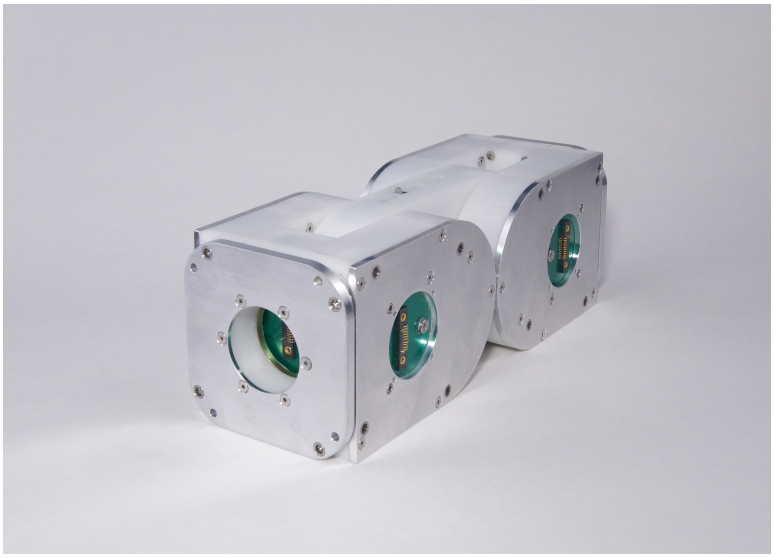
Barobo™

MoBot User's Guide

Version 1.2



Contents

1 The CMobot MoBot Remote Control Library

The CMobot library is a collection of functions geared towards controlling the motors and reading sensor values of a MoBot module via the Bluetooth wireless protocol. The functions are designed to be intuitive and easy to use. Various functions are provided to control or obtain the speed, direction, and position of the motors. The API includes C++ classes called CMobot and CMobotGroup to facilitate control of single and multiple MoBots.

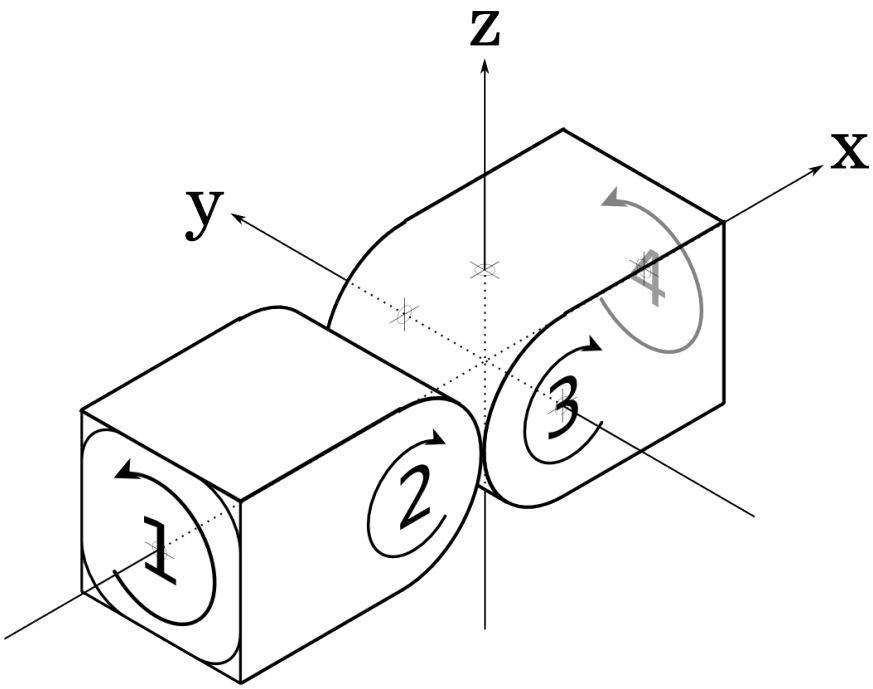


Figure 1: A schematic diagram of a MoBot module.

Figure 1 shows a schematic diagram displaying the locations and positive directions of the four joints of a MoBot module. The joints 1 and 4 shown in the figure are fully rotational and have no joint limits. Joints 2 and 3, however, can only move in the range -90 to +90 degrees.

This documentation introduces the basic computer setup required for controlling the MoBot, as well as several demo programs and a complete reference for all API function provided with the CMobot and CMobotGroup library.

2 Configuring MoBots for Remote Control

MoBot modules should be configured the first time they are used with a new computer. The process informs the computer which MoBots it is allowed to connect to. The is also necessary for certain functions in the CMobot API, such as connect(), to determine which robots to connect to.

The configuration is performed through the Barobo RobotController program. The remainder of the section contains step-by-step instructions and screenshots showing how to configure your MoBots.

To start the provided Barobo Robot Control Program click on the icon labeled “RobotController” on your desktop. The control dialog as shown in Figure 2 should pop up.

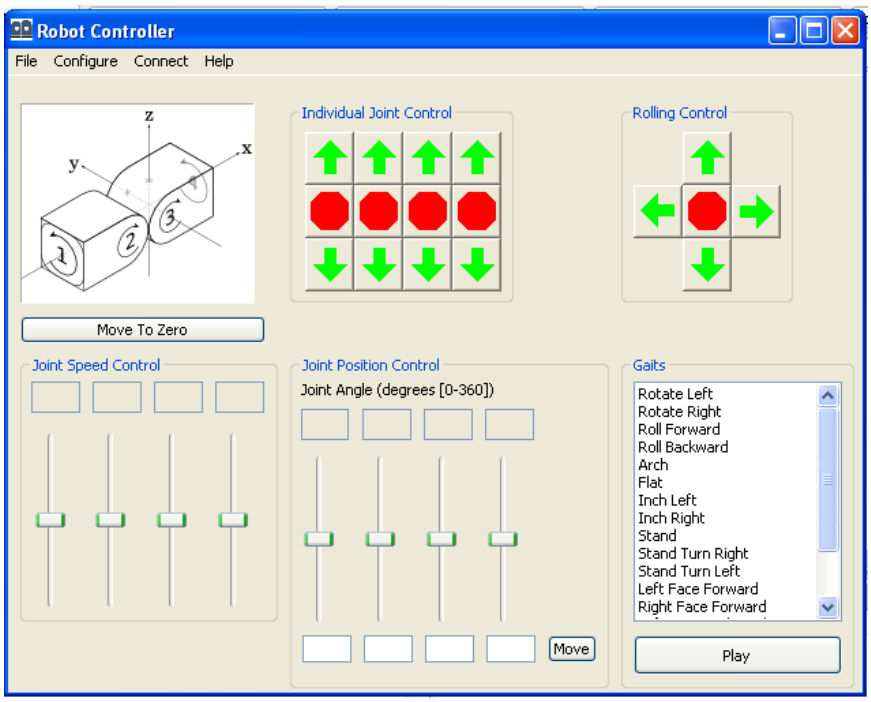


Figure 2: The graphical user interface of the RobotController.

2.1 Adding Bluetooth Addresses of Robots in RobotController.

Click on the menu item "Configure → Configure Robot Bluetooth", as shown in Figure 3.

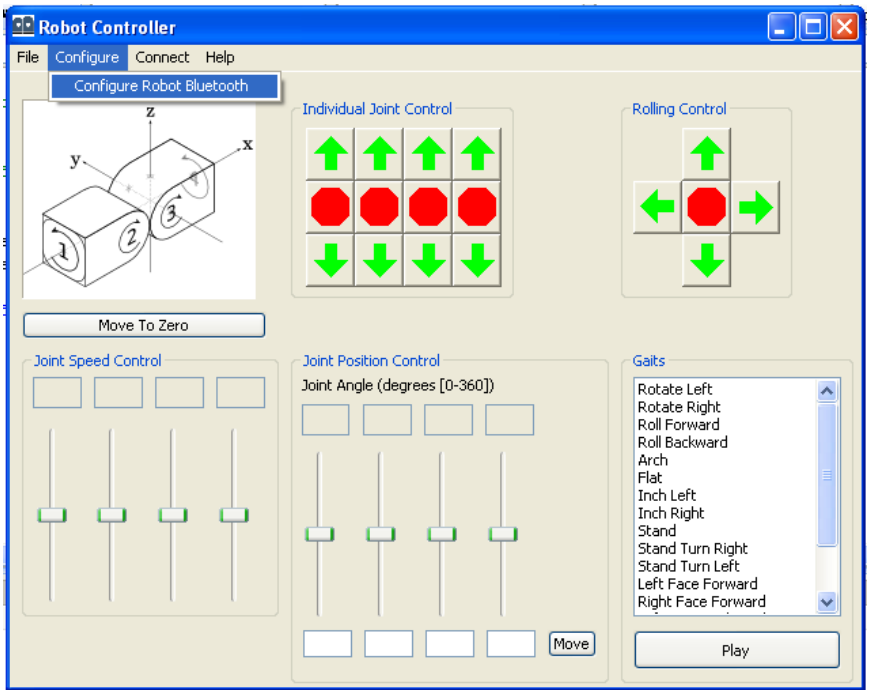


Figure 3: The graphical user interface of the RobotController.

This should bring up a second dialog, titled "Configure Robot Bluetooth", as shown in Figure 4.

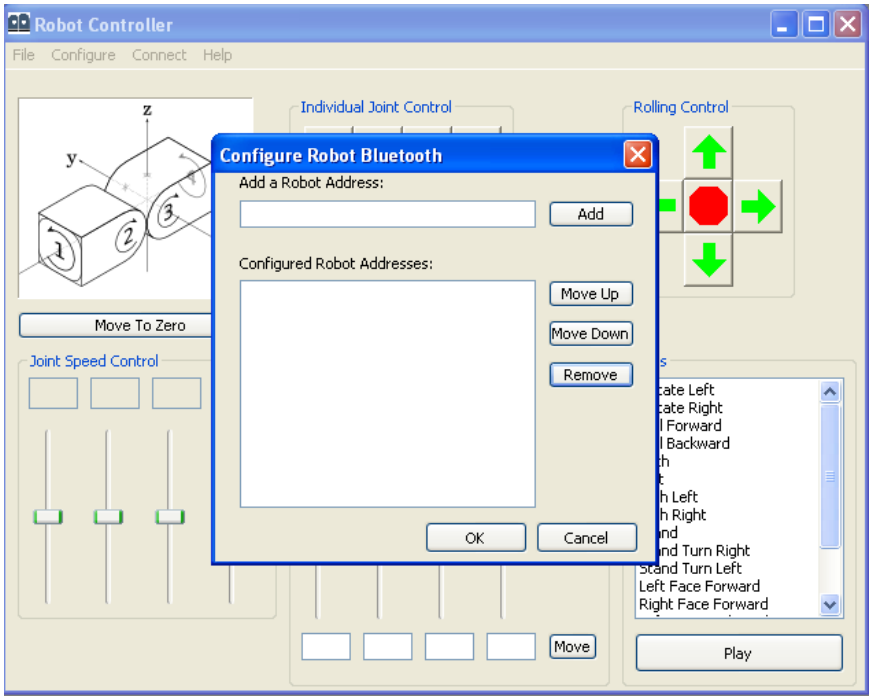


Figure 4: The dialog window for bluetooth connection.

This dialog allows us to add robot bluetooth addresses to the list of currently known robot bluetooth addresses. To add an address, first type in the address in the text box on the top of the dialog, as shown in Figure 5. You can find the bluetooth address of each robot inside the battery compartment of the robot on the same side as the power switch.

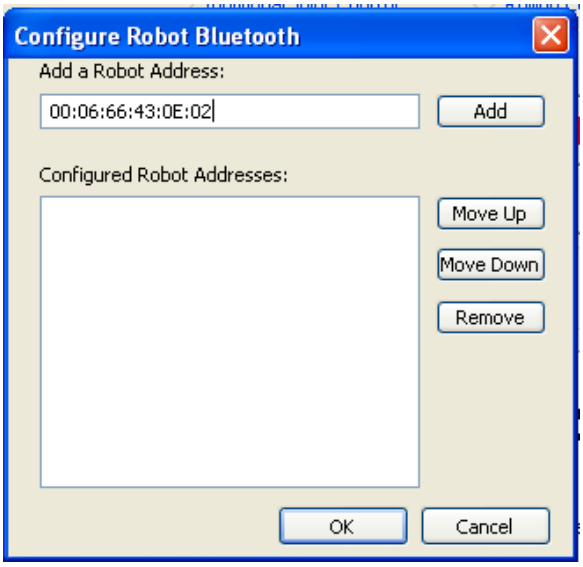


Figure 5: Adding the robot bluetooth address in the dialog window.

Next, click the "Add" button. The newly added address should appear in the list of known addresses, as shown in Figure 6. In our case, we have added the address of one of our robots, which is `"00:06:66:43:0E:02"`.

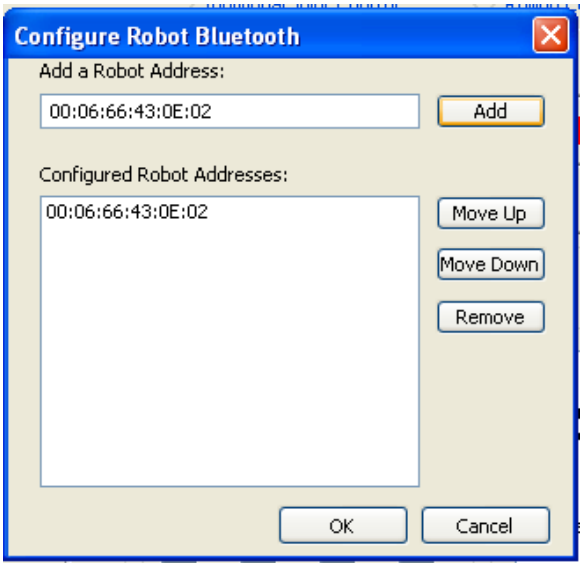


Figure 6: Displaying the added bluetooth address.

We use the same process to add our remaining two robots to the list, with addresses `"00:06:66:43:0D:F2"` and `"00:06:66:47:23:9C"`. The dialog now appears as shown in Figure 7.

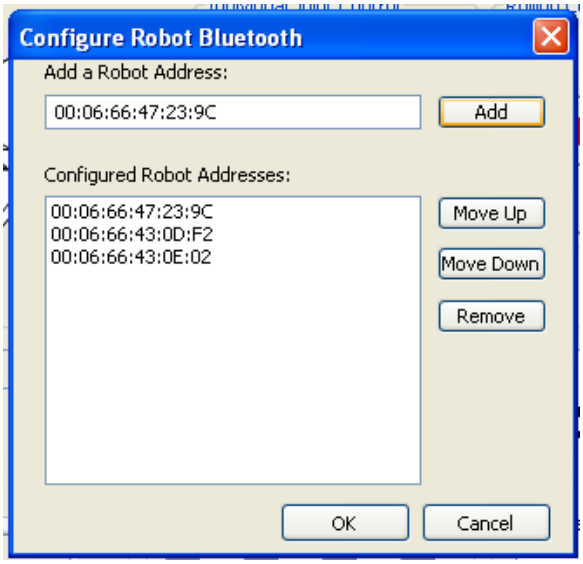


Figure 7: Displaying bluetooth addresses for three robots.

The dialog also allows users to reorder the addresses listed. The order the addresses are listed in affects the order in which the robots are connected to using the `connect()` member function. The remote control dialog

connects to the primary address located at the top of the list by default. To reorder the list of addresses, simply select the address to move and click on the "Move Up" or "Move Down" button to either move the address higher in the list or lower. For instance, the result of clicking on the address `"00:06:66:43:0D:F2"` and clicking the "Move Up" button is shown in Figure 8.

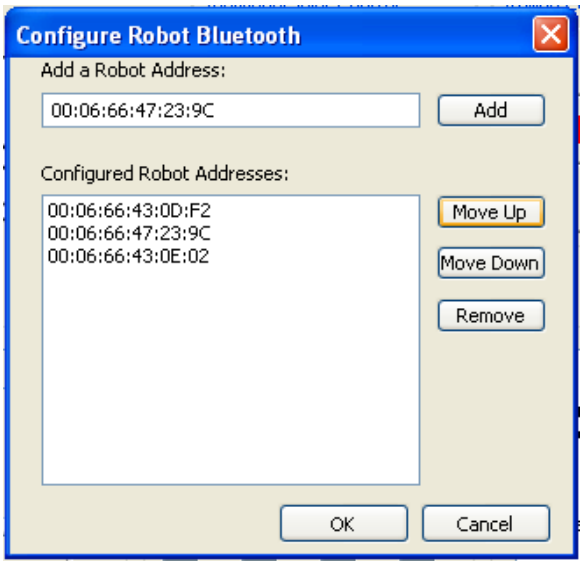


Figure 8: The graphical user interface of the RobotController.

2.2 Connecting and Disconnecting to Robots from the RobotController

Once bluetooth addresses are added to the RobotController, you may connect to the first address by clicking on the "Connect → Connect to Robot" menu item. The connected robot may be disconnected by clicking on the "Connect → Disconnect from Robot" menu item. Any connected robots are automatically disconnected upon exiting the program.

3 The Robot Remote Control Program

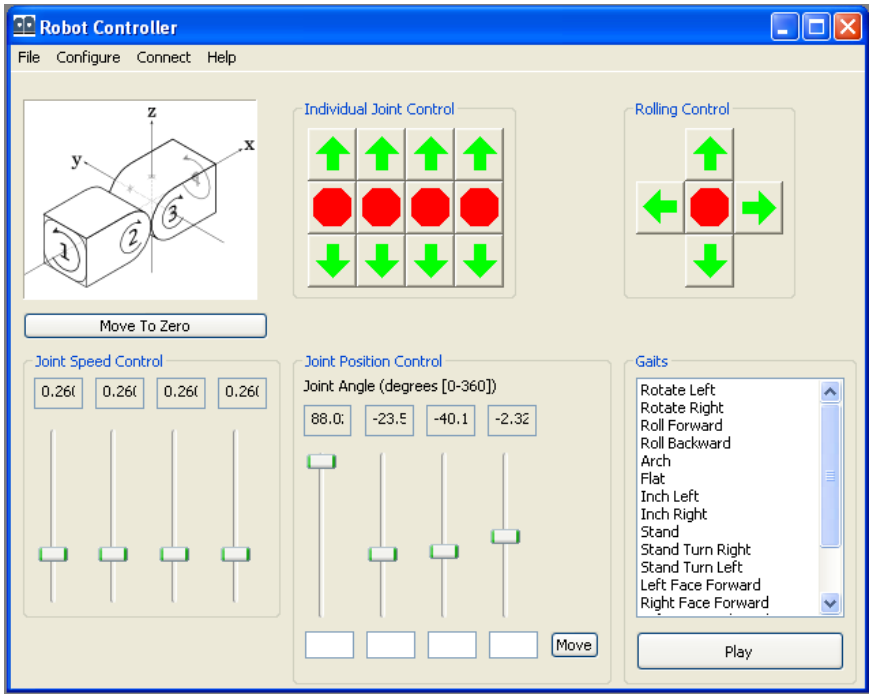


Figure 9: The graphical display of the RobotController while connected to a robot.

Once a robot is connected to the RobotController, the joint angles and speeds of the robot are displayed as shown in Figure 9. The RobotController can then be used to display information about the MoBot's joint positions, and also control the speeds and positions of the MoBot's joints. The interface is divided up into six sections; three on the top half of the interface, and three on the bottom half.

3.1 The MoBot Diagram and "Move To Zero" Button

The first section of the GUI located on the top left of the interface displays a schematic diagram of the MoBot, displaying motor positions. Underneath the diagram, there is a large button with the text "Move To Zero". When clicked, this button will command the connected MoBot to rotate all of its joints to a flat "Zero" position.

3.2 Individual Joint Control

The second section, located at the top-middle section of the interface, is the "Individual Joint Control" section. These buttons command the MoBot to move individual joints. When the up or down arrows are clicked, the MoBot begins to move the corresponding joint in either the positive, or negative direction. The joint will continue to move until the stop button, located between the up and down arrows, is clicked.

If the joint encounters any obstacle that prevents it from moving, the joint will automatically disengage power to the joint. This may happen, example, if a body joint attempts to rotate beyond its limits, or if it collides with the other corresponding body joint.

3.3 Rolling Control

This section contains buttons for controlling the MoBot as a two wheeled mobile robot. The up and down buttons cause the MoBot to roll forward or backward. The left and right buttons cause the MoBot to rotate towards the left, or towards the right. The stop button in the middle causes the MoBot to stop where it is.

3.4 Joint Speeds

The "Joint Speeds" section, located at the bottom left of the interface, displays and controls the current joint speeds of the MoBot. Joint speeds are a value between 0 and 1, with 1 meaning maximum joint power, and 0 meaning zero joint power. The speed may be set by sliding the vertical sliders to the desired positions.

3.5 Joint Positions

This section, located in the bottom-middle of the interface, is used to display and control the positions of each of the four joints of a MoBot. The joint positions are displayed in the numerical text located above each vertical slider. The displayed joint positions are in units of degrees.

The method of controlling the joints is by using the vertical sliders. Each vertical slider's position represents a joint's angle. The sliders for the two end joints vary from -180 degrees to 180 degrees, representing one complete rotation. The angles for the two body joints vary from -90 to 90 degrees. When the position of the slider is moved, the MoBot will move its joints to match the sliders.

Underneath the sliders, there are four text entry boxes. The text boxes accept specific angles for each joint which the user may type in. When the "Move" button is clicked, each joint will move to their respective desired positions. If any text entry is left blank, the joint will not move.

3.5.1 Joint Limits

Joints 1 and 4 are fully rotational and have no joint limits. Joints 2 and 3, however, are limited to a range of -90 to +90 degrees.

3.6 Motions

This section, located on the bottom right of the interface, contains a set of preprogrammed motions for the MoBot. To execute a preprogrammed motion, simply click on the name of the motion you wish to execute, and then click the button labeled "Play".

4 Getting Started Programming the MoBot

The first demo presents a minimal program which connects to a MoBot and moves some joints.

4.0.1 `gettingStarted.ch` Source Code

```
/* Filename: gettingStarted.ch
 * Move the robot endplates. */

#include <mobot.h>

CMobot robot;
double angle1, angle4; // Angles for joints 1 and 4

/* Connect to the paired MoBot */
robot.connect();
```

```
/* Set the robot to "home" position, where all joint angles are 0 degrees. */
robot.moveToZero();

/* Rotate each of the faceplates by 90 degrees */
angle1 = deg2rad(90);
angle4 = deg2rad(90);
robot.move(angle1, 0, 0, angle4);
robot.moveWait();
```

4.0.2 Demo Code for `gettingStarted.ch` Explained

The beginning of every MoBot control program will include header files. Each header file imports functions used for a number of tasks, such as printing data onto the screen or controlling the MoBot. The `mobot.h` header file must be included in order to use the `CMobot` class and related robotic control functions.

```
#include <mobot.h> // Required for MoBot control functions
```

Next, we must initialize the C++ class used to control the MoBot. This line initializes a new variable named `robot` which represents the remote MoBot module which we wish to control. This special variable is actually an instance of the `CMobot` class, which contains its own set of functions called "methods" or "member functions".

```
CMobot robot;
```

Next, we initialize two `double` type variables called "angle1" and "angle4", which we will use to store joint angles.

```
double angle1, angle4;
```

The next line will connect our new variable, `robot`, to a MoBot that has been previously configured with the computer in the process described in Section 2.

```
robot.connect();
```

Note that there are two common methods to connect to a remote MoBot. The most common method, demonstrated in the previous line of code, is used to connect to a MoBot that is already paired to the computer. It is also possible to connect to MoBots which are not paired with the computer. This method is necessary for connecting to multiple MoBots simultaneously, as only a single MoBot may be paired with the computer at a time. The second method uses the function `connectWithAddress()`, and its default usage is as such:

```
string_t address = "11:22:33:44:55:66";
int defaultChannel = 1;
robot.connectWithAddress(address, defaultChannel);
```

The string `"11:22:33:44:55:66"` represents the Bluetooth address of the MoBot, which must be known in advance. The channel number `1` represents the Bluetooth channel to connect to. Channel `1` is the default channel MoBots listen on for incoming connections, but may be set to other values depending on the type of robot.

The next line will use the `moveToZero` member function. The `moveToZero` function causes the MoBot to move all of its motors to the zero position.

```
robot.moveToZero();
```

The next lines of code command joints 1 and 4 to rotate 90 degrees. Note that the member function `move()` expects input angles in radians, so the angles in degrees must be converted to radians using the `deg2rad()` function. If desired, values in radians may also be converted to degrees using the counterpart function, `rad2deg()`. Joints 1 and 4 are the faceplates of the MoBot which are sometimes used to act as "wheels".

```
angle1 = deg2rad(90);
angle4 = deg2rad(90);
robot.move(angle1, 0, 0, angle4);
```

5 Preprogrammed Motions

The robot API contains functions for executing preprogrammed motions. The preprogrammed motions are motions which are commonly used for robot locomotion. Following is a list of available functions and a brief description about their effect on the robot.

- `motionArch()`: This function causes the robot to arch up for better clearance.
- `motionInchwormLeft()`: This function causes the robot to perform the inchworm gait once, moving the robot towards its left.
- `motionInchwormRight()`: This function causes the robot to perform the inchworm gait once, moving the robot towards its right.
- `motionRollBackward()`: This function causes the robot to rotate its faceplates, using them as wheels to roll backward.
- `motionRollForward()`: This function causes the robot to rotate its faceplates, using them as wheels to roll forward.
- `motionStand()`: This function causes the robot to stand up onto a faceplate, assuming the camera platform position.
- `motionTumble()`: This function
- `motionTurnLeft()`: Uses the robot's faceplates as wheels, turning them in opposite directions in order to rotate the robot towards its left.
- `motionTurnRight()`: Uses the robot's faceplates as wheels, turning them in opposite directions in order to rotate the robot towards its right.

Note that all of the functions listed above are "blocking" functions, meaning they will not return until the motion has completed. These functions also have non-blocking equivalents which are discussed in Section 7.

5.1 `inchworm.ch`: A Demo using the `motionInchwormLeft()` Preprogrammed Motion

5.1.1 `inchworm.ch` Source Code

```
/* File: inchworm.ch
 * Perform the "inchworm" gait four times */

#include <mobot.h>
```

```
CMobot robot;

/* Connect to the paired MoBot */
robot.connect();

/* Set robot motors to speed of 0.50 */
robot.setJointSpeedRatio(ROBOT_JOINT2, 0.50);
robot.setJointSpeedRatio(ROBOT_JOINT3, 0.50);

/* Set the robot to "home" position, where all joint angles are 0 degrees. */
robot.moveToZero();

/* Do the inchworm gait four times */
robot.motionInchwormLeft(4);
```

5.1.2 `inchworm.ch` Explained

First, the header file `mobot.h` is included. This header file is required before usage of the `CMobot` class and its associated member functions can be used. Next, we create a variable to reperesent our robot and connect to the robot with the following lines.

```
CMobot robot;

/* Connect to the paired MoBot */
robot.connect();
```

Next, we set the motor speeds to 50% speed with the following lines.

```
robot.setJointSpeedRatio(ROBOT_JOINT2, 0.50);
robot.setJointSpeedRatio(ROBOT_JOINT3, 0.50);
```

We then move the robot to its zero position in preparation for the inchworm gait.

```
robot.moveToZero();
```

Finally, we perform the inchworm gait four times. The argument for the function `motionInchwormLeft()` reperesents the number of times the gait should be performed.

```
robot.motionInchwormLeft(4);
```

5.2 `stand.ch`: A Demo Using the `motionStand()` Preprogrammed Motion

This demo is a simple demonstration of the `motionStand()` member function.

5.2.1 `stand.ch` Source Code

```
/* Filename: stand2.ch
 * Make a MoBot stand up on a faceplate */
#include <mobot.h>

CMobot robot;
/* Connect to the paired MoBot */
robot.connect();
```

```
/* Run the built-in motionStand function */
robot.motionStand();
```

5.2.2 `stand.ch` Explained

After the initialization and connection as seen in the previous demo, it executes the following line of code:

```
robot.motionStand();
```

This line of code causes the MoBot to perform a sequence of motions causing it to stand up on a faceplate. The function is a blocking function and will wait until the entire motion sequence is completed before continuing. There are also non-blocking versions of the motion functions, documented in Section 7.

6 Detailed Examples of Custom Motions

To help the user become acquainted with the MoBot control programs, sample programs will be presented to illustrate the basics and minimum requirements of a MoBot control program. The sample programs are located at `CHHOME/package/chmobot/demos`, where `CHHOME` is the Ch home directory, such as `C:\Ch` for Windows. For Windows, it is located at `C:\Ch\package\chmobot\demos` by default.

6.1 Inchworm Gait Demo

The next demo will demonstrate a simple gait known as the "Inchworm" gait.

6.1.1 `inchworm2.ch` Source Code

```
/* File: inchworm.ch
 * Perform the "inchworm" gait four times */

#include <mobot.h>

CMobot robot;

/* Connect to the paired MoBot */
robot.connect();

/* Set robot motors to speed of 0.50 */
robot.setJointSpeedRatio(ROBOT_JOINT2, 0.50);
robot.setJointSpeedRatio(ROBOT_JOINT3, 0.50);

/* Set the robot to "home" position, where all joint angles are 0 degrees. */
robot.moveToZero();

/* Do the inchworm gait four times */
int i, num = 4;
double angle2 = deg2rad(-45);
double angle3 = deg2rad(45);
for(i = 0; i < num; i++) {
  robot.moveJointTo(ROBOT_JOINT2, angle2);
  robot.moveJointTo(ROBOT_JOINT3, angle3);
  robot.moveJointTo(ROBOT_JOINT2, 0);
```

```
  robot.moveJointTo(ROBOT_JOINT3, 0);
}
```

6.1.2 Demo Code for `inchworm2.ch` Explained

The first portion of the code is identical to the previous demo, and performs the same function of declaring a MoBot variable and connecting to a paired MoBot.

```
#include <mobot.h>

CMobot robot;

/* Connect to the paired MoBot */
robot.connect();
```

The next lines of code set the joint speeds for the two body joints, joints two and three, to 50% speed. They are set to fifty percent speed in order to slow the gait down in order to minimize slippage.

```
/* Set robot motors to speed of 0.50 */
robot.setJointSpeedRatio(ROBOT_JOINT2, 0.50);
robot.setJointSpeedRatio(ROBOT_JOINT3, 0.50);
```

Next, we move the robot into a flat "zero" position.

```
/* Set the robot to "home" position, where all joint angles are 0 degrees. */
robot.moveToZero();
```

Finally, we perform the actual inchworm gait. The inchworm gait is a gait defined by a sequence of motions performed by the body joints. The motions are as such:

1. The first body joint, referred to as joint A, rotates towards the ground. This drags the MoBot towards the direction of joint A.
2. The other body joint, joint B, rotates towards the ground. Since the center of gravity is currently positioned over joint A, this causes the trailing body joint to slide toward joint A.
3. Joint A moves back to a flat position.
4. Joint B moves back to a flat position.
5. Repeat, if desired.

The direction of travel depends on the selection of the initial body joint. In the following code example, joint 2 is chosen as the initial body joint to move. In this case, the MoBot will traverse towards joint 2. The entire gait is encapsulated in a "for" loop which executes the entire gait four times.

```
/* Do the inchworm gait four times */
int i, num = 4;
double angle2 = deg2rad(-45);
double angle3 = deg2rad(45);
for(i = 0; i < num; i++) {
    robot.moveJointTo(ROBOT_JOINT2, angle2);
    robot.moveJointTo(ROBOT_JOINT3, angle3);
    robot.moveJointTo(ROBOT_JOINT2, 0);
    robot.moveJointTo(ROBOT_JOINT3, 0);
}
```

6.2 Standing Demo

6.2.1 `stand2.ch` Source Code

```
/* Filename: stand.ch
 * Make a MoBot stand up on a faceplate */

#include <mobot.h>

CMobot robot;

/* Connect to the paired MoBot */
robot.connect();

/* Set robot motors to slow speed of 10 degrees per second */
robot.setJointSpeed(ROBOT_JOINT2, deg2rad(10));
robot.setJointSpeed(ROBOT_JOINT3, deg2rad(10));
/* Set the robot to "home" position, where all joint angles are 0 degrees. */
robot.moveToZero();

/* Move the robot into a fetal position */
robot.moveJointTo(ROBOT_JOINT2, deg2rad(-85));
robot.moveJointTo(ROBOT_JOINT3, deg2rad(80));

/* Rotate the bottom faceplate by 45 degrees */
robot.moveJointTo(ROBOT_JOINT1, deg2rad(45));

/* Lift the body up */
robot.moveJointTo(ROBOT_JOINT2, deg2rad(20));

/* Pan the robot around for 3 seconds at 90 degrees per second*/
robot.setJointSpeed(ROBOT_JOINT1, deg2rad(90));
robot.moveContinuousTime(ROBOT_FORWARD, ROBOT_NEUTRAL, ROBOT_NEUTRAL,
                         ROBOT_NEUTRAL, 3000);
```

6.2.2 `stand2.ch` Explained

The first portion of the program performs the necessary setup and connecting, similar to the previous demos. Similar to the previous inchworm demo, the motor speeds are set to a speed of 10 degrees per second, and the function `moveToZero()` is called to put the robot into a flat position. Next, the following lines are executed:

```
robot.moveJointTo(ROBOT_JOINT2, deg2rad(-85));
robot.moveJointTo(ROBOT_JOINT3, deg2rad(80));
```

These movement commands cause the MoBot to curl up into a fetal position with both of its endplates facing toward the ground. Next, the MoBot rotates one of the endplates by 45 degrees.

```
robot.moveJointTo(ROBOT_JOINT1, deg2rad(45));
```

This endplate will eventually become the “foot” of the standing MoBot. Next, the MoBot lifts itself into a standing position, balancing on its endplate.

```
robot.moveJointTo(ROBOT_JOINT2, deg2rad(20));
```

Note that the previous joint angle for Joint 2, a body joint, was -85 degrees. This motion causes joint 2 to rotate all the way to a 20 degree position, which lift up the body of the MoBot such that the MoBot is balancing on faceplate joint 1.

Finally, we rotate joint 1, the foot joint, for three seconds which causes the entire MoBot to rotate in place. The speed is first set to 30% speed to make the rotation a slow rotation. Next, the `moveContinuousTime` member function is used to continuously rotate a joint for a desired amount of time.

```
robot.setJointSpeed(ROBOT_JOINT1, 0.30);
robot.moveContinuousTime(ROBOT_JOINT_FORWARD, ROBOT_JOINT_NEUTRAL, ROBOT_JOINT_NEUTRAL,
                         ROBOT_JOINT_NEUTRAL, 3000);
```

The macros `ROBOT_JOINT_FORWARD, ROBOT_JOINT_NEUTRAL,` and `ROBOT_JOINT_BACKWARD` indicate the directions for each motor to turn. More information regarding these macros may be found in Section **??**.

6.3 New Samples to be Documented

```
/* Filename: simple.ch
 * Rotate the faceplates by 90 degrees */

#include <mobot.h>

CMobot robot;

/* Connect to the paired MoBot */
if(robot.connect())
{
printf("Connect error.\n");
exit(0);
}
/* Set the robot to "home" position, where all joint angles are 0 degrees. */
robot.moveToZero();

/* Rotate each of the faceplates by 90 degrees */
robot.move(deg2rad(90), 0, 0, deg2rad(90));
/* Move the motors back to where they were */
robot.move(deg2rad(-90), 0, 0, deg2rad(-90));
```

```
/* Filename: tumble.ch
 * Tumbling robot */

#include <mobot.h>
#define deg2rad(x) ((x) * M_PI/180.0)

CMobot robot;

/* Connect to the paired MoBot */
robot.connect()
/* Set the robot to "home" position, where all joint angles are 0 degrees. */
robot.moveToZero();

/* Tumble five times */
robot.motionTumble(5);
```

```
/* Filename: tumble2.ch
 * Tumbling robot */

#include <mobot.h>
#define deg2rad(x) ((x) * M_PI/180.0)

CMobot robot;

/* Connect to the paired MoBot */
if(robot.connect())
{
printf("Connect error.\n");
exit(0);
}

/* Set the robot to "home" position, where all joint angles are 0 degrees. */
robot.moveToZero();

/* Move the robot into a fetal position */
robot.moveJointTo(ROBOT_JOINT2, deg2rad(-85));
robot.moveJointTo(ROBOT_JOINT3, deg2rad(80));

/* Set the robot to "home" position, where all joint angles are 0 degrees. */
//robot.moveToZero();

/* Move the robot into a fetal position */
robot.moveJointTo(ROBOT_JOINT2, deg2rad(0));
robot.moveJointTo(ROBOT_JOINT3, deg2rad(0));
robot.moveJointTo(ROBOT_JOINT2, deg2rad(60));
robot.moveJointTo(ROBOT_JOINT3, deg2rad(-85));
robot.moveJointTo(ROBOT_JOINT2, deg2rad(80));

robot.moveJointTo(ROBOT_JOINT3, deg2rad(0));
robot.moveJointTo(ROBOT_JOINT2, deg2rad(0));
robot.moveJointTo(ROBOT_JOINT3, deg2rad(60));
robot.moveJointTo(ROBOT_JOINT2, deg2rad(-85));
robot.moveJointTo(ROBOT_JOINT3, deg2rad(80));

//robot.moveTo(0, deg2rad(70), deg2rad(-90), 0);

/* Move the robot into a fetal position */
//robot.moveJointTo(ROBOT_JOINT3, deg2rad(-85));
//robot.moveJointTo(ROBOT_JOINT2, deg2rad(80));

/* Set the robot to "home" position, where all joint angles are 0 degrees. */
//robot.moveToZero();

/* Move the robot into a fetal position */
//robot.moveJointTo(ROBOT_JOINT2, deg2rad(-85));
//robot.moveJointTo(ROBOT_JOINT3, deg2rad(80));
```

```
/* Filename: motion.ch
 * Move the two wheeled robot. */

#include <mobot.h>

CMobot robot;

/* Connect to the paired MoBot */
robot.connect();

/* Set the robot to "home" position, where all joint angles are 0 degrees. */
robot.moveToZero();

/* test all pre-programmed motions */
robot.motionInchwormLeft(1);
robot.motionInchwormRight(1);
robot.motionRollBackward(deg2rad(360));
robot.motionRollForward(deg2rad(360));
robot.motionTurnLeft(deg2rad(360));
robot.motionTurnRight(deg2rad(360));
robot.motionStand();
robot.motionUnstand();
```

```
/* Filename: lift.ch
   Control two modules and make them stand simultaneously.
   The joint4 of the first robot should be connected to the joint1 of the second robot. */

#include <mobot.h>

CMobot robot1;
CMobot robot2;

/* Connect robot variables to the robot modules. */
robot1.connect();
robot2.connect();

/* Set the robot to "home" position, where all joint angles are 0 degrees. */
robot1.moveToZeroNB();
robot2.moveToZeroNB();

robot1.moveWait();
robot2.moveWait();

/* First lift */
robot1.moveNB(0, deg2rad(-90),  0, 0);
robot2.moveNB(0, 0, deg2rad(90), 0);
robot1.moveWait();
robot2.moveWait();
/* Second lift */
robot1.moveToNB(0, 0, deg2rad(90),  0);
robot2.moveToNB(0,  deg2rad(-90), 0, 0);
```

```
robot1.moveWait();
robot2.moveWait();

/* File: nonblock.ch
   use the non-blocking functoin move() . */
#include <mobot.h>

CMobot robot;

/* Connect to the paired MoBot */
robot.connect();

robot.moveToZero();
/* Rotate each of the faceplates by 90 degrees */

//robot.move(5*deg2rad(360), 0, 0, 5*deg2rad(360)); // Blocking version
robot.moveNB(2*deg2rad(360), 0, 0, 2*deg2rad(360)); // Non-Blocking version
while(robot.isMoving()) {
    printf("robot is moving ...\n");
}
printf("move finished!\n");
```

```
/* File: nonblock2.ch
   use the non-blocking functoin move() . */
#include <mobot.h>

CMobot robot;

/* Connect to the paired MoBot */
robot.connect();

robot.moveToZero();
/* Rotate each of the faceplates by 90 degrees */

//robot.moveJoint(ROBOT_JOINT1, deg2rad(360)); // Blocking version
robot.moveJointNB(ROBOT_JOINT1, deg2rad(360)); // Non-Blocking version
robot.moveJoint(ROBOT_JOINT4, deg2rad(360));
```

```
/* File: nonblock3.ch
   Roll and arch simultaneously. */
#include <mobot.h>

CMobot robot;

/* Connect to the paired MoBot */
robot.connect();

robot.moveToZero();
/* Rotate each of the faceplates by 90 degrees */

printf("Rolling 360 degrees.\n");
robot.motionRollForward(deg2rad(360));
```

```
printf("Rolling while arching for 360 degrees.\n");
robot.motionArchNB(deg2rad(90));
robot.motionRollForwardNB(deg2rad(360));
robot.motionWait();

/* Filename: setspeed.ch
  Move the two wheeled robot with different speed. */

#include <mobot.h>
#include <math.h>

CMobot robot;

/* Connect to the paired MoBot */
robot.connect();

/* Set the robot to "home" position, where all joint angles are 0 degrees. */
robot.moveToZero();

double speed, radius;
robot.getJointMaxSpeed(ROBOT_JOINT1, speed);
printf("The maximum speed is %lf degrees/s\n", rad2deg(speed));

robot.setJointSpeed(ROBOT_JOINT1, deg2rad(90));
robot.setJointSpeed(ROBOT_JOINT4, deg2rad(90));

//robot.setJointSpeedRatio(ROBOT_JOINT1, 0.5);
//robot.setJointSpeedRatio(ROBOT_JOINT4, 0.5);

//robot.setJointSpeeds(deg2rad(90), 0, 0, deg2rad(90));

printf("Roll forward 360 degrees.\n");
robot.motionRollForward(deg2rad(360));

speed = (3.5/2) * M_PI / 2;      // 2.75 inch/s
radius = 3.5/2;     // radius is 1.75
robot.setTwoWheelRobotSpeed(speed, radius, "inch");

printf("Move 360 degrees.\n");
robot.move(deg2rad(360), 0, 0, deg2rad(360));

/* move at 2inch/sec with the radius 3.5 inches for 3 seconds */
printf("Move continuously for 3 seconds.\n");
robot.moveContinuousTime(ROBOT_JOINT_FORWARD, ROBOT_JOINT_HOLD,
                         ROBOT_JOINT_HOLD, ROBOT_JOINT_FORWARD, 3000);
```

7 Blocking and Non-Blocking Functions

All of the MoBot movement functions may be designated as either "blocking" functions or "non-blocking" functions. A blocking function is a function which does not return while operations are being performed. For instance, the `moveWait()` function is a blocking function. When called, the function will hang, or "block",

until all the joints have stopped moving. After all joints have stopped moving, the `moveWait()` function will return, and the rest of the program will execute.

Furthermore, some functions have both a blocking version and a non-blocking version. For these functions, the suffix `NB` denotes that the function is non-blocking. For instance, the function `motionStand()` is blocking, meaning the function will not return until the motion is completed, whereas the function `motionStandNB()` is non-blocking, meaning the function returns immediately and the robot performs the "standing" motion asynchronously.

The function `moveNB()` is an example of a non-blocking function. When the `moveNB()` function is called, the function immediately returns as the joints begin moving. Any lines of code following the call to `moveNB()` will be executed even if the current motion is still in progress.

Demos for the non-blocking functions are located in the next section of this document.

7.1 List of Blocking Movement Functions

- `move()`
- `moveContinuousTime()`
- `moveJointTo()`
- `moveTo()`
- `moveToZero()`
- `moveJointWait()`
- `moveWait()`
- `motionInchwormLeft()`
- `motionInchwormRight()`
- `motionRollBackward()`
- `motionRollForward()`
- `motionStand()`
- `motionTurnLeft()`
- `motionTurnRight()`

7.2 List of Non-Blocking Movement Functions

- `moveNB()`
- `moveContinuousNB()`
- `moveJointToNB()`
- `moveToNB()`
- `moveToZeroNB()`
- `motionInchwormLeftNB()`
- `motionInchwormRightNB()`

- `motionRollBackwardNB()`
- `motionRollForwardNB()`
- `motionStandNB()`
- `motionTurnLeftNB()`
- `motionTurnRightNB()`

8 Controlling Multiple Modules

The MoBot control software is designed to be able to control multiple modules simultaneously. There are a couple important differences in the program which enable the control of multiple modules. A small demo program which controls two modules simultaneously will first be presented, followed by a detailed explanation of the program elements.

8.1 `multipleModules.ch` Source Code

```
/* Filename: multipleModules.ch
 * Control two modules and make them stand simultaneously. */

#include <mobot.h>

CMobot robot1;
CMobot robot2;

/* Connect robot variables to the robot modules. The */
robot1.connect();
robot2.connect();

/* Set the robot to "home" position, where all joint angles are 0 degrees. */
robot1.moveToZeroNB();
robot2.moveToZeroNB();

robot1.moveWait();
robot2.moveWait();

/* Instruct the first robot to stand and the second robot to inchworm
 * simultaneously. */
robot1.motionStandNB();
robot2.motionInchwormLeftNB();
robot1.motionWait();
robot2.motionWait();
```

8.2 Demo Explanation

The first two lines of interest appear as such:

```
CMobot robot1;
CMobot robot2;
```

These two lines declare two separate variables which will represent the two separate MoBot modules. Next, we need to connect each variable to a physically separate MoBot. This is done with the following lines.

```
robot1.connect();
robot2.connect();
```

These two lines connect the robots to the first two addresses of the known robot addresses. The list of the computer's known robot addresses may be configured in the process detailed in Section 2 on page 6. For each separate control program, the first call to the `connect()` member function will connect to the first robot listed in the configuration file. Each successive call to the `connect()` function will connect to successive robots listed in the configuration file. The order in which they are connected may be modified using the "Configure Robot Bluetooth" dialog, as discussed in Section 2.

```
robot1.moveToZeroNB();
robot2.moveToZeroNB();
```

These two lines command the two robots to move to their zero positions. Note that these functions are non-blocking. This means that the `moveToZeroNB()` function will return immediately, and will not wait for the first robot to finish completing the motion before commanding the second robot to begin. In a normal program, this effectively causes both robots to move to their zero positions simultaneously.

```
robot1.moveWait();
robot2.moveWait();
```

Since the `moveToZeroNB()` functions are non-blocking, we would like the program to wait until the motions are complete before continuing. By calling `moveWait()` on both of the robots, we can be assured that the robots have finished moving before the program continues.

```
robot1.motionStandNB();
robot2.motionInchwormLeftNB();
robot1.motionWait();
robot2.motionWait();
```

Similar to the calls to `moveToZeroNB()`, this block of code instructs the first MoBot to stand and the second MoBot to perform the inchworm gait. Note that we call the non-blocking versions of the functions, `motionStandNB()` and `motionInchwormLeftNB`. Since these functions are non-blocking, both robots will effectively perform the motions simultaneously. Next, we call the member function `motionWait()` to wait for the motions to finish before exiting the program.

9 Commanding Multiple Robots to Perform Identical Tasks

There is a specialized class called `CMobotGroup` which may be used to control multiple modules simultaneously. The `CMobotGroup` represents a group of robots. Any command that is given to the group of modules is duplicated to each member of the group.

The majority of the movement functions available in the `CMobot` class are also available in the `CMobotGroup` class. For documentation of each member function, please consult the documentation for the corresponding functions in the `CMobot` class. Following is a complete listing of the available member functions in the `CMobotGroup` class.

Function Name	Description
`addRobot()`	Adds a robot to the group.
`move()`	Move joints on robots from their current positions for all robots in the group.
`moveContinuousNB()`	Move joints continuously for all robots in the group.
`moveContinunousTime()`	Move joints continuously for a specific amount of time for all robots in the group.
`moveJointContinuousNB()`	Move a single joint continuously for all robots in the group.
`moveJointContinuousTime()`	Move a single joint continuously for a specific amount of time for all robots in the group.
`moveJointTo()`	Move a joint to an absolute angle for all robots in the group.
`moveJointWait()`	Wait for a joint to finish moving for all robots in the group.
`moveTo()`	Move joints to a set of specific angles for all robots in the group.
`moveWait()`	Wait for all joints to finish moving.
`moveToZero()`	Move all joints to their zero positions.
`setJointSpeed()`	Set the speed of a joint, in radians per second.
`setJointSpeedRatio()`	Set the speed of a joint as a ratio of the maximum speed.
`setTwoWheelRobotSpeed()`	Move the robot at a linear speed, given the wheel size and desired speed.
`stop()`	Stop all the joints of all robots in the group.
`motionInchwormLeft()`	Cause all robots in the group to inchworm.
`motionInchwormLeftNB()`	Cause all robots in the group to inchworm.
`motionInchwormRight()`	Cause all robots in the group to inchworm.
`motionInchwormRightNB()`	Cause all robots in the group to inchworm.
`motionRollBackward()`	Makes all robots in the group roll backward.
`motionRollForwardNB()`	Makes all robots in the group roll forward.
`motionStand()`	Makes all robots in the group stand.
`motionStandNB()`	Makes all robots in the group stand.
`motionTurnLeft()`	Makes all robots in the group turn left.
`motionTurnLeftNB()`	Makes all robots in the group turn left.
`motionTurnRight()`	Makes all robots in the group turn right.
`motionTurnRightNB()`	Makes all robots in the group turn right.

9.1 Demo program `group.ch`

9.1.1 Source Code

```
/* Filename: group.ch
 * Control multiple MoBot modules simultaneously using the CMobotGroup class */

#include <mobot.h>

CMobot robot1;
CMobot robot2;

CMobotGroup group;
```

```
/* Connect to the robots listed in the configuration file. */
robot1.connect();
robot2.connect();

/* Add the two modules to be members of our group */
group.addRobot(robot1);
group.addRobot(robot2);

/* Now, any commands given to "group" will cause both robot1 and robot2 to
 * execute the command. */
group.motionInchwormLeft(); /* Cause both robots to inchworm left */
group.motionStand(); /* Cause both robots to stand */
```

9.1.2 Demo Explanation

The first lines of interest appear as such:

```
CMobot robot1;
CMobot robot2;

CMobotGroup group;
```

These lines declare two robot variables, and one variable which will represent a group of robots. Next, we connect the robot variables to their physical counterparts.

```
robot1.connect();
robot2.connect();
```

Once they are connected, we wish to add both of these robots to our robot group, which we have named `group`.

```
group.addRobot(robot1);
group.addRobot(robot2);
```

Finally, we wish for all of the robots in our robot group, namely `robot1` and `robot2`, to peform an inchworm gait, followed by a standing gait. This is done with the following lines:

```
group.motionInchwormLeft(); /* Cause both robots to inchworm left */
group.motionStand(); /* Cause both robots to stand */
```

A Data Types

There are data types which are used by the MoBot library to represent certain values, such as joint id's and motor directions.

Data Type	Description
`robotJointId_t`	An enumerated value that indicates a MoBot joint.
`robotJointState_t`	The current state of a MoBot joint.

A.1 `robotJointId_t`

This datatype is an enumerated type used to identify a joint on the MoBot. Valid values for this type are:

```
typedef enum mobot_joints_e {
  ROBOT_JOINT1 = 1,
  ROBOT_JOINT2 = 2,
  ROBOT_JOINT3 = 3,
  ROBOT_JOINT4 = 4
} robotJointId_t;
```

Value	Description
`ROBOT_JOINT1`	Joint number 1 on the MoBot, which is a faceplate joint.
`ROBOT_JOINT2`	Joint number 2 on the MoBot, which is a body joint.
`ROBOT_JOINT3`	Joint number 3 on the MoBot, which is a body joint.
`ROBOT_JOINT4`	Joint number 4 on the MoBot, which is a faceplate joint.

A.2 `robotJointState_t`

This datatype is an enumerated type used to designate the current movement state of a joint. The values may be retrieved from the robot with the `getJointState()` function and may be set with the `moveContinuous()` family of functions. Valid values are:

```
typedef enum mobot_joint_state_e {
  ROBOT_NEUTRAL   = 0,
  ROBOT_FORWARD   = 1,
  ROBOT_BACKWARD  = 2,
  ROBOT_HOLD      = 3
} robotJointState_t;
```

Value	Description
`ROBOT_NEUTRAL`	This value indicates that the joint is not moving and is not actuated. The joint is freely backdrivable.
`ROBOT_FORWARD`	This value indicates that the joint is currently moving forward.
`ROBOT_BACKWARD`	This value indicates that the joint is currently moving backward.
`ROBOT_HOLD`	This value indicates that the joint is currently not moving and is holding its current position. The joint is not currently backdrivable.

B CMobot API

The header file **mobot.h** defines all the data types, macros and function prototypes for the robot API library. The header file declares a class called `CMobot` which contains member functions which may be used to control the robot.

Table 1: CMobot Member Functions.

Function	Description
`CMobot()`	The CMobot constructor function. This function is called automatically and should not be called explicitly.
`~CMobot()`	The CMobot destructor function. This function is called automatically and should not be called explicitly.
`connect()`	Connect to a remote robot module. This function connects to the first robot listed in the Barobo configuration file. To edit the configuration file, use the robot control graphical user interface, and select the menu item "Robot → Configure Robot Bluetooth".
`connectWithAddress()`	Connect to a robot module by specifying its Bluetooth address.
`disconnect()`	Disconnect from a robot module.
`getJointAngle()`	Gets a joint's angle.
`getJointMaxSpeed()`	Gets a joint's maximum speed in radians per second.
`getJointSpeed()`	Gets a motor's current speed setting in radians per second.
`getJointSpeeds()`	Gets all motor's current speed settings in radians per second.
`getJointSpeedRatio()`	Gets a motor's speed as a ratio of the motor's maximum speed.
`getJointSpeedRatios()`	Gets all motor speeds as ratios of the motor's maximum speed.
`getJointState()`	Gets a motor's current status.
`isConnected()`	This function is used to check the connection to a robot.
`isMoving()`	This function is used to check if any joints are currently in motion.
`move()`	Move all four joints of the robot by specified angles.
`moveNB()`	Identical to `move()` but non-blocking.
`moveContinuousNB()`	Move joints continuously. Joints will move untill stopped.
`moveContinuousTime()`	Move joints continuously for a certain amount of time.
`moveTo()`	Move all four joints of the robot to specified absolute angles.
`moveToNB()`	Identical to `moveTo()` but non-blocking.
`moveJointTo()`	Set the desired motor position.
`moveJointToNB()`	Identical to `moveJointTo()` but non-blocking.
`moveJointWait()`	Wait until the specified motor has stopped moving.
`moveWait()`	Wait until all motors have stopped moving.
`moveToZero()`	Instructs all motors to go to their zero positions.
`moveToZeroNB()`	Identical to `moveToZero()` but non-blocking.
`setJointSpeed()`	Sets a motor's speed setting in radians per second.
`setJointSpeeds()`	Sets all motor speeds in radians per second.
`setJointSpeedRatio()`	Set a joints speed setting to a fraction of its maximum speed, a value between 0 and 1.
`setJointSpeedRatios()`	Set all joint speed settings to a fraction of its maximum speed, expressed as a value from 0 to 1.
`stop()`	Stop all currently executing motions of the robot.

Table 2: CMobot Member Functions for Compound Motions.

Compound Motions	These are convenience functions of commonly used compound motions.
`motionInchwormLeft()`	Inchworm gait towards the left.
`motionInchwormLeftNB()`	Identical to `motionInchwormLeft` but non-blocking.
`motionInchwormRight()`	Inchworm gait towards the right.
`motionInchwormRightNB()`	Identical to `motionInchwormRight` but non-blocking.
`motionRollBackward()`	Roll on the faceplates toward the backward direction.
`motionRollBackwardNB()`	Identical to `motionRollBackward()` but non-blocking.
`motionRollForward()` .	Roll on the faceplates forwards.
`motionRollForwardNB()`	Identical to `motionRollForward()` but non-blocking.
`motionStand()`	Stand the robot up on its end.
`motionStandNB()`	Identical to `motionStandNB()` but non-blocking.
`motionTurnLeft()`	Rotate the robot counterclockwise.
`motionTurnLeftNB()` ..	Identical to `motionTurnLeft()` but non-blocking.
`motionTurnRight()` ...	Rotate the robot clockwise.
`motionTurnRightNB()` .	Identical to `motionTurnRight()` but non-blocking.
`motionWait()`	Wait for a motion to finish.

CMobot::connect()

Synopsis
```
#include <mobot.h>
int CMobot::connect();
```

Purpose
Connect to a remote robot via Bluetooth.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function is used to connect to a robot. The function looks inside of a Barobo configuration file and connects to the first robot listed in the file. The configuration file may be created and/or modified using the Robot Controller Interface, and selecting the "Robot → Configure Robot Bluetooth" menu item.
Example
Please see the example in Section 4.0.2 on page 14.

See Also
`connectWithAddress(), disconnect()`

CMobot::connectWithAddress()

Synopsis
```
#include <mobot.h>
int CMobot::connectWithAddress(char address[], int channel);
```

Purpose
Connect to a remote robot via Bluetooth by specifying the specific Bluetooth address of the device.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
`address` The Bluetooth address of the robot.
`channel` The Bluetooth channel that the listening program is listening on. The default channel is channel 1.

Description
This function is used to connect to a robot.
Example

See Also
`connect(), disconnect()`

CMobot::disconnect()

Synopsis

```
#include <mobot.h>
int CMobot::disconnect();
```

Purpose
Disconnect from a remote robot.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function is used from disconnect to a robot. A call to this function is not necessary before the termination of a program. It is only necessary if another connection will be established within the same program at a later time.

Example

See Also
`connect()`, `connectWithAddress()`

CMobot::getJointAngle()

Synopsis
```
#include <mobot.h>
int CMobot::getJointAngle(robotJointId_t id, double &position);
```

Purpose
Retrieve a robot joint's current angle.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`id`	The joint number. This is an enumerated type discussed in Section A.1 on page 30.
`position`	A variable to store the current position of the robot motor. The contents of this variable will be overwritten with a value that represents the motor's angle in radians.

Description
This function gets the current motor position of a robot's motor. The position returned is in units of degrees and is accurate to roughly ± 0.1 degrees.

Example

See Also

CMobot::getJointMaxSpeed()

Synopsis

```
#include <mobot.h>
int CMobot::getJointMaxSpeed(robotJointId_t id, double &speed);
```

Purpose
Get the maximum speed of a joint on the robot.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`id`	The joint number. This is an enumerated type discussed in Section A.1 on page 30.
`speed`	A variable of type `double`. The value of this variable will be overwritten with the maximum speed setting of the joint, which is in units of radians per second.

Description
This function is used to find the maximum speed setting of a joint. This is the maximum speed at which the joint will accept speed setting from the function `setJointSpeed()`. The values are in units of radians per second.
Example

See Also
`getJointSpeed()`, `getJointMaxSpeedRatio()`, `setJointSpeed()`, `setJointSpeedRatio()`

CMobot::getJointSpeed()

Synopsis
```
#include <mobot.h>
int CMobot::getJointSpeed(robotJointId_t id, double &speed);
```

Purpose
Get the speed of a joint on the robot.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`id`	The joint number to pose. This is an enumerated type discussed in Section A.1 on page 30.
`speed`	A variable of type `double`. The value of this variable will be overwritten with the current speed setting of the joint, which is in units of radians per second.

Description
This function is used to find the current speed setting of a joint. This is the speed at which the joint will move when given motion commands. The values are in units of radians per second.
Example

See Also
`getJointMaxSpeed()`, `getJointSpeedRatio()`, `setJointSpeed()`, `setJointSpeedRatio()`

CMobot::getJointSpeedRatio()

Synopsis
```
#include <mobot.h>
int CMobot::getJointSpeedRatio(robotJointId_t id, double &ratio);
```

Purpose
Get the speed ratio settings of a joint on the robot.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
- `id` Retrieve the speed ratio setting of this joint. This is an enumerated type discussed in Section A.1 on page 30.
- `ratio` A variable of type double. The value of this variable will be overwritten with the current speed ratio setting of the joint.

Description
This function is used to find the speed ratio setting of a joint. The speed ratio setting of a joint is the percentage of the maximum joint speed, and the value ranges from 0 to 1. In other words, if the ratio is set to 0.5, the joint will turn at 50% of its maximum angular velocity while moving continuously or moving to a new goal position.

Example

See Also
`setJointSpeeds()`, `getJointSpeedRatio()`, `getJointSpeed()`

CMobot::getJointSpeedRatios()

Synopsis
```
#include <mobot.h>
int CMobot::getJointSpeedRatios(double ratio[4]);
```

Purpose
Get the speed ratio settings of all joints on the robot.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
- `speed` An array of type double. The four elements of the array will be overwritten with the four speed ratio settings of the robot's joints, which are expressed in as a value from 0 to 1.

Description
This function is used to retrieve all four joint speed ratio settings of a robot simultaneously. The speeds are as a value from 0 to 1.

Example

See Also
`setJointSpeeds()`, `getJointSpeedRatios()`, `getJointSpeed()`

CMobot::getJointSpeeds()

Synopsis
```
#include <mobot.h>
int CMobot::getJointSpeeds(double speed[4]);
```

Purpose
Get the speed settings of all joints on the robot.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
`speed` An array of type double. The four elements of the array will be overwritten with the four speed settings of the robot's joints, which are expressed in radians per second.

Description
This function is used to retrieve all four joint speed settings of a robot simultaneously. The speeds are in radians per second.

Example

See Also
`setJointSpeeds()`, `getJointSpeedRatios()`, `getJointSpeed()`

CMobot::getJointState()

Synopsis
```
#include <mobot.h>
int CMobot::getJointState(robotJointId_t id, robotJointState_t &state);
```

Purpose
Determine whether a motor is moving or not.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
`id` The joint number. This is an enumerated type discussed in Section A.1 on page 30.
`state` An integer variable which will be overwritten with the current state of the motor. This is an enumerated type discussed in Section A.2 on page 30.

Description

This function is used to determine the current state of a motor. Valid states are:

Value	Description
`ROBOT_NEUTRAL`	This value indicates ated. The joint is fr
`ROBOT_FORWARD`	This value indicates
`ROBOT_BACKWARD`	This value indicates
`ROBOT_HOLD`	This value indicates holding its current p able.

Example

See Also
`isMoving()`

CMobot::isConnected()

Synopsis

```
#include <mobot.h>
int CMobot::isConnected();
```

Purpose
Check to see if currently connected to a remote robot via Bluetooth.

Return Value
The function returns zero if it is not currently connected to a robot or if an error has occured, or 1 if the robot is connected.

Parameters
None.

Description
This function is used to check if the software is currently connected to a robot.

Example

See Also
`connect()`, `disconnect()`

CMobot::isMoving()

Synopsis
```
#include <mobot.h>
int CMobot::isMoving();
```

Purpose
Check to see if a robot is currently moving any of its joints.

Return Value
This function returns 0 if none of the joints are being driven or if an error has occured, or 1 if any joint is being driven. A value of 1 is equivalent to either a joint state of `ROBOT_FORWARD` or `ROBOT_BACKWARD` from `getJointState()`

Parameters
None.

Description
This function is used to determine if a robot is currently moving any of its joints.

Example

See Also
`getJointState()`

CMobot::motionInchwormLeft()
CMobot::motionInchwormLeftNB()

Synopsis
```
#include <mobot.h>
int CMobot::motionInchwormLeft();
```

```
int CMobot::motionInchwormLeftNB();
```

Purpose
Perform the inch-worm gait to the left.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function causes the robot to perform a single cycle of the inchworm gait to the left. This function comes in two flavors; a blocking and a non blocking version. The function `motionInchwormLeft()` is blocking, and the function will hang until the motion has finished. The alternative function, `motionInchwormLeftNB()` will return immediately, and the motion will execute asynchronously.

See Also
`motionInchwormRight()`

CMobot::motionInchwormRight()
CMobot::motionInchwormRightNB()

Synopsis
```
#include <mobot.h>
int CMobot::motionInchwormRight();
int CMobot::motionInchwormRightNB();
```

Purpose
Perform the inch-worm gait to the right.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function causes the robot to perform a single cycle of the inchworm gait to the right.

This function has both a blocking and non-blocking version. The blocking version, `motionInchwormRight()`, will block until the robot motion has completed. The non-blocking version, `motionInchwormRightNB()`, will return immediately, and the motion will be performed asynchronously.

See Also
`motionInchwormLeft()`

CMobot::motionRollBackward()
CMobot::motionRollBackwardNB()

Synopsis

```
#include <mobot.h>
int CMobot::motionRollBackward();
int CMobot::motionRollBackwardNB();
```

Purpose
Use the faceplates as wheels to roll backward.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function causes each of the faceplates to rotate 90 degrees to roll the robot backward.

This function has both a blocking and non-blocking version. The blocking version, `motionRollBackward()`, will block until the robot motion has completed. The non-blocking version, `motionRollBackwardNB()`, will return immediately, and the motion will be performed asynchronously.

See Also
`motionRollForward()`

CMobot::motionRollForward() CMobot::motionRollForwardNB()

Synopsis
```
#include <mobot.h>
int CMobot::motionRollForward();
int CMobot::motionRollForwardNB();
```

Purpose
Use the faceplates as wheels to roll forward.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function causes each of the faceplates to rotate 90 degrees to roll the robot forward.

This function has both a blocking and non-blocking version. The blocking version, `motionRollForward()`, will block until the robot motion has completed. The non-blocking version, `motionRollForwardNB()`, will return immediately, and the motion will be performed asynchronously.

See Also
`motionRollBackward()`

CMobot::motionStand()

CMobot::motionStandNB()

Synopsis
```
#include <mobot.h>
int CMobot::motionStand();
int CMobot::motionStandNB();
```

Purpose
Stand the robot up on a faceplate.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function causes the robot to motionStand up into the camera platform.

This function has both a blocking and non-blocking version. The blocking version, `motionStand()`, will block until the robot motion has completed. The non-blocking version, `motionStandNB()`, will return immediately, and the motion will be performed asynchronously.

See Also

CMobot::motionTurnLeft()
CMobot::motionTurnLeftNB()

Synopsis
```
#include <mobot.h>
int CMobot::motionTurnLeft();
int CMobot::motionTurnLeftNB();
```

Purpose
Rotate the robot using the faceplates as wheels.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function causes the robot to rotate the faceplates in opposite directions to cause the robot to rotate counter-clockwise.

This function has both a blocking and non-blocking version. The blocking version, `motionTurnLeft()`, will block until the robot motion has completed. The non-blocking version, `motionTurnLeftNB()`, will return immediately, and the motion will be performed asynchronously.

See Also
`motionTurnRight()`

CMobot::motionTurnRight()
CMobot::motionTurnRightNB()

Synopsis
```
#include <mobot.h>
int CMobot::motionTurnRight();
int CMobot::motionTurnRightNB();
```

Purpose
Rotate the robot using the faceplates as wheels.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function causes the robot to rotate the faceplates in opposite directions to cause the robot to rotate clockwise.

This function has both a blocking and non-blocking version. The blocking version, `motionTurnRight()`, will block until the robot motion has completed. The non-blocking version, `motionTurnRightNB()`, will return immediately, and the motion will be performed asynchronously.

See Also
`motionTurnLeft()`

CMobot::motionWait()

Synopsis
```
#include <mobot.h>
int CMobot::motionWait();
```

Purpose
Wait for a motion to complete execution.

Return Value
The function returns 0 on success and non-zero otherwise.

Description
This function is used to wait for a motion function to fully complete its cycle. The `CMobot` motion functions are those member functions which begin with "motion" as part of their name, such as `motionInchwormLeft()`.
Example

See Also

CMobot::move()

CMobot::moveNB()

Synopsis

```
#include <mobot.h>
int CMobot::move(double angle1, double angle2, double angle3, double angle4);
int CMobot::moveNB(double angle1, double angle2, double angle3, double angle4);
```

Purpose

Move all of the joints of a robot by specified angles.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

`angle1`	The amount to move joint 1, expressed in radians.
`angle2`	The amount to move joint 2, expressed in radians.
`angle3`	The amount to move joint 3, expressed in radians.
`angle4`	The amount to move joint 4, expressed in radians.

Description

This function moves all of the joints of a robot by the specified number of degrees from their current positions.

The function `move()` is a blocking function, which means that the function will not return until the commanded motion is completed. The function `moveNB()` is the non-blocking version of the `move()` function, which means that the function will return immediately and the physical robot motion will occur asynchronously. For more information on blocking and non-blocking functions, please refer to Section 7 on page 24.

Example

Please see the demo at Section 4.0.2 on page 14.

See Also

CMobot::moveContinuousNB()

Synopsis

```
#include <mobot.h>
int CMobot::moveContinuousNB(
  robotJointState_t dir1,
  robotJointState_t dir2,
  robotJointState_t dir3,
  robotJointState_t dir4);
```

Purpose

Move the joints of a robot continuously in the specified directions.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

Each integer parameter specifies the direction the joint should move. The types are enumerated in `mobot.h` and have the following values:

- ROBOT_NEUTRAL : The joint should not move.
- ROBOT_FORWARD : The joint will begin moving in the positive direction.
- ROBOT_BACKWARD: The joint will begin moving in the negative direction.
- ROBOT_HOLD: The joint will hold its current position.

More documentation about these types may be found at Section A.2 on page 30.

Description
This function causes joints of a robot to begin moving at the previously set speed. The joints will continue moving until the joint hits a joint limit, or the joint is stopped by setting the speed to zero. This function is a non-blocking function.

Example

See Also

CMobot::moveContinuousTime()

Synopsis

```
#include <mobot.h>
int CMobot::moveContinuousTime( robotJointState_t dir1,
                                robotJointState_t dir2,
                                robotJointState_t dir3,
                                robotJointState_t dir4,
                                int msecs);
```

Purpose
Move the joints of a robot continuously in the specified directions.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
Each integer direction parameter specifies the direction the joint should move. The types are enumerated in `mobot.h` and have the following values:

- ROBOT_NEUTRAL : The joint should not move.
- ROBOT_FORWARD : The joint will begin moving in the positive direction.
- ROBOT_BACKWARD: The joint will begin moving in the negative direction.
- ROBOT_HOLD: The joint will hold its current position.

The `msecs` parameter is the time to perform the movement, in milliseconds.

Description
This function causes joints of a robot to begin moving. The joints will continue moving until the joint hits a joint limit, or the time specified in the `msecs` parameter is reached. This function will block until the motion is completed.

Example

See Also

CMobot::moveTo()
CMobot::moveToNB()

Synopsis

```
#include <mobot.h>
int CMobot::moveTo(double angle1, double angle2, double angle3, double angle4);
int CMobot::moveToNB(double angle1, double angle2, double angle3, double angle4);
```

Purpose

Move all of the joints of a robot to the specified positions.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

None.

Description

This function moves all of the joints of a robot to the specified absolute positions.

The function `moveTo()` is a blocking function, which means that the function will not return until the commanded motion is completed. The function `moveToNB()` is the non-blocking version of the `moveTo()` function, which means that the function will return immediately and the physical robot motion will occur asynchronously. For more details on blocking and non-blocking functions, please refer to Section 7 on page 24.

Example

Please see the demo at Section 4.0.2 on page 14.

See Also

CMobot::moveJointTo()
CMobot::moveJointToNB()

Synopsis

```
#include <mobot.h>
int CMobot::moveJointTo(robotJointId_t id, double position);
int CMobot::moveJointToNB(robotJointId_t id, double position);
```

Purpose

Move a joint on the robot to an absolute position.

Return Value

The function returns 0 on success and non-zero otherwise.

Parameters

`id` The joint number to wait for.
`position` The absolute angle to move the motor to.

Description
This function commands the motor to move to a position specified in degrees at the current motor's speed. The current motor speed may be set with the `setJointSpeed()` member function. Please note that if the motor speed is set to zero, the motor will not move after calling the `moveJointTo()` function.

The function `moveJointTo()` is a blocking function, which means that the function will not return until the commanded motion is completed. The function `moveJointToNB()` is the non-blocking version of the `moveJointTo()` function, which means that the function will return immediately and the physical robot motion will occur asynchronously. For more details on blocking and non-blocking functions, please refer to Section 7 on page 24.

Example
Please see the example in Section 4.0.2 on page 14.

See Also
`connectWithAddress()`

CMobot::moveJointWait()

Synopsis
```
#include <mobot.h>
int CMobot::moveJointWait(robotJointId_t id);
```

Purpose
Wait for a joint to stop moving.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
`id` The joint number to wait for.

Description
This function is used to wait for a joint motion to finish. Functions such as `moveJoint()` and `moveJointTo()` do not wait for a joint to finish moving before continuing to allow multiple joints to move at the same time. The `moveJointWait()` function is used to wait for robotic motions to complete.

Please note that if this function is called after a motor has been commanded to turn indefinitely, this function may never return and your program may hang.

Example
Please see the example in Section 4.0.2 on page 14.

See Also
`moveWait()`

CMobot::moveWait()

Synopsis
```
#include <mobot.h>
int CMobot::moveWait();
```

Purpose
Wait for all joints to stop moving.

Return Value
The function returns 0 on success and non-zero otherwise.

Description
This function is used to wait for all joint motions to finish. Functions such as `move()` and `moveTo()` do not wait for a joint to finish moving before continuing to allow multiple joints to move at the same time. The `moveWait()` function is used to wait for robotic motions to complete.

Please note that if this function is called after a motor has been commanded to turn indefinitely, this function may never return and your program may hang.

Example
See the sample program in Section 4.0.2 on page 14.
See Also
`moveWait(), moveJointWait()`

CMobot::moveToZero()
CMobot::moveToZeroNB()

Synopsis
```
#include <mobot.h>
int CMobot::moveToZero();
int CMobot::moveToZeroNB();
```

Purpose
Move all of the joints of a robot to their zero position.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function moves all of the joints of a robot to their zero position. Please note that the function `moveToZeroNB()` is non-blocking and will return immediately. Use this function in conjunction with the `moveWait()` function to block until the movement completes.

The function `moveToZero()` is a blocking function, which means that the function will not return until the commanded motion is completed. The function `moveToZeroNB()` is the non-blocking version of the `moveToZero()` function, which means that the function will return immediately and the physical robot motion will occur asynchronously. For more details on blocking and non-blocking functions, please refer to Section 7 on page 24.

Example
Please see the demo at Section 4.0.2 on page 14.

See Also

CMobot::setJointSpeed()

Synopsis
```
#include <mobot.h>
int CMobot::setJointSpeed(robotJointId_t id, double speed);
```

Purpose
Set the speed of a joint on the robot.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`id`	The joint number to pose.
`speed`	An variable of type `double` for the requested average angular speed in radians per second.

Description
This function is used to set the angular speed of a joint of a robot. The maximum possible angular speed for a particular joint may be obtained by using the function `getJointMaxSpeed()`.
Example

See Also

CMobot::setJointSpeedRatio()

Synopsis
```
#include <mobot.h>
int CMobot::setJointSpeedRatio(robotJointId_t id, double ratio);
```

Purpose
Set the speed ratio settings of a joint on the robot.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`id`	Set the speed ratio setting of this joint. This is an enumerated type discussed in Section A.1 on page 30.
`ratio`	A variable of type double with a value from 0 to 1.

Description
This function is used to set the speed ratio setting of a joint. The speed ratio setting of a joint is the percentage of the maximum joint speed, and the value ranges from 0 to 1. In other words, if the ratio is set to 0.5, the joint will turn at 50% of its maximum angular velocity while moving continuously or moving to a new goal position.

Example

See Also
`setJointSpeeds()`, `setJointSpeedRatio()`, `getJointSpeed()`

CMobot::setJointSpeedRatios()

Synopsis
```
#include <mobot.h>
int CMobot::setJointSpeedRatios(double ratios[4]);
```

Purpose
Set the speed ratio settings of all joints on the robot.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
`ratios` An array of type double. Each element of the array represents the speed ratio for a joint, expressed in a value from 0 to 1.

Description
This function is used to simultaneously set the angular speed ratio settings of all four joints of a robot. The speed ratio is a percentage of the maximum speed of a joint, expressed in a value from 0 to 1.

Example

See Also
`getJointSpeeds(), setJointSpeed(), getJointSpeed()`

CMobot::setJointSpeeds()

Synopsis
```
#include <mobot.h>
int CMobot::setJointSpeeds(double speeds[4]);
```

Purpose
Set the speed settings of all joints on the robot.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
`speeds` An array of type double. Each element of the array represents the speed, expressed in radians per second, to set a joint.

Description
This function is used to simultaneously set the angular speed settings of all four joints of a robot.

Example

See Also
`getJointSpeeds(), setJointSpeed(), getJointSpeed()`

CMobot::setTwoWheelRobotSpeed()

Synopsis

```
#include <mobot.h>
int CMobot::setTwoWheelRobotSpeed(double speed, double radius, char unit[]);
```

Purpose
Roll the robot at a certain speed in a straight line.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`speed` The speed at which to roll the robot. The units used will be the units specified in the `unit` parameter.

`radius` The radius of the wheels attached to the robot. The units of the parameter should match the units provided in the `unit` parameter.

`unit` Specify the units used for the `speed` and `radius` parameters.

speed	radius	unit
cm/s	cm	`"cm"`
m/s	m	`"m"`
inch/s	inch	`"inch"`
foot/s	foot	`"foot"`

Description
This function is used to make a two wheeled robot roll at a certain speed. The desired speed and radius of the wheels is provided and the function will rotate the wheels at the appropriate rate in order to achieve the desired speed.

Example

See Also

CMobot::stop()

Synopsis
```
#include <mobot.h>
int CMobot::stop();
```

Purpose
Stop all current motions on the robot.

Return Value
The function returns 0 on success and non-zero otherwise.

Description
This function stops all currently occuring movements on the robot. Internally, this function simply sets all motor speeds to zero. If it is only required to stop a single motor, use the `setJointSpeed()` function to set the motor's speed to zero.

Example

See Also
`setJointSpeed()`

C CMobotGroup API

The `CMobotGroup` class is used to control multiple modules simultaneously. The member functions of the `CMobotGroup` class closely mimic those of the `CMobot` group. The main difference is that the member functions of the `CMobot` class affect a single robot, whereas the member functions of the `CMobotGroup` class move and affect a group of many robots.

Table 3: CMobotGroup Member Functions.

Function	Description
`CMobotGroup()`	The CMobotGroup constructor function. This function is called automatically and should not be called explicitly.
`~CMobotGroup()`	The CMobotGroup destructor function. This function is called automatically and should not be called explicitly.
`addRobot()`	Add a robot to be a member of the robot group.
`move()`	Move all four joints of the robots by specified angles.
`moveNB()`	Identical to `move()` but non-blocking.
`moveContinuousNB()`	Move joints continuously. Joints will move untill stopped.
`moveContinuousTime()`	Move joints continuously for a certain amount of time.
`moveJointContinuousNB()`	Move a single joint on all robots continuously.
`moveJointContinuousTime()`	Move a single joint on all robots continuously for a specific amount of time.
`moveTo()`	Move all four joints of the robots to specified absolute angles.
`moveToNB()`	Identical to `moveTo()` but non-blocking.
`moveJointTo()`	Set the desired motor position.
`moveJointToNB()`	Identical to `moveJointTo()` but non-blocking.
`moveJointWait()`	Wait until the specified motor has stopped moving.
`moveWait()`	Wait until all motors have stopped moving.
`moveToZero()`	Instructs all motors to go to their zero positions.
`moveToZeroNB()`	Identical to `moveToZero()` but non-blocking.
`setJointSpeed()`	Sets a motor's speed setting in radians per second.
`setJointSpeeds()`	Sets all motor speeds in radians per second.
`setJointSpeedRatio()`	Set a joints speed setting to a fraction of its maximum speed, a value between 0 and 1.
`setJointSpeedRatios()`	Set all joint speed settings to a fraction of its maximum speed, expressed as a value from 0 to 1.
`setTwoWheelRobotSpeed()`	Moves the robot at a constant forward velocity.
`stop()`	Stop all currently executing motions of the robot.

Table 4: CMobot Member Functions for Compound Motions.

Compound Motions	These are convenience functions of commonly used compound motions.
`motionInchwormLeft()`	Inchworm gait towards the left.
`motionInchwormLeftNB()`	Identical to `motionInchwormLeft` but non-blocking.
`motionInchwormRight()`	Inchworm gait towards the right.
`motionInchwormRightNB()`	Identical to `motionInchwormRight` but non-blocking.
`motionRollBackward()`	Roll on the faceplates toward the backward direction.
`motionRollBackwardNB()`	Identical to `motionRollBackward()` but non-blocking.
`motionRollForward()` .	Roll on the faceplates forwards.
`motionRollForwardNB()`	Identical to `motionRollForward()` but non-blocking.
`motionStand()`	Stand the robot up on its end.
`motionStandNB()`	Identical to `motionStandNB()` but non-blocking.
`motionTurnLeft()`	Rotate the robot counterclockwise.
`motionTurnLeftNB()` ..	Identical to `motionTurnLeft()` but non-blocking.
`motionTurnRight()` ...	Rotate the robot clockwise.
`motionTurnRightNB()` .	Identical to `motionTurnRight()` but non-blocking.

CMobotGroup::addRobot()

Synopsis
```
#include <mobot.h>
int CMobotGroup::addRobot(CMobot &robot);
```

Purpose
Add a robot to a robot group.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
A robot handle attached to the robot to add to the group.

Description
This function is used to add a robot to a robot group.
Example

See Also

CMobotGroup::motionInchwormLeft()
CMobotGroup::motionInchwormLeftNB()

Synopsis
```
#include <mobot.h>
int CMobotGroup::motionInchwormLeft();
int CMobotGroup::motionInchwormLeftNB();
```

Purpose
Make all robots in the group perform the inch-worm gait to the left.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function causes the robots to perform a single cycle of the inchworm gait to the left. This function comes in two flavors; a blocking and a non blocking version. The function `motionInchwormLeft()` is blocking, and the function will hang until the motion has finished. The alternative function, `motionInchwormLeftNB()` will return immediately, and the motion will execute asynchronously.

See Also
`motionInchwormRight()`

CMobotGroup::motionInchwormRight()

CMobotGroup::motionInchwormRightNB()

Synopsis
```
#include <mobot.h>
int CMobotGroup::motionInchwormRight();
int CMobotGroup::motionInchwormRightNB();
```

Purpose
Make all the robots in the group perform the inch-worm gait to the right.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function causes the robots to perform a single cycle of the inchworm gait to the right.

This function has both a blocking and non-blocking version. The blocking version, `motionInchwormRight()`, will block until the robot motion has completed. The non-blocking version, `motionInchwormRightNB()`, will return immediately, and the motion will be performed asynchronously.

See Also
`motionInchwormLeft()`

CMobotGroup::motionRollBackward()
CMobotGroup::motionRollBackwardNB()

Synopsis
```
#include <mobot.h>
int CMobotGroup::motionRollBackward();
int CMobotGroup::motionRollBackwardNB();
```

Purpose
Use the faceplates as wheels to roll all the robots in a group backward.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function causes each of the faceplates to rotate 90 degrees to roll the robots backward.

This function has both a blocking and non-blocking version. The blocking version, `motionRollBackward()`, will block until the robot motion has completed. The non-blocking version, `motionRollBackwardNB()`, will return immediately, and the motion will be performed asynchronously.

See Also
`motionRollForward()`

CMobotGroup::motionRollForward()
CMobotGroup::motionRollForwardNB()

Synopsis
```
#include <mobot.h>
int CMobotGroup::motionRollForward();
int CMobotGroup::motionRollForwardNB();
```

Purpose
Use the faceplates as wheels to roll robots forward.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function causes each of the faceplates to rotate 90 degrees to roll the robots forward.

This function has both a blocking and non-blocking version. The blocking version, `motionRollForward()`, will block until the robot motion has completed. The non-blocking version, `motionRollForwardNB()`, will return immediately, and the motion will be performed asynchronously.

See Also
`motionRollBackward()`

CMobotGroup::motionStand()
CMobotGroup::motionStandNB()

Synopsis
```
#include <mobot.h>
int CMobotGroup::motionStand();
int CMobotGroup::motionStandNB();
```

Purpose
Stand robots up on a faceplate.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function causes the robots to stand up into the camera platform.

This function has both a blocking and non-blocking version. The blocking version, `motionStand()`, will block until the robot motion has completed. The non-blocking version, `motionStandNB()`, will return immediately, and the motion will be performed asynchronously.

See Also

CMobotGroup::motionTurnLeft()
CMobotGroup::motionTurnLeftNB()

Synopsis
```
#include <mobot.h>
int CMobotGroup::motionTurnLeft();
int CMobotGroup::motionTurnLeftNB();
```

Purpose
Rotate the robots using the faceplates as wheels.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function causes the robots to rotate the faceplates in opposite directions to cause the robot to rotate counter-clockwise.

This function has both a blocking and non-blocking version. The blocking version, `motionTurnLeft()`, will block until the robot motion has completed. The non-blocking version, `motionTurnLeftNB()`, will return immediately, and the motion will be performed asynchronously.

See Also
`motionTurnRight()`

CMobotGroup::motionTurnRight()
CMobotGroup::motionTurnRightNB()

Synopsis
```
#include <mobot.h>
int CMobotGroup::motionTurnRight();
int CMobotGroup::motionTurnRightNB();
```

Purpose
Rotate the robots using the faceplates as wheels.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function causes the robots to rotate the faceplates in opposite directions to cause the robot to rotate clockwise.

This function has both a blocking and non-blocking version. The blocking version, `motionTurnRight()`, will block until the robot motion has completed. The non-blocking version, `motionTurnRightNB()`, will return immediately, and the motion will be performed asynchronously.

See Also
`motionTurnLeft()`

CMobotGroup::move()
CMobotGroup::moveNB()

Synopsis
```
#include <mobot.h>
int CMobotGroup::move(double angle1, double angle2, double angle3, double angle4);
int CMobotGroup::moveNB(double angle1, double angle2, double angle3, double angle4);
```

Purpose
Move all of the joints of robots in a group by specified angles.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`angle1`	The amount to move joint 1, expressed in radians.
`angle2`	The amount to move joint 2, expressed in radians.
`angle3`	The amount to move joint 3, expressed in radians.
`angle4`	The amount to move joint 4, expressed in radians.

Description
This function moves all of the joints of a robot by the specified number of degrees from their current positions.

The function `move()` is a blocking function, which means that the function will not return until the commanded motion is completed. The function `moveNB()` is the non-blocking version of the `move()` function, which means that the function will return immediately and the physical robot motion will occur asynchronously. For more information on blocking and non-blocking functions, please refer to Section 7 on page 24.

Example

See Also

CMobotGroup::moveContinuousNB()

Synopsis
```
#include <mobot.h>
int CMobotGroup::moveContinuousNB(
  robotJointState_t dir1,
  robotJointState_t dir2,
  robotJointState_t dir3,
  robotJointState_t dir4);
```

Purpose
Move the joints of grouped robots continuously in the specified directions.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
Each integer parameter specifies the direction the joint should move. The types are enumerated in `mobot.h` and have the following values:

- `MOBOT_NEUTRAL` : The joint should not move.
- `MOBOT_FORWARD` : The joint will begin moving in the positive direction.
- `MOBOT_BACKWARD`: The joint will begin moving in the negative direction.

More documentation about these types may be found at Section A.2 on page 30.
Description
This function causes joints of robots to begin moving at the previously set speed. The joints will continue moving until the joint hits a joint limit, or the joint is stopped by setting the speed to zero. This function is a non-blocking function.

Example

See Also

CMobotGroup::moveContinuousTime()

Synopsis
```
#include <mobot.h>
int CMobotGroup::moveContinuousTime( robotJointState_t dir1,
                                     robotJointState_t dir2,
                                     robotJointState_t dir3,
                                     robotJointState_t dir4,
                                     int msecs);
```

Purpose
Move the joints of robots continuously in the specified directions.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
Each integer direction parameter specifies the direction the joint should move. The types are enumerated in `mobot.h` and have the following values:

- `MOBOT_NEUTRAL` : The joint should not move.
- `MOBOT_FORWARD` : The joint will begin moving in the positive direction.
- `MOBOT_BACKWARD`: The joint will begin moving in the negative direction.

The `msecs` parameter is the time to perform the movement, in milliseconds.

Description
This function causes joints of robots to begin moving. The joints will continue moving until the joint hits a joint limit, or the time specified in the `msecs` parameter is reached. This function will block until the motion is completed.

Example

See Also

CMobotGroup::moveTo()
CMobotGroup::moveToNB()

Synopsis
```
#include <mobot.h>
int CMobotGroup::moveTo(double angle1, double angle2, double angle3, double angle4);
int CMobotGroup::moveToNB(double angle1, double angle2, double angle3, double angle4);
```

Purpose
Move all of the joints of robots in the group to the specified positions.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function moves all of the joints of robots in the group to the specified absolute positions.

The function `moveTo()` is a blocking function, which means that the function will not return until the commanded motion is completed. The function `moveToNB()` is the non-blocking version of the `moveTo()` function, which means that the function will return immediately and the physical robot motion will occur asynchronously. For more details on blocking and non-blocking functions, please refer to Section 7 on page 24.

Example
Please see the demo at Section 4.0.2 on page 14.

See Also

CMobotGroup::moveJointTo()
CMobotGroup::moveJointToNB()

Synopsis
```
#include <mobot.h>
int CMobotGroup::moveJointTo(robotJointId_t id, double position);
int CMobotGroup::moveJointToNB(robotJointId_t id, double position);
```

Purpose
Move a joint on robots to an absolute position.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`id`	The joint number to wait for.
`position`	The absolute angle to move the motor to.

Description
This function commands the motor on robots in a group to move to a position specified in degrees at the current motor's speed. The current motor speed may be set with the `setJointSpeed()` member function. Please note that if the motor speed is set to zero, the motor will not move after calling the `moveJointTo()` function.

The function `moveJointTo()` is a blocking function, which means that the function will not return until the commanded motion is completed. The function `moveJointToNB()` is the non-blocking version of the `moveJointTo()` function, which means that the function will return immediately and the physical robot motion will occur asynchronously. For more details on blocking and non-blocking functions, please refer to Section 7 on page 24.

Example
Please see the example in Section 4.0.2 on page 14.

See Also

CMobotGroup::moveJointWait()

Synopsis
```
#include <mobot.h>
int CMobotGroup::moveJointWait(robotJointId_t id);
```

Purpose
Wait for a joint to stop moving on all robots in a group.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`id`	The joint number to wait for.

Description
This function is used to wait for a joint motion to finish. Functions such as `moveJointToNB()` and `moveJointNB()` do not wait for a joint to finish moving before continuing to allow multiple joints to move at the same time. The `moveWait()` or `moveJointWait()` functions are used to wait for robotic motions to complete.

Please note that if this function is called after a motor has been commanded to turn indefinitely, this function may never return and your program may hang.

Example
Please see the example in Section 4.0.2 on page 14.

See Also
`moveWait()`

CMobotGroup::moveWait()

Synopsis
```
#include <mobot.h>
int CMobotGroup::moveWait();
```

Purpose
Wait for all joints of all robots in the group to stop moving.

Return Value
The function returns 0 on success and non-zero otherwise.

Description
This function is used to wait for all joint motions to finish. Functions such as `moveJointToNB()` and `moveJointNB()` do not wait for a joint to finish moving before continuing to allow multiple joints to move at the same time. The `moveWait()` or `moveJointWait()` functions are used to wait for robotic motions to complete.

Please note that if this function is called after a motor has been commanded to turn indefinitely, this function may never return and your program may hang.

Example
See the sample program in Section 4.0.2 on page 14.
See Also
`moveWait(), moveJointWait()`

CMobotGroup::moveToZero()
CMobotGroup::moveToZeroNB()

Synopsis
```
#include <mobot.h>
int CMobotGroup::moveToZero();
int CMobotGroup::moveToZeroNB();
```

Purpose
Move all of the joints of robots in the group to their zero position.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
None.

Description
This function moves all of the joints of robots in the group to their zero position. Please note that this function is non-blocking and will return immediately. Use this function in conjunction with the `moveWait()` function to block until the movement completes.

The function `moveToZero()` is a blocking function, which means that the function will not return until the commanded motion is completed. The function `moveToZeroNB()` is the non-blocking version of the

`moveToZero()` function, which means that the function will return immediately and the physical robot motion will occur asynchronously. For more details on blocking and non-blocking functions, please refer to Section 7 on page 24.

Example
Please see the demo at Section 4.0.2 on page 14.

See Also

CMobotGroup::setJointSpeed()

Synopsis

```
\vspace{-8pt}
#include <mobot.h>
int CMobotGroup::setJointSpeed(robotJointId_t id, double speed);
```

Purpose
Set the speed of a joint on all robots in the group.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`id`	The joint number to pose.
`speed`	An variable of type `double` for the requested average angular speed in radians per second.

Description
This function is used to set the angular speed of a joint of all robots in the group.
Example

See Also

CMobotGroup::setJointSpeedRatio()

Synopsis

```
#include <mobot.h>
int CMobotGroup::setJointSpeedRatio(robotJointId_t id, double ratio);
```

Purpose
Set the speed ratio settings of a joint on all robots in the group.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`id`	Set the speed ratio setting of this joint. This is an enumerated type discussed in Section A.1 on page 30.
`ratio`	A variable of type double with a value from 0 to 1.

Description
This function is used to set the speed ratio setting of a joint for all robots in the group. The speed ratio setting of a joint is the percentage of the maximum joint speed, and the value ranges from 0 to 1. In other words, if the ratio is set to 0.5, the joint will turn at 50% of its maximum angular velocity while moving continuously or moving to a new goal position.

Example

See Also
`setJointSpeeds()`, `setJointSpeedRatio()`

CMobotGroup::setJointSpeedRatios()

Synopsis
```
#include <mobot.h>
int CMobotGroup::setJointSpeedRatios(double ratios[4]);
```

Purpose
Set the speed ratio settings of all joints on the robots in the group.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
`ratios` An array of type double. Each element of the array represents the speed ratio for a joint, expressed in a value from 0 to 1.

Description
This function is used to simultaneously set the angular speed ratio settings of all four joints of a robot for all robots in the group. The speed ratio is a percentage of the maximum speed of a joint, expressed in a value from 0 to 1.

Example

See Also
`getJointSpeeds()`, `setJointSpeed()`, `getJointSpeed()`

CMobotGroup::setJointSpeeds()

Synopsis
```
#include <mobot.h>
int CMobotGroup::setJointSpeeds(double speeds[4]);
```

Purpose
Set the speed settings of all joints on all robot in the group.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters
`speeds` An array of type double. Each element of the array represents the speed, expressed in radians per second, to set a joint.

Description
This function is used to simultaneously set the angular speed settings of all four joints of all robots in the group.

Example

See Also
`setJointSpeed()`

CMobotGroup::setTwoWheelRobotSpeed()

Synopsis
```
#include <mobot.h>
int CMobotGroup::setTwoWheelRobotSpeed(double speed, double radius, char unit[]);
```

Purpose
Roll the robots in the group at a certain speed in a straight line.

Return Value
The function returns 0 on success and non-zero otherwise.

Parameters

`speed` The speed at which to roll the robot. The units used will be the units specified in the `unit` parameter.

`radius` The radius of the wheels attached to the robot. The units of the parameter should match the units provided in the `unit` parameter.

`unit` Specify the units used for the `speed` and `radius` parameters.

speed	radius	unit
cm/s	cm	`"cm"`
m/s	m	`"m"`
inch/s	inch	`"inch"`
foot/s	foot	`"foot"`

Description
This function is used to make a two wheeled robot roll at a certain speed. The desired speed and radius of the wheels is provided and the function will rotate the wheels at the appropriate rate in order to achieve the desired speed.

Example

See Also

CMobotGroup::stop()

Synopsis
```
#include <mobot.h>
int CMobotGroup::stop();
```

Purpose
Stop all current motions on all robot in the group.

Return Value
The function returns 0 on success and non-zero otherwise.

Description
This function stops all currently occuring movements on the robot. Internally, this function simply sets all motor speeds to zero. If it is only required to stop a single motor, use the `setJointSpeed()` function to set the motor's speed to zero.

Example

See Also
`setJointSpeed()`

Index