 and Compliance Slope to 0x40 for a Dynamixel actuator with an ID of 0**

Compliance The Angle Error and Torque Output can be represented with the following graph.



Even if the position deviates a little from the goal position in the CW direction, a large amount of torque is generated in the CCW direction to compensate for this. However, since inertia must be considered, a realistic implementation differs from this approach. Considering this, the given conditions can be represented by the following graph.



A : CCW Compliance Slope (Address0x1D) = 0x40 (about 18.8°)
B : CCW Compliance Margin (Address0x1B) = 0x01 (about 0.29°)
C : CW Compliance Margin (Address0x01A) = 0x01 (about 0.29°)
D : CW Compliance Slope (Address0x1C) = 0x40 (about 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 Compliance Slope takes effect with discrete steps of 2^n (n is integer). Thus any Compliance value between 0x11 and 0x20 has identical effects.

Example 18 **Position the output of a Dynamixel actuator with an ID of 0 to 180° with an angular velocity of 057RPM**

Set Address 0x1E (Goal Position) to 0x200 and Address 0x20 (Moving Speed) to 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 **Position the output of a Dynamixel actuator with an ID of 0 to 0° and Position the output of a Dynamixel actuator with an ID of 1 to 300°, and initiate the movement at the same time.**

If the WRITE_DATA is used, the movement of the two actuators cannot be initiate at the same time, thus the REG_WRITE and ACTION instructions should be used instead.

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 **Lock all addresses except for Address 0x18 ~ Address0x23 for a Dynamixel actuator with an ID of 0**

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

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)
```

Range Error

Example 21 **Set the minimum power (Punch) to 0x40 for a Dynamixel actuator with an ID of 0**

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

Communication

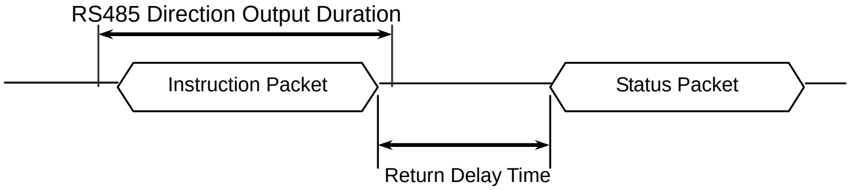
```
->[Dynamixel]:FF FF 00 05 03 30 40 00 87 (LEN:009)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)
```

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
}
```

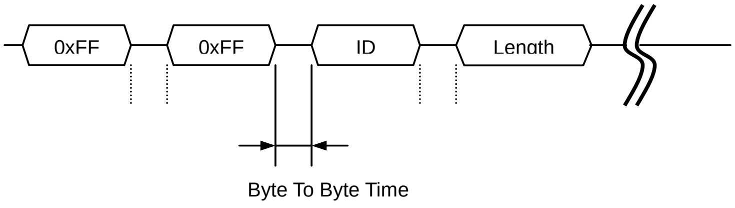
When changing the direction, the TXD_SHIFT_REGISTER_EMPTY_BIT must be checked.

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.



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+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);
}

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_ENA
                      BLE_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+1
                      L) );  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: 4

Model Number

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

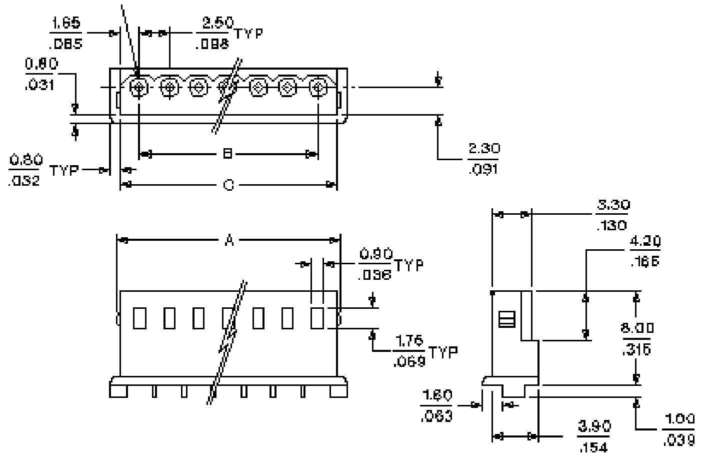
Temperature range : -40°C to +105°C

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

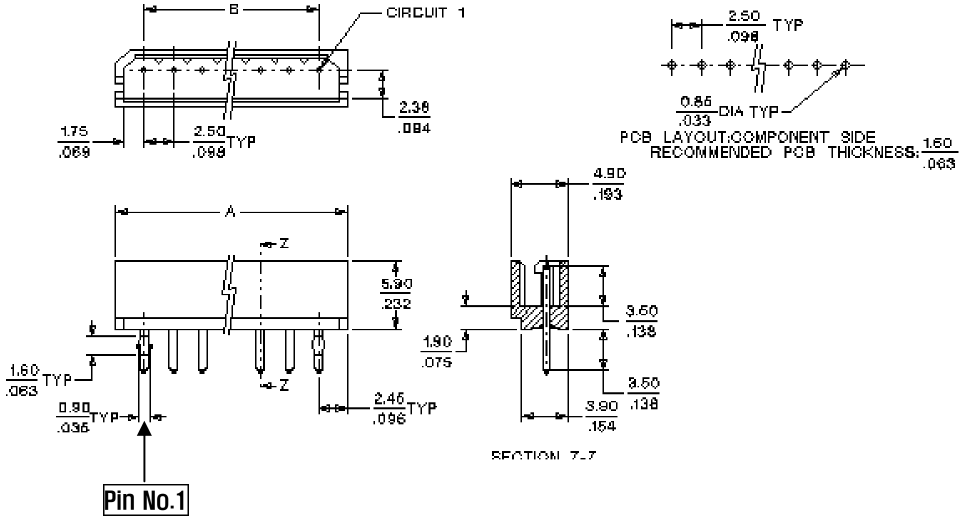
Contact Retention Force-min : 14.7N (3.30 lb)

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

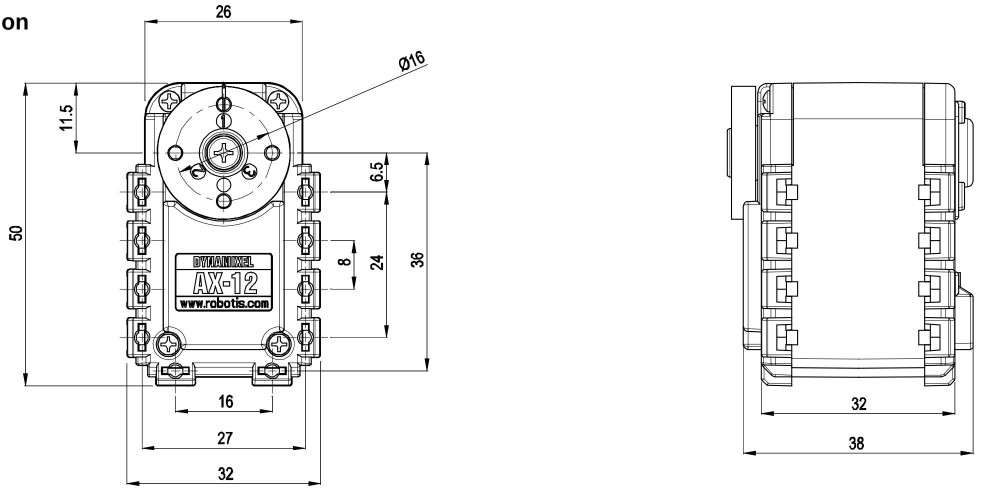
Female Connector



Male Connector



Dimension



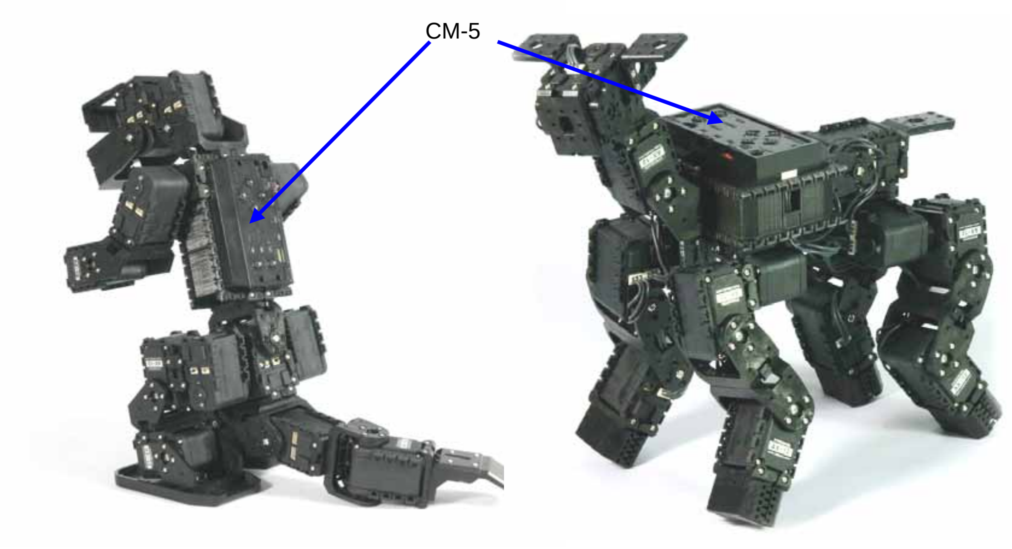
CM-5

Dedicated AX-12 control box. Able to control 30 AX-12 actuators.

6 push buttons (5 for selection, 1 for reset)

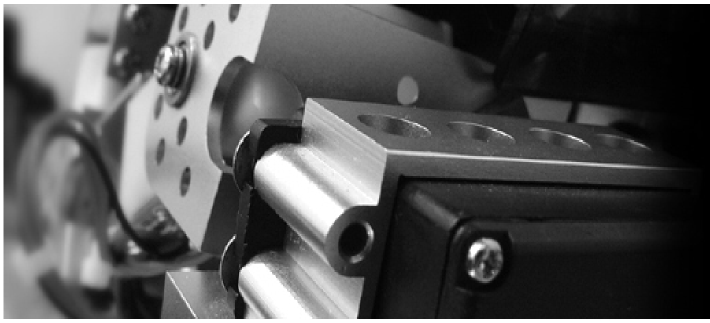
Optional installable wireless devices available

Battery compartment (AA x 8) with recharging capability (when connected to an external SMPS)



5.2 AX-S1

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

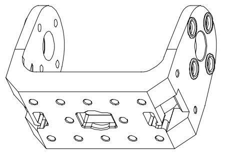
Dynamixel
Networked Sensor Module AX-S1 for Robot Application

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

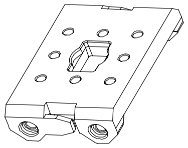
2. Dynamixel Operation

2-1. Mechanical Assembly

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

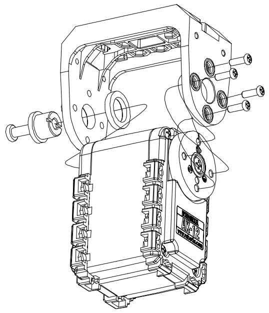


OF-12SH

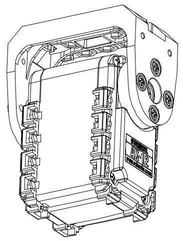


OF-12S

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



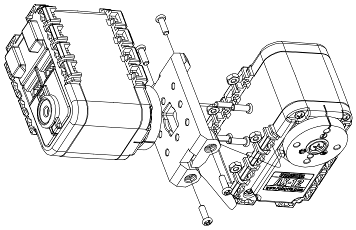
Exploded view



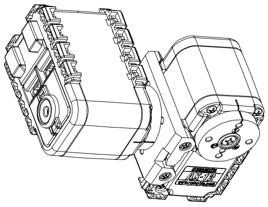
Assembled

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

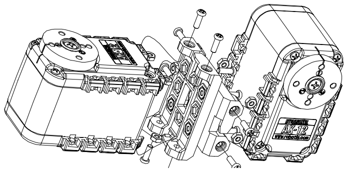


Exploded view

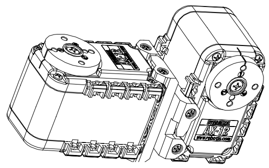


Assembled

Body2Body



Exploded view

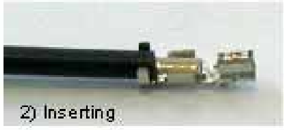


Assembled

2-2. Connector Assembly

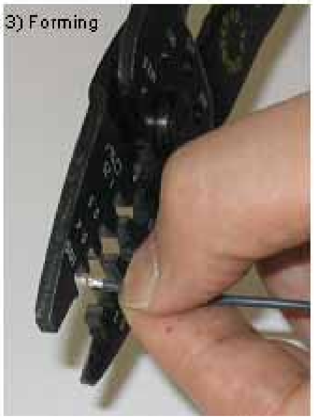
Assemble the connectors as shown below. Attach the wires to the terminals using the correct crimping tool. If you do not have access to a crimping tool, solder the terminals to the wires to ensure that they do not become loose during operation.

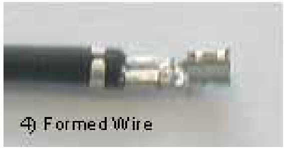
1) Striping



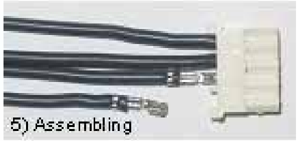
2) Inserting

3) Forming

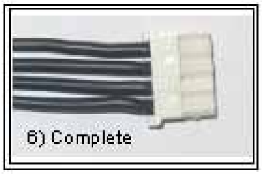




4) Formed Wire



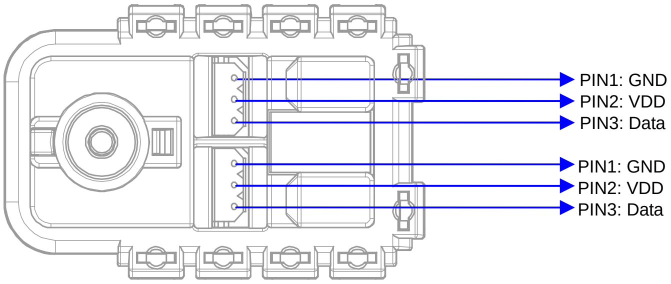
5) Assembling



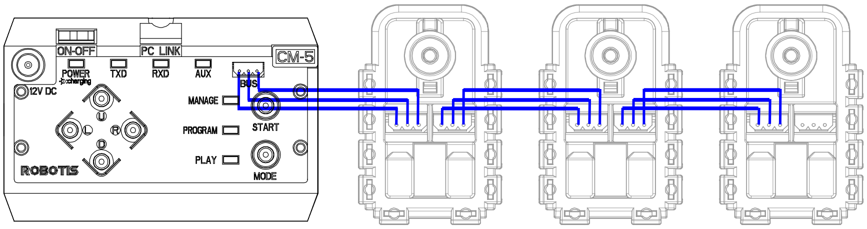
6) Complete

2-3. Dynamixel Wiring

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



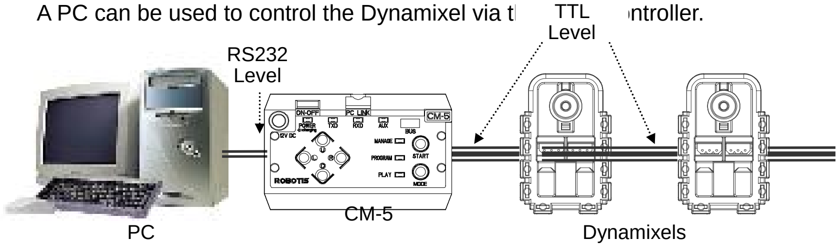
Wiring Connect the AX-2 actuators pin to pin as shown below. Many AX-S1 andAX-12 actuators can be controlled with a single bus in this manner.



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.

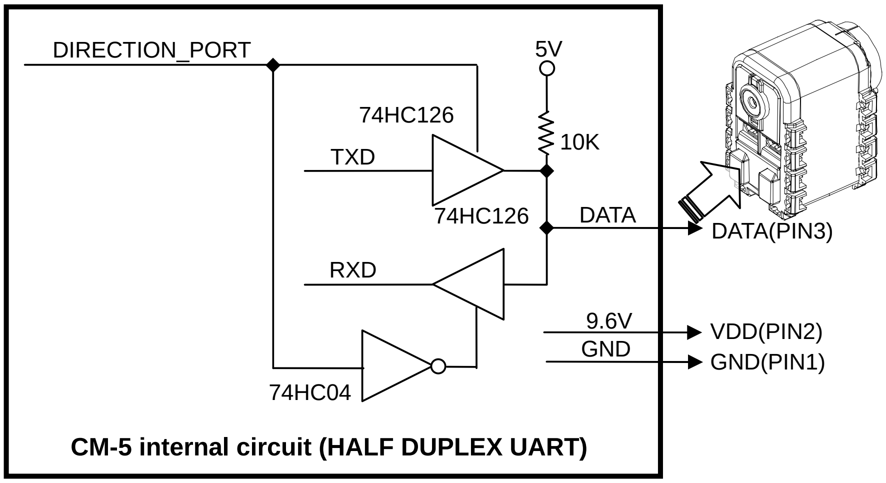


Bioloid A robot can be built using only the CM-5 controller, a number of AX-12 actuators and AX-S1. An edutainment robotic kit named “Bioloid” is available which is based on the CM-5 controller, the AX-12 actuators and AX-S1

An example of a robot built with Bioloid

For details, please refer to the Bioloid manual.

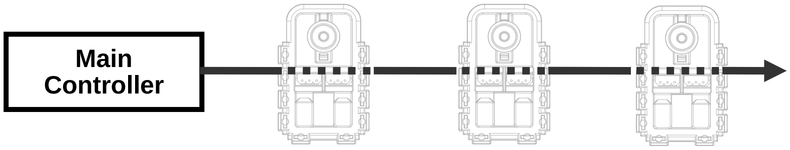
Connection to UART To control the Dynamixel actuators, the main controller needs to convert its UART signals to the half duplex type. The recommended circuit diagram for this is shown below.



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.

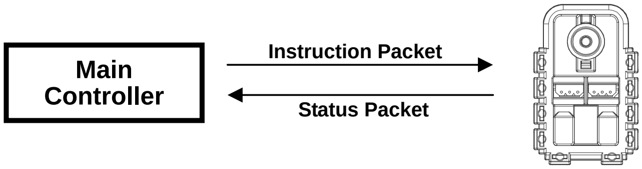
Inspection If the above operation was not successful, then check the connector pin assignment and the voltage/current limit of the power supply.

3. Communication Protocol

3-1. Communication Overview

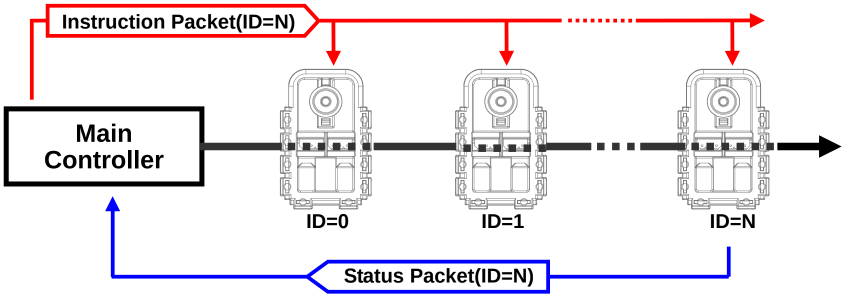
Packet

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

EEPROM Area

RAM Area

Address	Item	Access	Initial Value
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)
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.

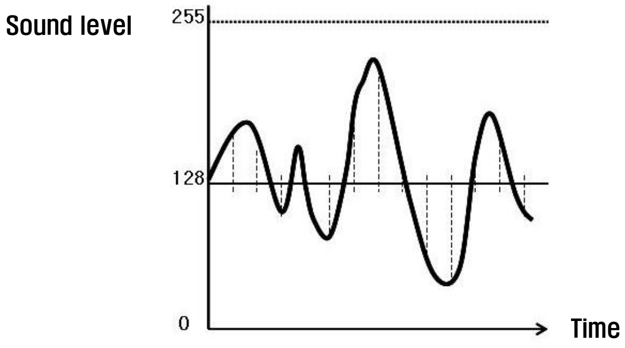
Bit	Representation
Bit 2	An object is detected on the Right Sensor /Light Detected
Bit 1	An object is detected on the Center Sensor /Light Detected
Bit 0	An object is detected on the Left Sensor /Light Detected

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)

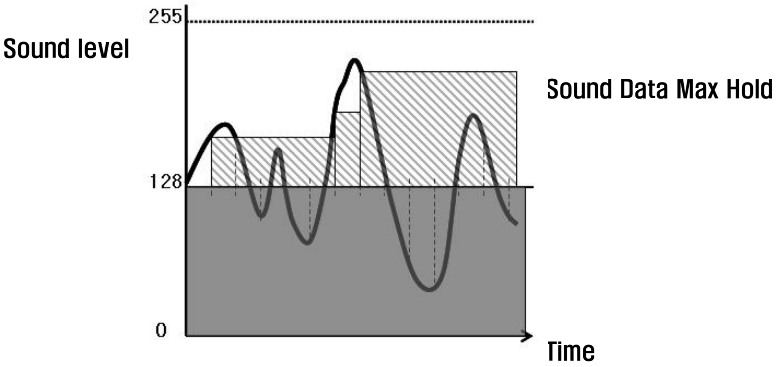
Address 0x23

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.



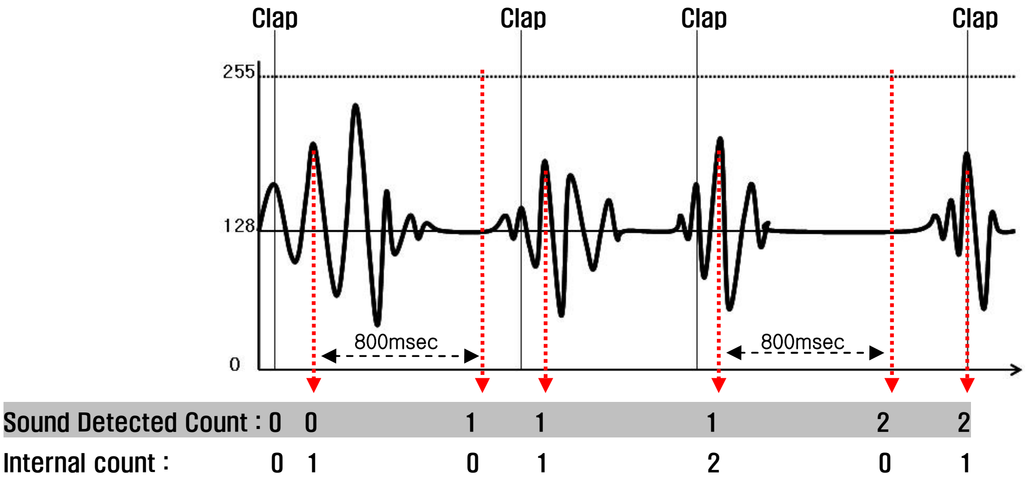
Be cautious as the Sound Data Max Hold is 255 (0xFF) and there is no value that can represent intensity of loudness greater than the optimal loudness, and thus, 255 (0xFF) will be maintained as the Sound Data Max Hold.

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.

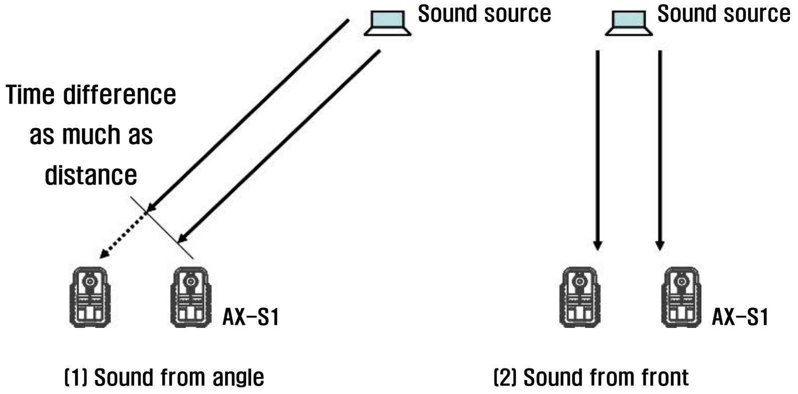


Address 0x26, 0x27

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.



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~

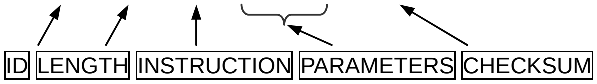
4-1. WRITE_DATA

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 **<u>Setting the ID of a connected Dynamixel actuator to 1</u>**

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.

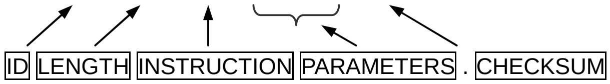
4-2. READ_DATA

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 **<u>Reading the internal temperature of the Dynamixel actuator with an ID of 1</u>**

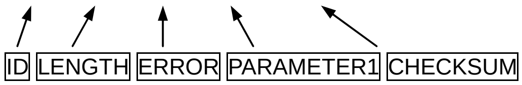
Read 1 byte from address 0x2B of the control table.

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`

ID LENGTH INSTRUCTION CHECKSUM

The returned Status Packet is as the following

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

ID LENGTH ERROR CHECKSUM

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`

ID LENGTH INSTRUCTION CHECKSUM

The returned Status Packet is as the following

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

ID LENGTH ERROR CHECKSUM

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	Data for 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	Data for 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	

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)

Range Error

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

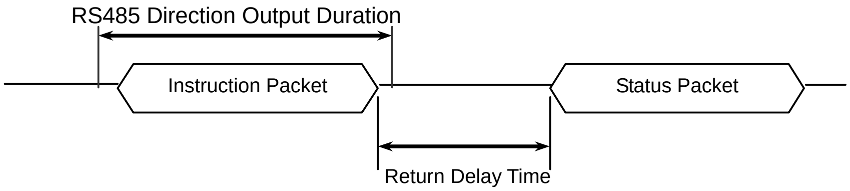
```
->[Dynamixel]:FF FF 00 05 03 30 40 00 87 (LEN:009)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)
```

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
}
```

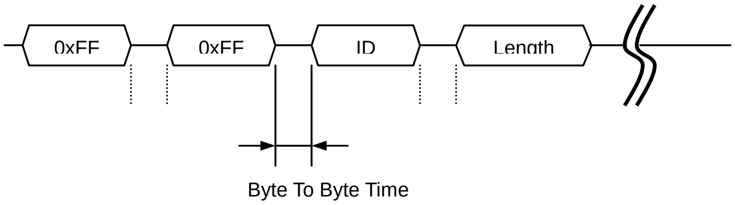
When changing the direction, the TXD_SHIFT_REGISTER_EMPTY_BIT must be checked.

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.



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

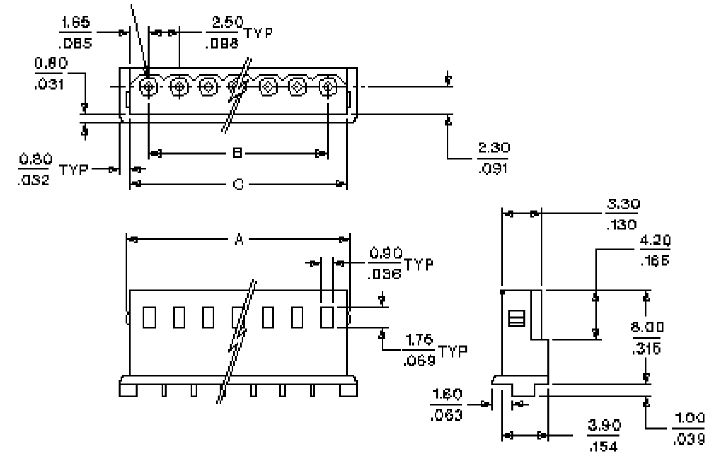
Temperature range : -40°C to +105°C

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

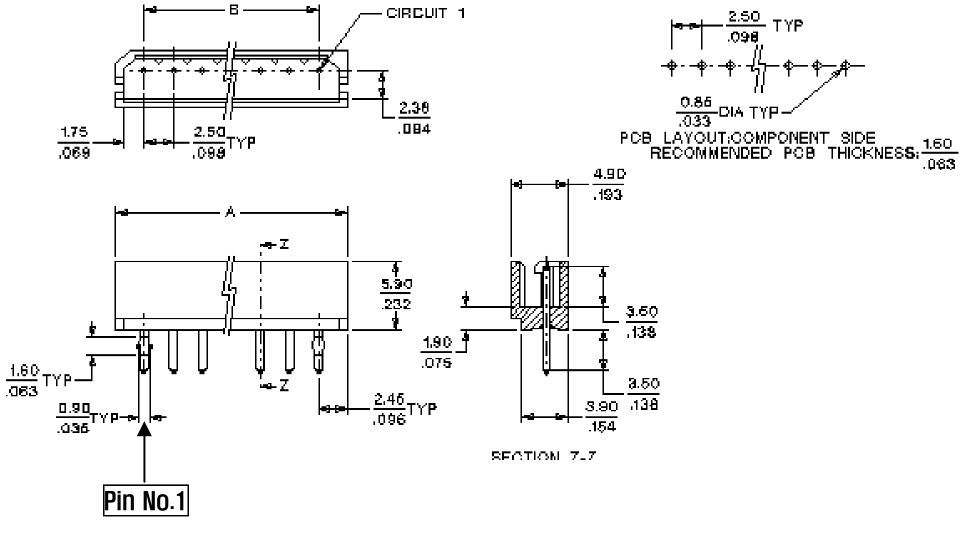
Contact Retention Force-min : 14.7N (3.30 lb)

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

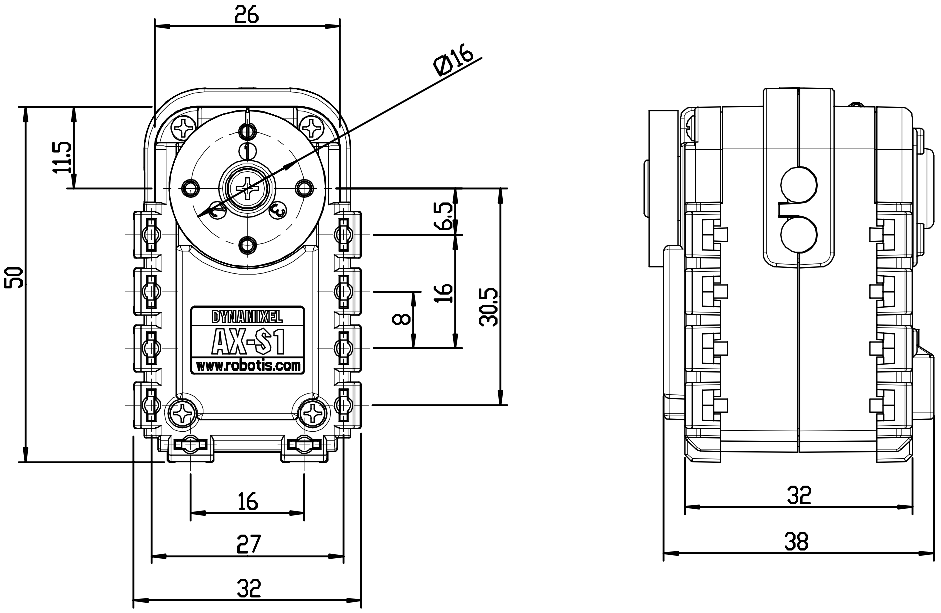
Female Connector



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)