Introduction

This document will cover how the Robot API Library works and how to utilize it to control robots. This user manual will detail the classes contained in the API and how to use their methods. Directions on how to define a Robot and communicate with it will be covered. Instructions on how to extend the API for new and exciting robot implementations will also detailed.

Prerequisites

Readers are assumed to have a basic understanding of Object Oriented programming concepts. If you need to brush on the topic you can go to Wikipedia for more information.

Class Descriptions

The Robot Class

The Robot class is responsible for passing commands to the Robot's protocol and storing the values returned by the Robot.

Assigning a Protocol

The first thing that you'll want to do with the Robot object is assign it a Protocol object. The Robot object uses this object to interact with the physical Robot.

The protocol is stored in the Protocol property in the Robot Object.

Below is an example of assigning a serial Protocol object to the robot Object.

```csharp
//Create the robot object
Robot robot = new Robot();

//Assign a new Serial Protocol Object
robot.Protocol = new Serial();
```

When a Protocol is assigned the Robot Object will invoke it's Initialize() method and it will set up it's data structures to support the new protocol.

Adding Motors

Motors are essential to any robot. The Robot class can support any number of motors, but is limited to the number of independent motors based on the # of motor channels the protocol supports.

To add a motor to a Robot object you will need to use:

```csharp
AddMotor(int Channel);
```

When adding a motor you need to specify what channel it will be using. The channel the motor is given when it's added will be used when setting it's power.
All motors on the same channel will have the same power and direction.

**Communicating Data with the Robot**

There are 2 ways to get data about the Robot: Properties and Methods. To send data to the robot you must invoke a Method.

Methods will communicate across the protocol to get the most recent values from the physical Robot or to send commands to the Robot. Properties store the last received data from the robot or the last sent value.

```java
Robot robot = new Robot();

//Retrieves new data from the robot
int power = robot.GetChannelPower(0);

//Retrieves the last retrieved data
power = robot.Motors[0];

//Sends a command to the robot
power = 100;
robot.SetMotorChannelPower(0,power);

//Now the property is the sent value
power == robot.Motors[0];
```

**Encoder Readings**

You can read Encoder data from the robot using:

```java
int GetEncoderData(int channel)
```

Where channel is between 0 and the max # of encoder channels the Communications protocol supports. This will also update the EncoderData Property for that channel.

**External Input**

You can read External Input data from the robot using:

```java
int GetExternalInput(int channel)
```

Where channel is between 0 and the max # of External Input channels the Communications protocol supports. This will also update the ExternalInput Property for that channel.

**Analog Inputs**

You can read Analog Input data from the robot using:

```java
float GetAnalogInput(int channel)
```

Where channel is between 0 and the max # of analog channels the Communications protocol supports. This will also update the AnalogPorts Property for that channel.

**Motor Control**

You can control motors using

```java
void SetMotorChannelPower(int channel, int power)
```

**Adding Motors**
Where channel is between 0 and the max # of motor channels the Communications protocol supports and power ranges between -100 and 100; This will also update the Motors Property for that channel.

If you want to get the current power the robot has a channel set at

```c
int GetChannelPower(int channel)
```

Where channel is between 0 and the max # of motor channels the Communications protocol supports This will also update the Motors Property for that channel.

**Drive Enabled**

To enable the drive:

```c
void SetDriveEnabled(bool val)
```

where val is either true, to enabled, or false, to disable!! This will update the DriveEnabled? Property

**External Outputs**

To set External outputs

```c
void SetExternalOutput(int channel,int value)
```

Where channel is between 0 and the max # of External Output channels the Communications protocol supports Value must be between 0 and 255; This will also update the ExternalOutput Property for that channel.

**Heart Beat**

The API needs to send the robot a heartbeat every so often. This lets the robot know that it is still connected and that the PC hasn't crashed. If the robot fails to receive a heartbeat in a specified time, the robot will shut itself down.

There are 2 ways to maintain a heartbeat.

```c
void SendHeartBeat()
SetKeepAlive(bool alive)
```

SetKeepAlive will enable or disable automatic protocol support for heartbeats. The delay between heartbeats can be set by

```c
void SetKeepAliveDelay(int milliseconds)
```

where milliseconds is >=0

SendHeartBeat is a manual way to send a heart beat to the robot. Every time it is invoked it will send one.

**RP10**

The RP10 is an extension of the Robot Class.

""
## Changes

### Motor Control

Motors for the RP10 are arranged in a bay format. Each bay is comprised of Rotation motors and Drive motors. Rotation motors in a bay can have independent speeds but must share the same direction. Drive motors in a bay share direction and speed.

To add a motor to a bay

```csharp
void AddMotorToBay(int bayNum, RP10MotorType type, int channel)
```

where: bayNum is the bay the motor will be added to type is either Drive or Rotation channel is the channel to motor will communicate on

To control the motors

```csharp
void SetDriveMotorPower(int bayNum, int power)
void SetRotationalMotorPower(int bayNum, int channel, int power)
```

Each method will set the correct values for each Drive or rotation motor in the bay.

Using

```csharp
SetMotorChannelPower(int channel, int power)
```

will reroute the command to either the SetRotationMotorPower or SetRotationalMotorPower methods so that the motor bay rules aren't violated.

### Accessing Motor Data

The RP10 stores a list of MotorBays. A Motor bay stores all the motors that are in the bay and the Rotation Direction and Drive Power.

### Making New Robots

There's two ways to create a new robot, inheritance and manual creation.

**Inheritance Example**

```csharp
public class RP10Lab:RP10
{
    public override void Initialize()
    {
        base.Initialize();
        //Motor Bay 1;
        base.AddMotorToBay(1, RP10MotorType.Drive, 2);
        base.AddMotorToBay(1, RP10MotorType.Drive, 3);
        base.AddMotorToBay(1, RP10MotorType.Rotation, 4);
        base.AddMotorToBay(1, RP10MotorType.Rotation, 6);

        //Motor Bay 2;
        base.AddMotorToBay(2, RP10MotorType.Drive, 0);
        base.AddMotorToBay(2, RP10MotorType.Drive, 1);
        base.AddMotorToBay(2, RP10MotorType.Rotation, 5);
    }
}
```
By overriding the internalize method every time we create an RP10Lab object the Robot will be all setup with the motors we need.

Manual Example:

```csharp
Robot robot = new Robot();
Serial protocol = new Serial();
robot.Protocol = protocol;
robot.AddMotor(1);
robot.AddMotor(2);
robot.AddMotor(3);
robot.AddMotor(4);
```

In this example we created a new robot added the protocol to it and then added some motors. This way allows for more dynamic robot definitions. This would be one way of creating a robot from a flat file.