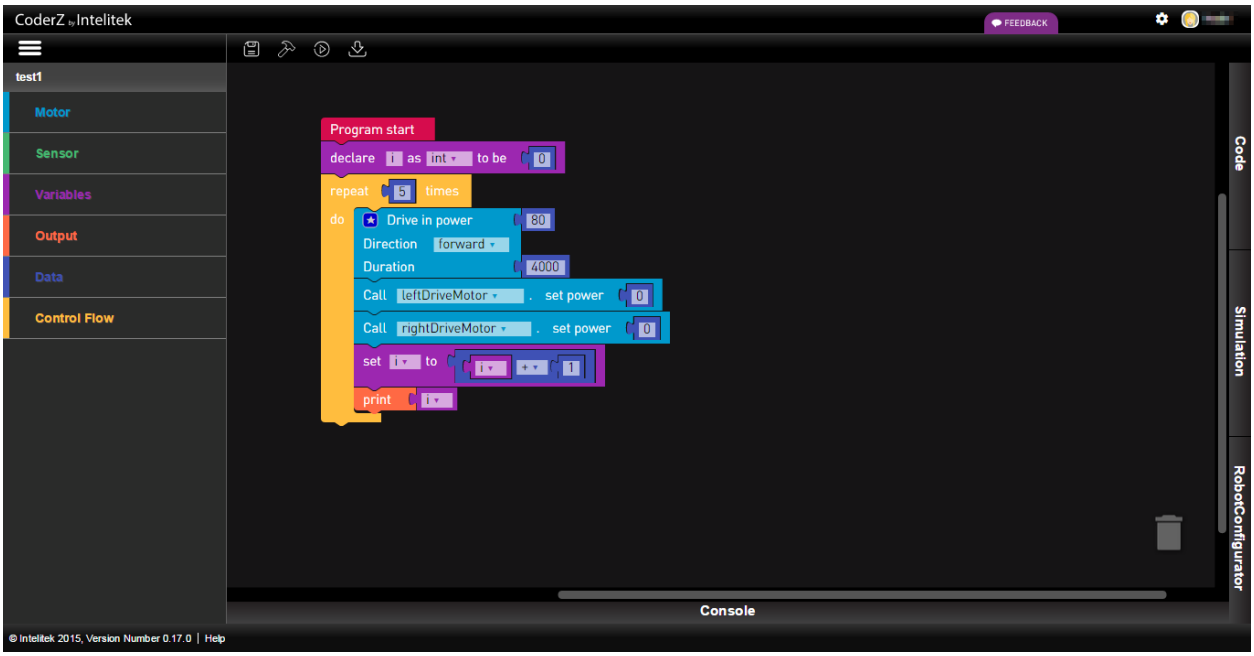
CoderZ User Guide



September 2015

Catalog # Rev

intelitek®

Table of Contents

Intelitek Professional Development User Guide
September 2015
Version
website: http://www.intelitek.com
email: info@intelitek.com
Tel: (603) 625-8600
Fax: (603) 625-2137

1. Introduction

CoderZ is a tool for programming robots using either its Blocks visual programming interface, or via Java or C++.

This document is structured as follows:

Section	Description
1. Introduction	Introduces the document.
2. Users	Explains how to sign up to CoderZ and how to change your profile.
3. Home Interface	Describes the CoderZ home interface.
4. Managing Projects	Describes how to create, open, rename, delete and download projects.
5. Blocks Interface	Describes the Blocks interface, how to write Blocks programs and the function of each block.
6. Code Interface (C++ / Java)	Describes the code interface and lists the C++ library with descriptions of key classes and functions.
7. Project Interface Panes	Describes the Simulation, RobotConfigurator and Console panes that are common for all project interfaces.
8. Downloading a Program to a Robot	Describes the process of downloading a program to a robot.
9. Classes	Describes how to create and manage student classes.

2. Users

In order to use CoderZ you must be a registered user. This section describes the following:

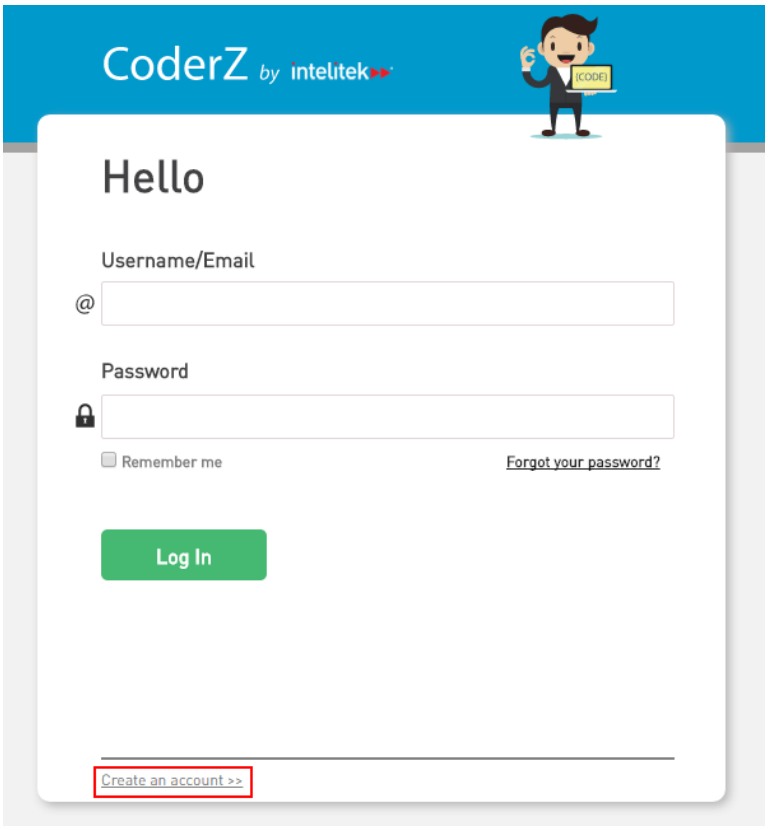
- Signing Up
- Profile

2.1.1. Signing Up

You must be a registered user in order to use CoderZ.

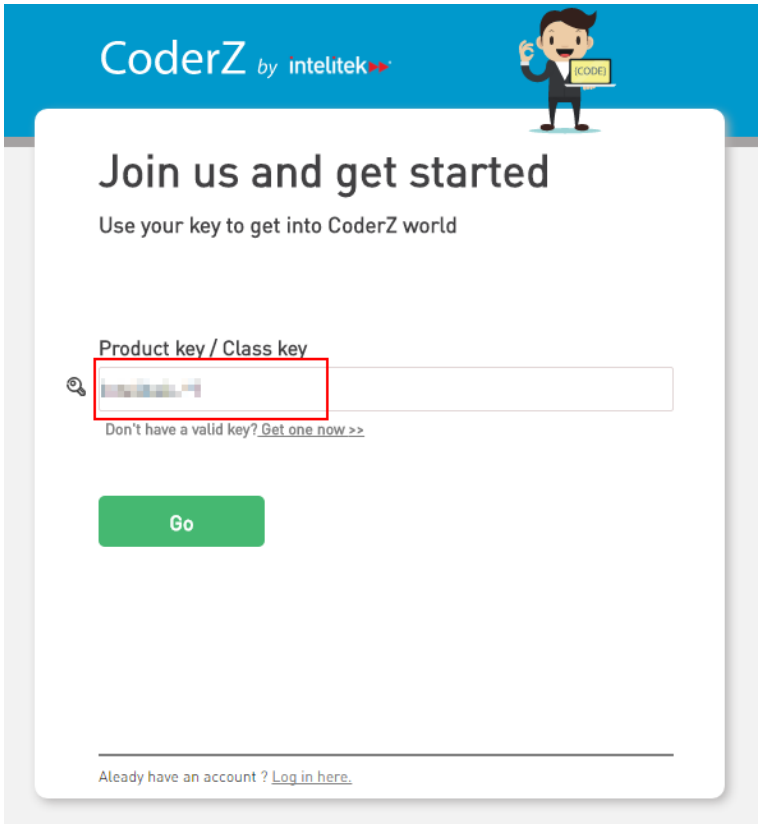
To sign up to CoderZ:

1. Navigate to _____.
2. Click **Create an Account >>**.

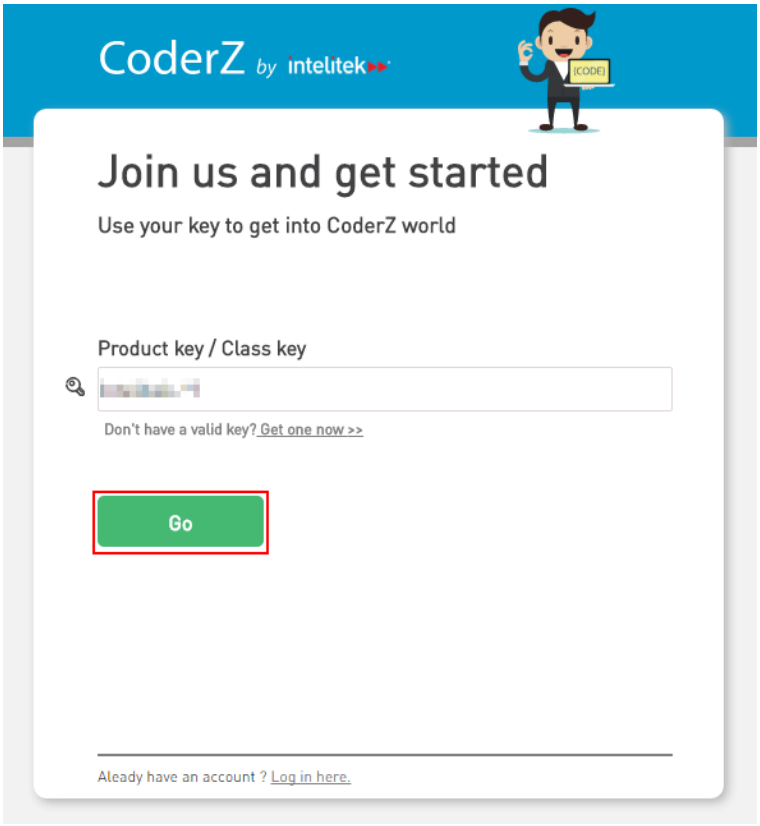


The Product Key / Class Key page appears.

3. Type your CoderZ Product or Class key.

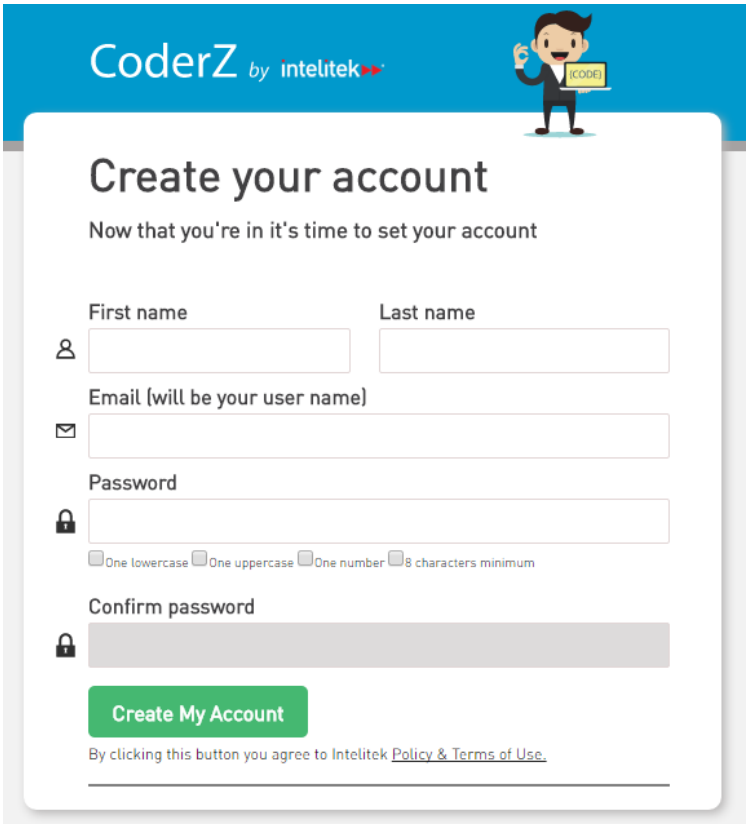


4. Click **Go**.

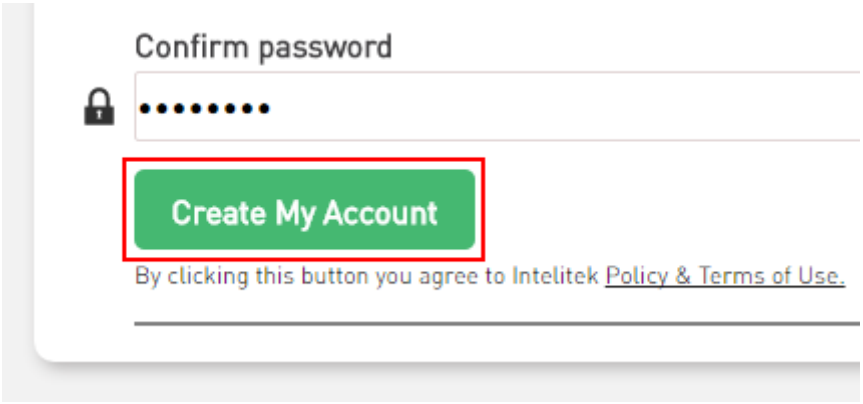


The Signup form appears.

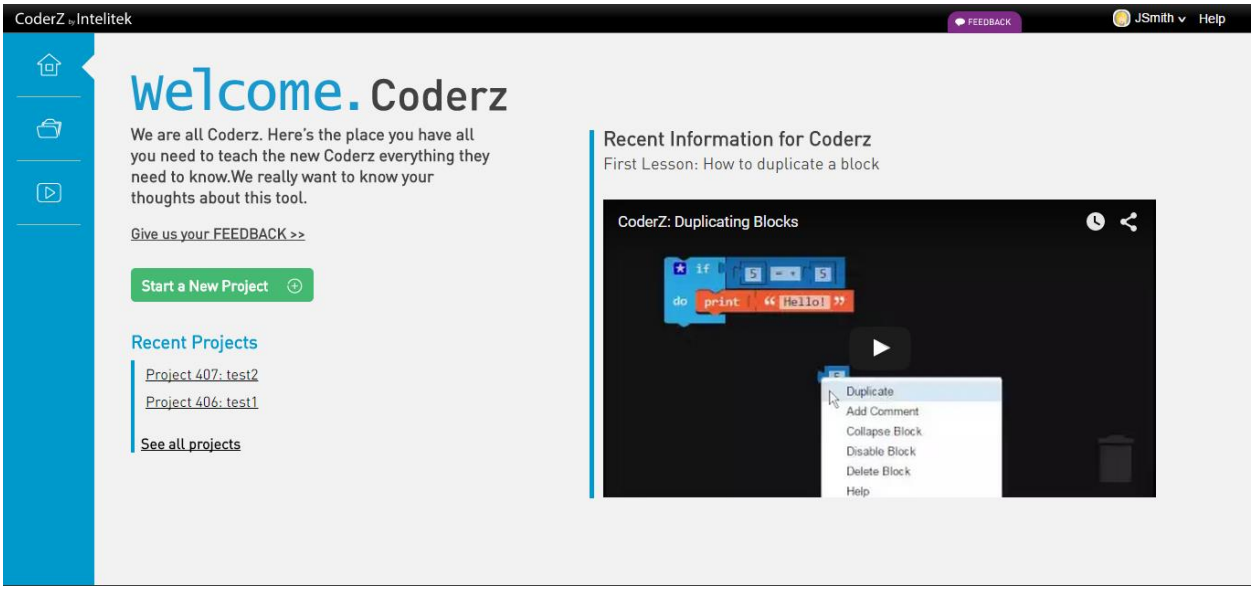
5. Complete the signup form.



6. Click **Submit**.



Your account is created and the Welcome page appears.

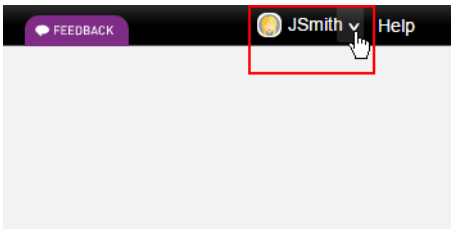


2.1.2. Profile

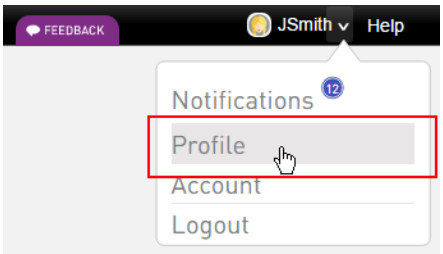
The Profile area offers options for changing your profile settings.

To access the Profile area, on the top-right of the CoderZ window:

1. Click the arrow beside your username at the top-right of the page.



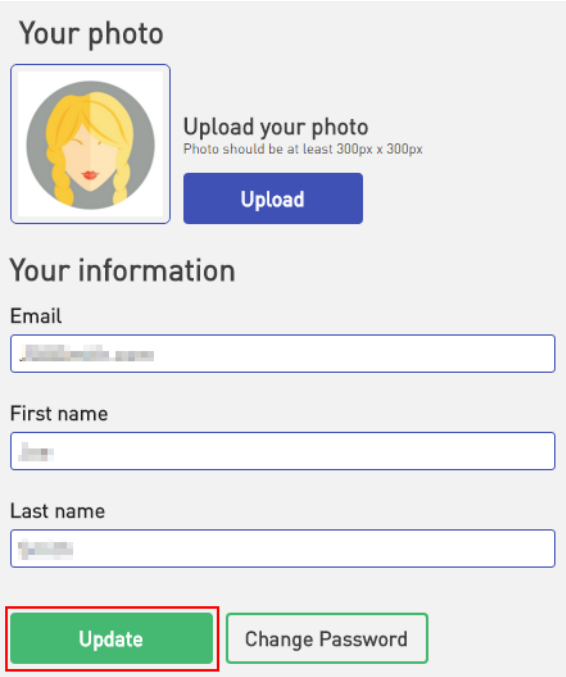
2. Click Profile.



The Profile area is shown.

In the Profile area, you can update your e-mail address, first name, last name and profile picture.

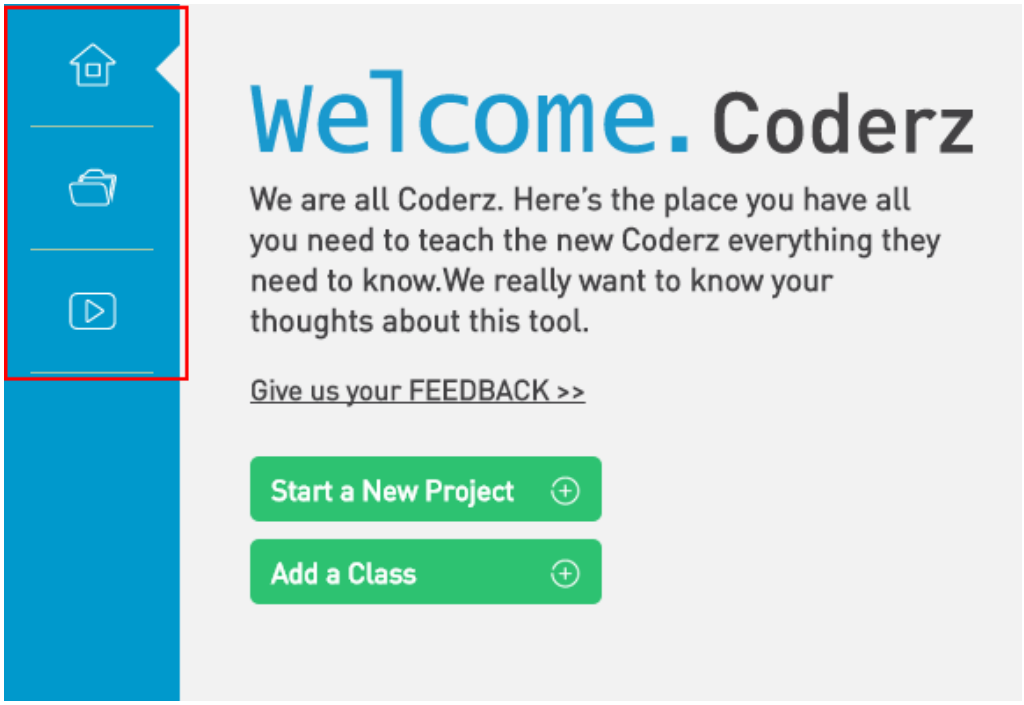
Click **Update** to save changes to your profile.



3. Home Interface

Upon logging in to CoderZ, the Home interface appears.

The top-left of the Home interface has three icons. Click these icons to switch between the three Home interface pages.



The Home interface has the following three pages:

	**Welcome**	The Welcome page contains the following two buttons: 1. Start a New Project. Click this option to open the Project Wizard. For more information, refer to section 4.1 Creating a Project, below. 2. Add a Class. Click this option to open the Class Wizard. For more information, refer to section 9.1 Creating a Class, below.
	**Projects**	The Projects page displays all of your programs. From the Projects page you can: 1. Create a project. For more information, refer to section 4.1 Creating a Project, below. 2. Open a project. For more information, refer to the section 4.2 Opening a Project, below. 3. Download a project. For more information, refer to the section **Error! Reference source not found. Error! Reference source not found.**, **Error! Reference source not found.**.
	**Tutorials**	The Tutorials page contains a number of tutorial videos in ascending order of difficulty. These tutorials are designed to help you get started with using and writing programs in CoderZ.

4. Managing Projects

This section describes the following processes for managing projects in CoderZ:

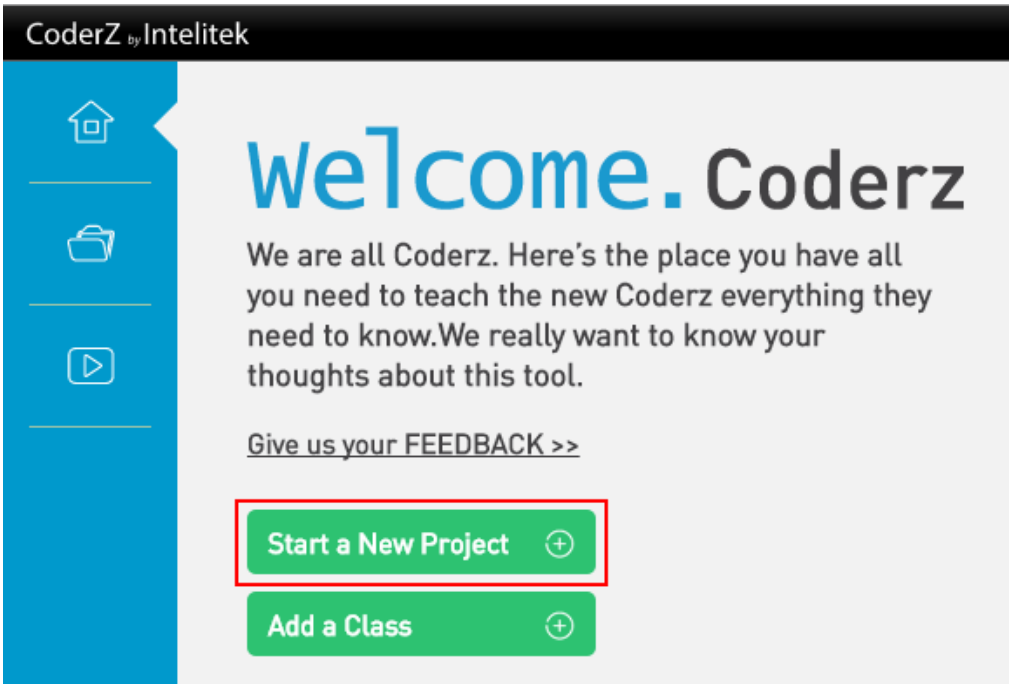
- Creating a Project
- Opening a Project
- Renaming a Project
- Deleting a Project

4.1. CREATING A PROJECT

To create a new CoderZ project:

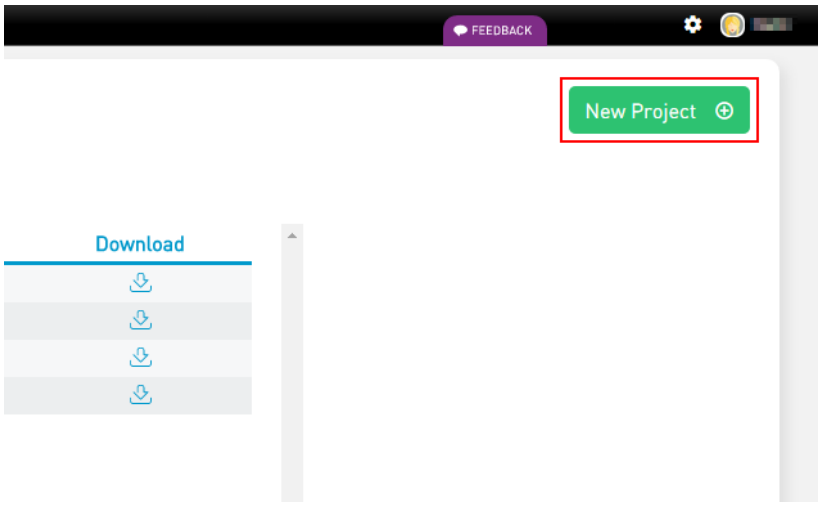
3. Either:

   - On the Welcome page of the Home interface, click **Start a New Project**.



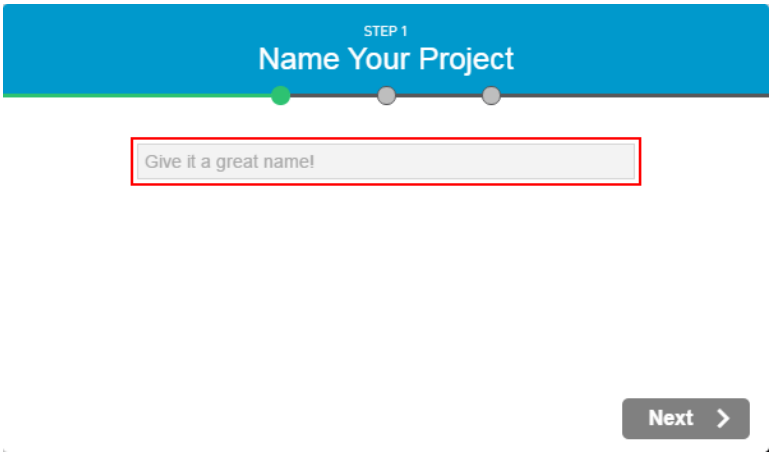
   or

- On the top-right of the Projects page of the Home interface, click **New Project**.

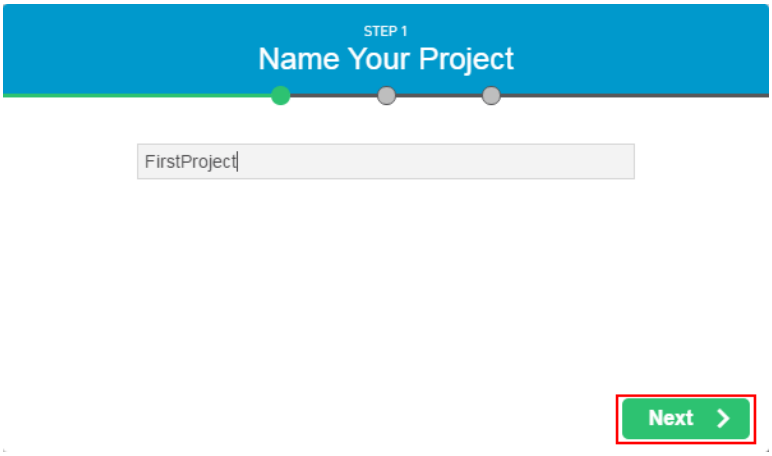


The Project Wizard appears.

4. In the field, name your project.



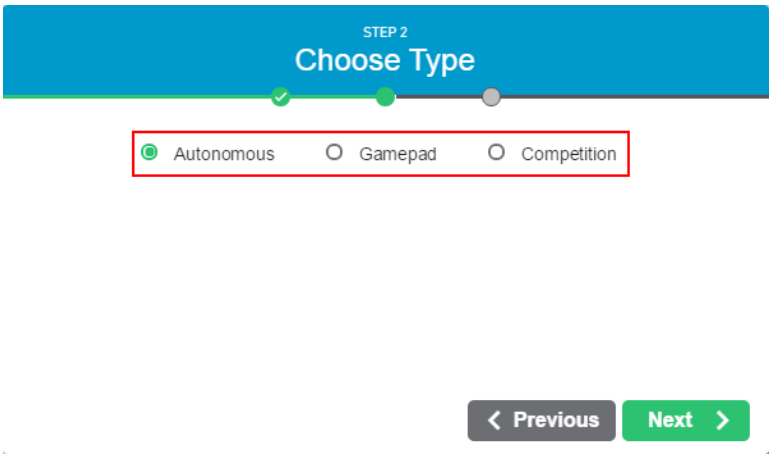
5. Click **Next**.



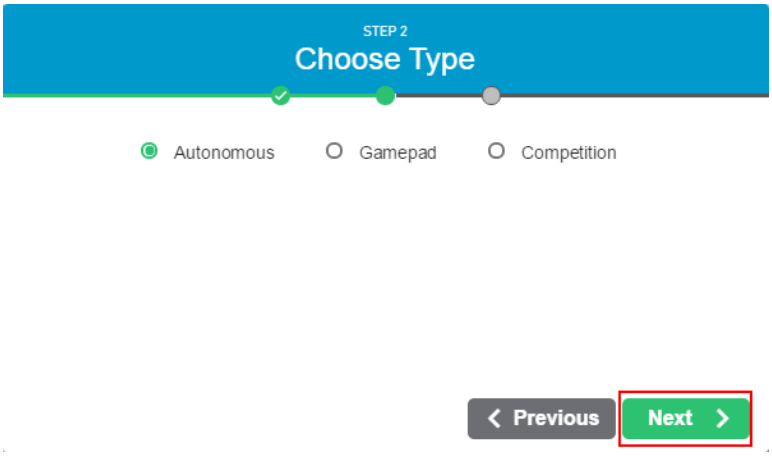
Step 2 of the Project Wizard appears.

6. Select the type of project to create.

   - **Autonomous**: Select Autonomous if you will be writing a program that will control a robot with no human input while it is running.
   - **Gamepad**: Select Gamepad if you will be writing a program that will control a robot using a gamepad.
   - **Competition**: Select Competition if you will be writing a time limited program for use in a competition.



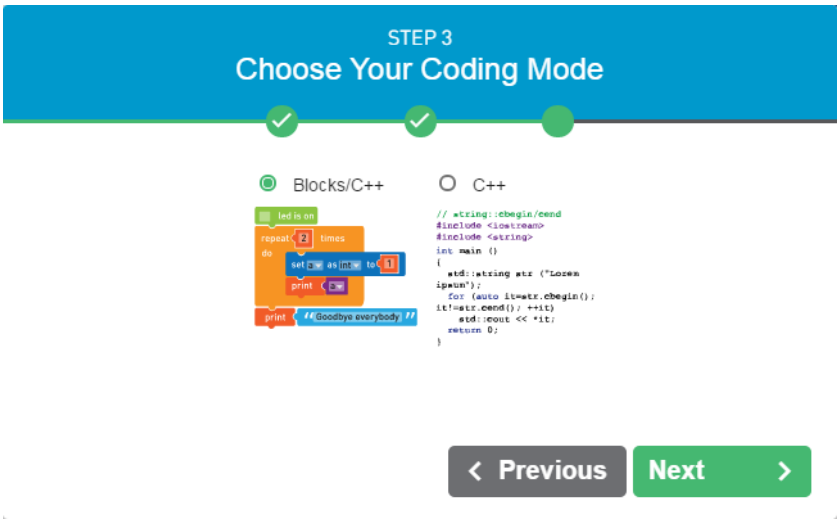
7. Click **Next**.



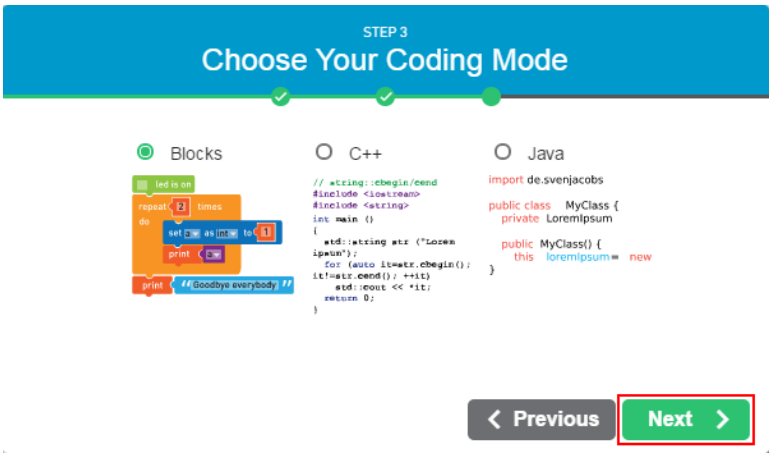
Step 3 of the Project Wizard appears.

8. Select a method for writing a program for the project.

   - **Blocks/C++**: A Blocks program is written by dragging and configuring blocks. You do not need to know or use any programming languages for this method. The code that is written when you add and configure blocks is C++.
   - **C++**: A C++ program is written using the C++ programming language.



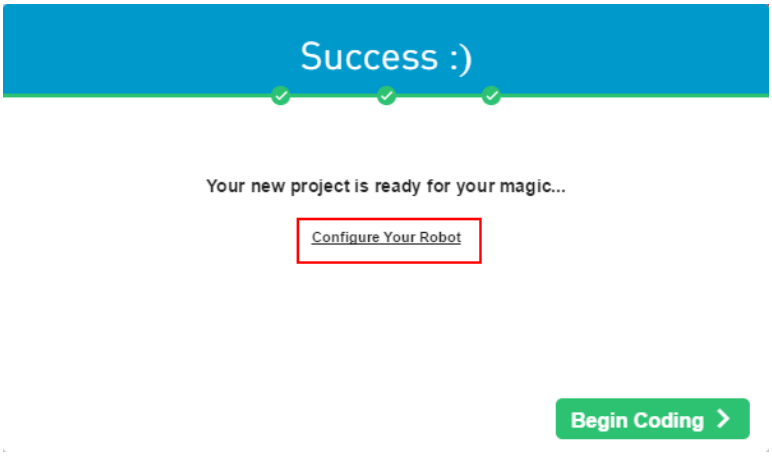
9. Click **Next**.



The Success page of the Project Wizard appears.

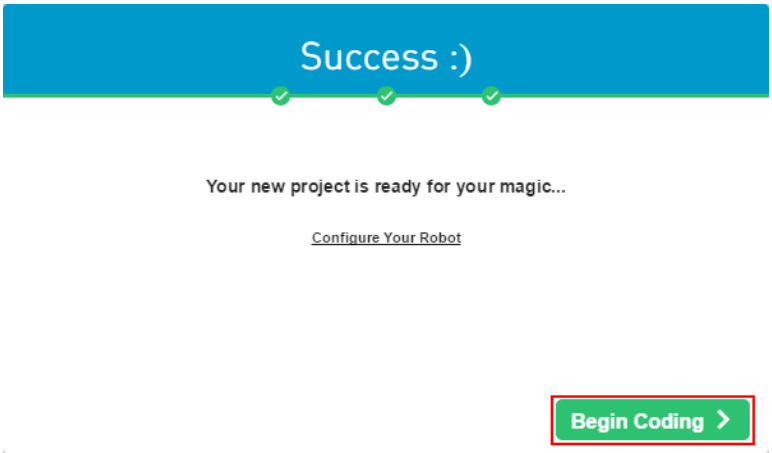
10. The ClawBot robot is the default robot for a project.

- Click **Configure Your Robot** to configure the ports for your robot's motors, servos and sensors. The RobotConfigurator appears. For more information, refer to the RobotConfigurator section below.



or

- Click **Begin Coding** to skip robot configuration and open the project.

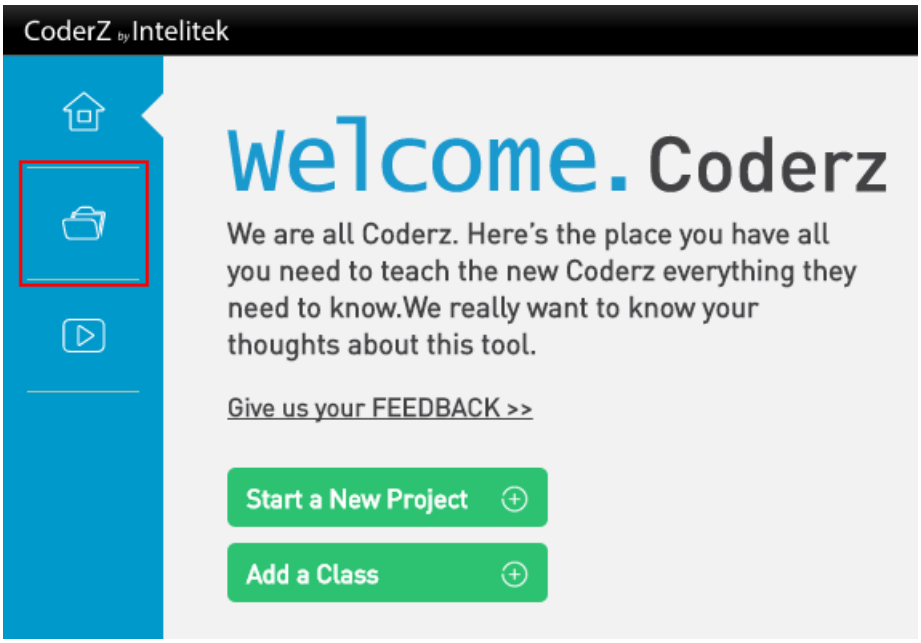


The new project appears.

4.2. OPENING A PROJECT

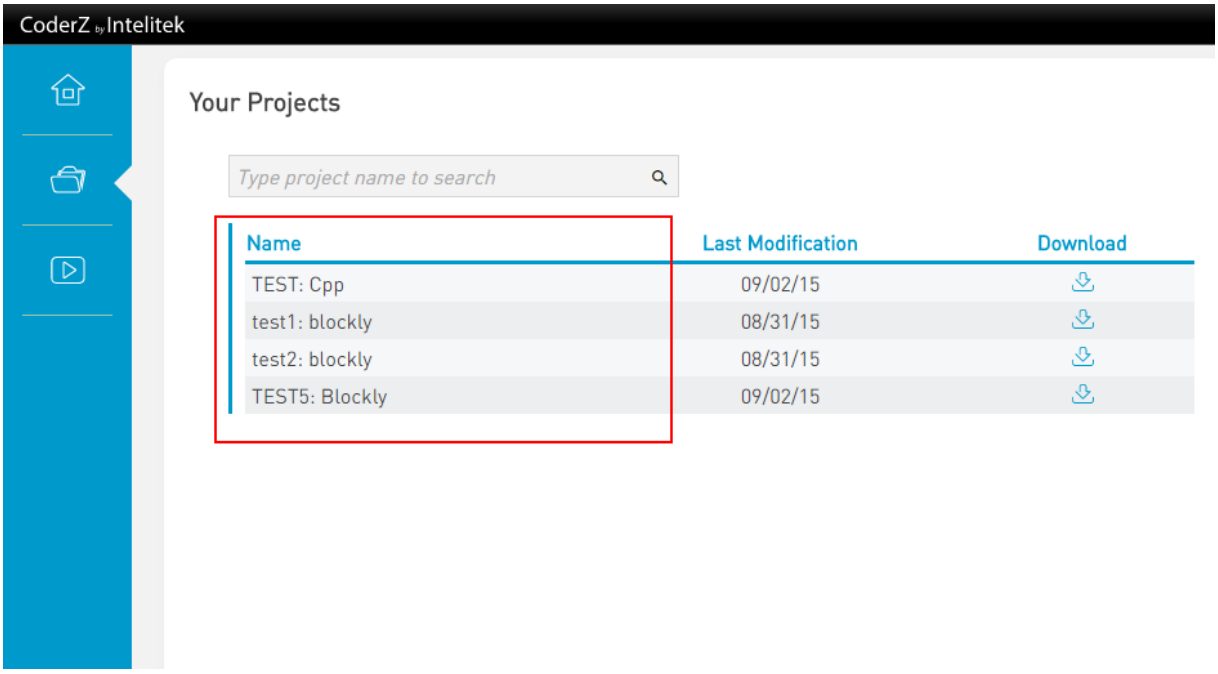
To open an existing project:

1. On the top-right of the Welcome page of the Home interface, click the  Projects icon.



The Projects page appears.

2. From the Your Projects list, click the name of the desired project.

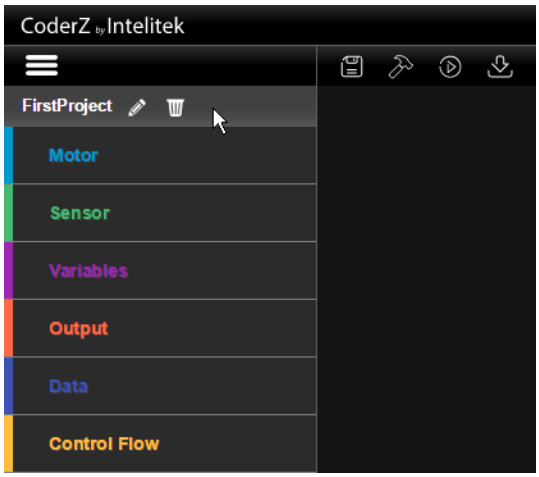


The project is opened.

4.3. RENAMING A PROJECT

To rename a project:

1. If the project you want to rename is not already open, open it as described in section 4.2 Opening a Project, above.
2. On the top-left of the project window, hover over the bar that contains the project name.

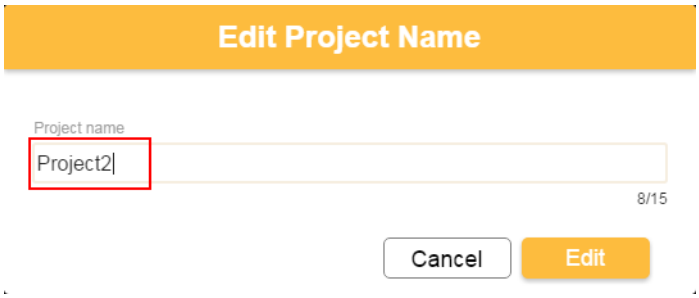


Two icons appear.

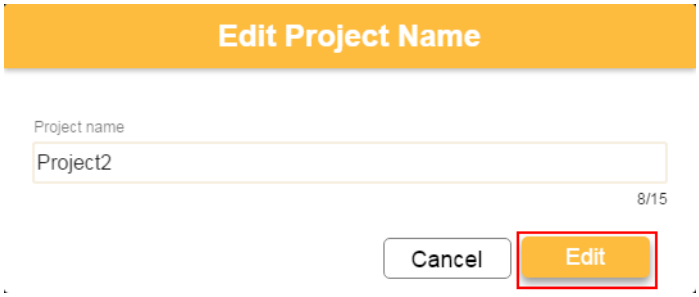
3. Click the pencil icon.

The Edit Project Name dialog box appears.

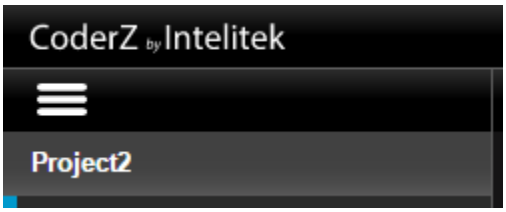
4. In the Project name field, replace the existing project name with another name.



5. Click **Edit**.



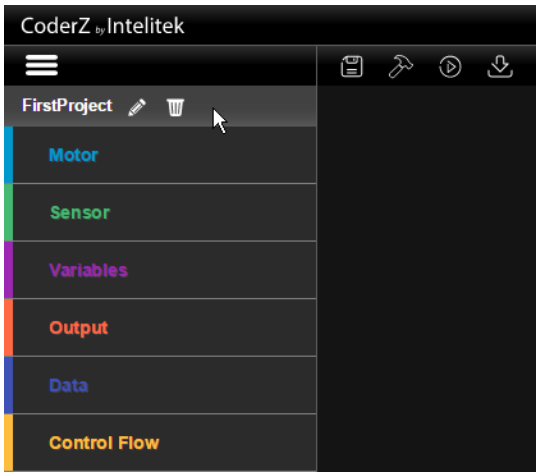
The project is renamed. The new project name appears on the top-left of the project window.



4.4. DELETING A PROJECT

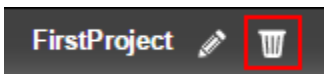
To delete a project:

1. If the project you want to rename is not already open, open it as described in section 4.2 Opening a Project, above.
2. On the top-left of the project window, hover over the bar that contains the project name.



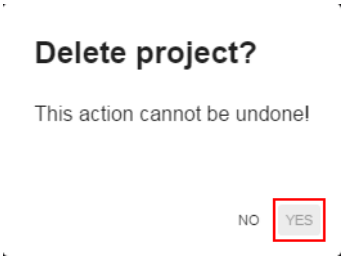
Two icons appear.

3. Click the trashcan icon.



The Delete project? dialog box appears.

4. Click **YES**.



The project is deleted and the Projects page of the Home interface appears.

5. Blocks Interface

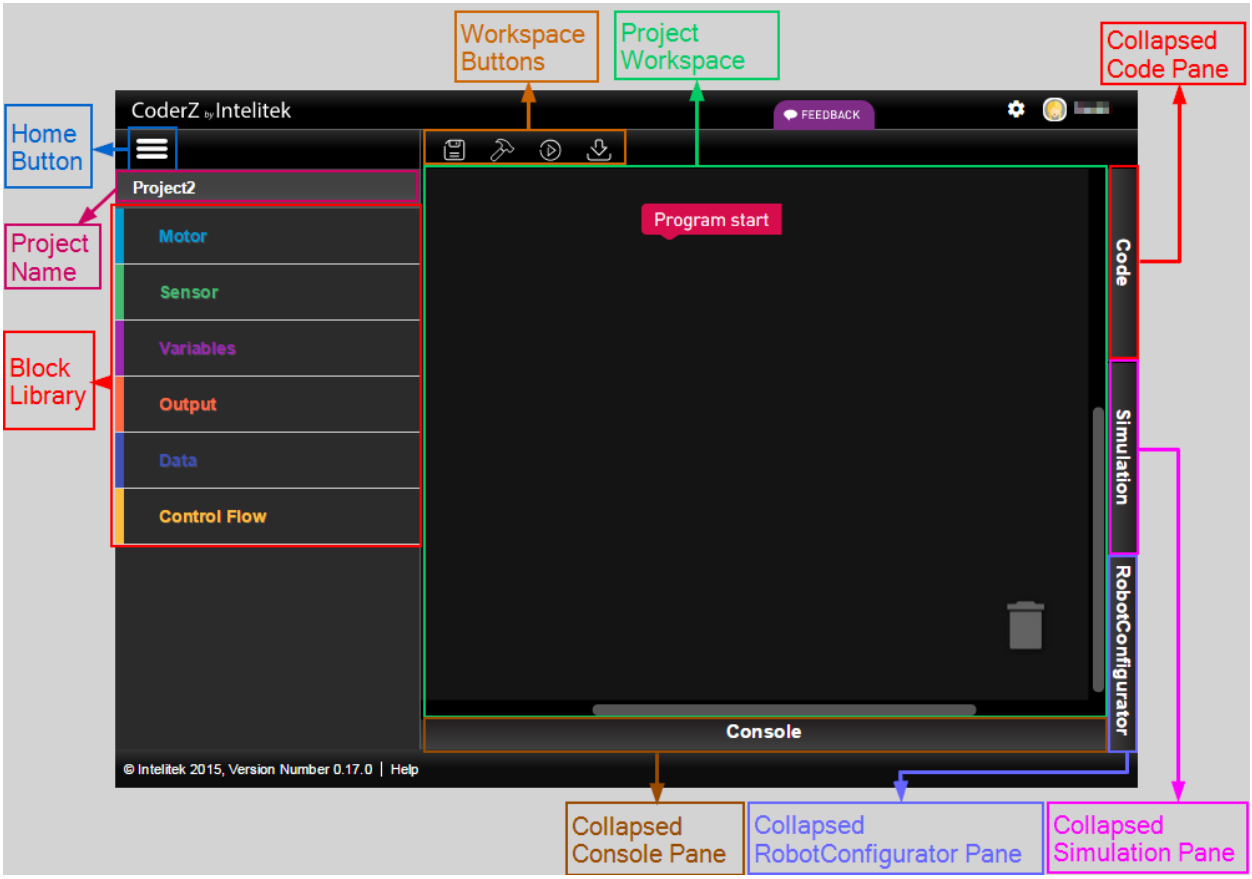
The Blocks interface offers the ability to write visual programs for controlling a robot.

This section introduces the Blocks interface and explains how to write programs using the Blocks method.

This section describes the following:

- Overview
- Adding Blocks
- Configuring Blocks
- Dragging Blocks
- Deleting Blocks
- Comments
- Context Options
- Block Library
- Code Pane

5.1. OVERVIEW



This section describes the elements of the Blocks interface. Refer to the image above for the location of these elements.

The Blocks interface contains the following elements:

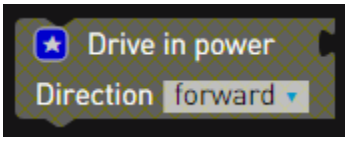
Name	Description
Home Button	Click the home button to return to the Welcome page of the Home interface.
Project Name	Displays the project name. Hover over the Project Name area to display the trashcan and pencil icons for deleting and renaming the project. For more information, refer to section 4.3 Renaming a Project and section 4.4 Deleting a Project above.
Block Library	Contains blocks for writing a program, divided by category. Click a category to display its blocks. Drag blocks from the Blocks Library to the Project Workspace to write your program. For more information, refer to section 5.2 Adding Blocks, below.
Workspace Buttons	The following Workspace buttons are above the Project Workspace:  Save: Saves the open project.  Compile: Compiles the program. After clicking this button, the Console is displayed with information about compilation. If there are problems with the program, details of the problems are written to the console. If there are no problems with the program, a success message is written to the console.  Run Simulation: Compiles the program and runs it in simulation. After clicking this button, the Simulation Pane expands and the simulated robot runs the program.  Download to Robot: Downloads the program to a physical robot. The robot must be connected via a USB cable to the PC. For more information, refer to section Downloading to Robot, below.
Project Workspace	Programs are written in the Project Workspace. Drag blocks from the Blocks Library to the Project Workspace to write your program. For more information on building programs in the Project Workspace, refer to the following sections.
Collapsed Code Pane	Expands the Code Pane. The Code pane displays the C++ code behind the Blocks programs that you write. Any changes made in a Blocks program will dynamically change the C++ code in the Code Pane.
Collapsed Simulation Pane	Expands the Simulation pane. The Simulation Pane contains a simulation of the robot as configured in the RobotConfigurator that can run programs. For more information, refer to section 7.1 Simulation, below.

Name	Description
Collapsed RobotConfigurator Pane	Expands the RobotConfigurator Pane . In this pane you can configure your robot and its ports. For more information, refer to section 7.2 RobotConfigurator, below.
Collapsed Console Pane	Expands the Console Pane. Program information including compilation details and program running details is written to the console. For more information, refer to section 7.3 Console, below.

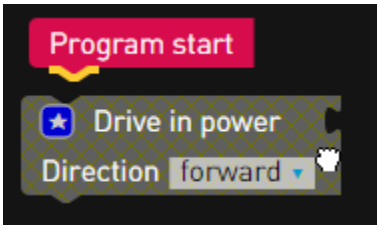
5.2. ADDING BLOCKS

Blocks programs are written by adding blocks to the program in the Program area, and configuring them.

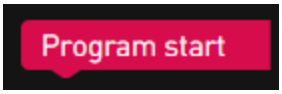
All program blocks must be attached to one another. If a block is not attached to the other blocks in a program, the block will be grayed out, indicating that it is disabled, and will be ignored when running the program.



Blocks have puzzle piece connectors. The shape of each block indicates which type of other block it can be connected to. When the connector of a block is dragged near to the connector of a compatible block, the connector of the block in the program is highlighted, indicating that the dragged block can be released and added to the program.

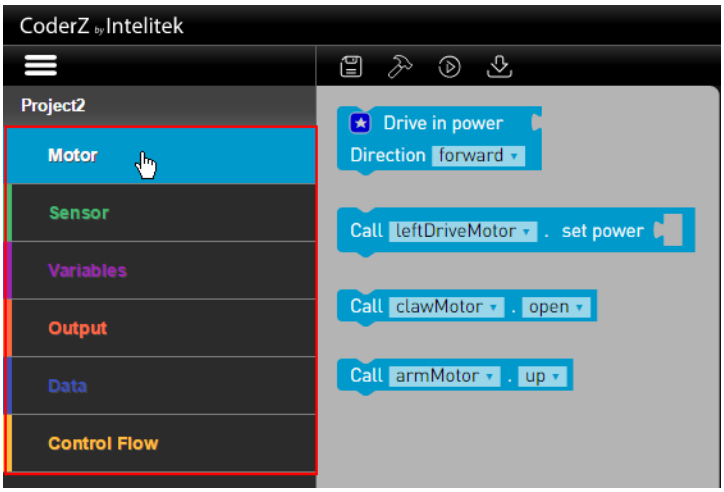


The first block in a program must be attached immediately below the Program start block that is included automatically in a new project.



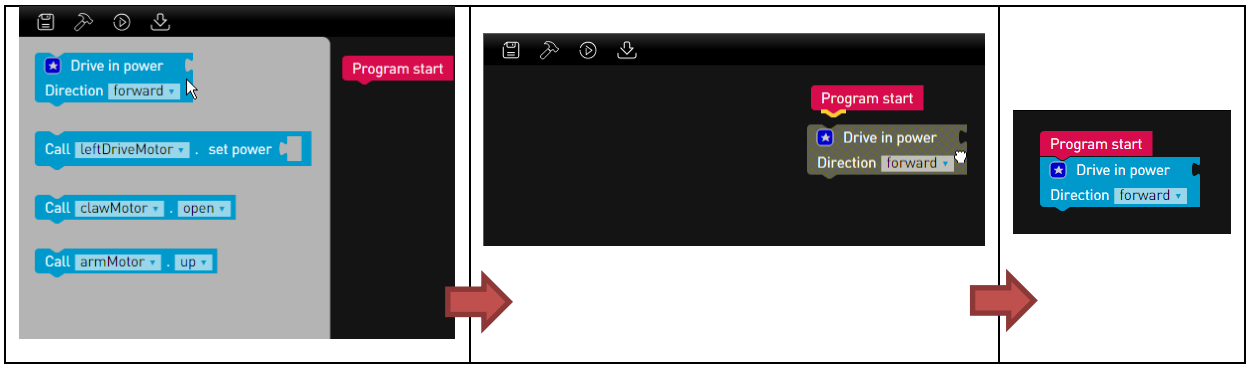
To add a block:

1. In the Block Library, click a Category to expand it.



   A gray extension of the Blocks Library that contains the category's blocks appears.

2. Click and drag a block into the program in the Program Workspace.



   The block must be attached below or beside, or inserted into a compatible block in the program.

3. Release the block when the connector of the block that you are attaching the dragged block to is highlighted, indicating that it is compatible.

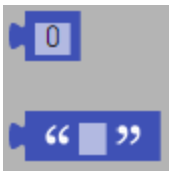
   The block is added to the program.

5.3. CONFIGURING BLOCKS

Many blocks require configuration for them to serve their intended command in the program. Different blocks require different configuration. This section describes the configuration of the different types of blocks.

5.3.1. Text Input

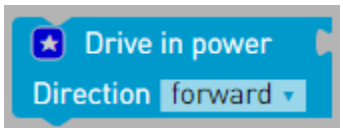
Some blocks require text input. Examples include the number and text blocks shown below.



Click in the field and type a value to add text or a number.

5.3.2. Drop-down Menu

Some blocks contain a drop-down menu, for example the Drive block shown below.



Click the drop-down menu and select an option as required.

5.3.3. Blue Star

There are two blocks that contain a  blue star icon that allow you to add additional parameters to the basic block or remove them.

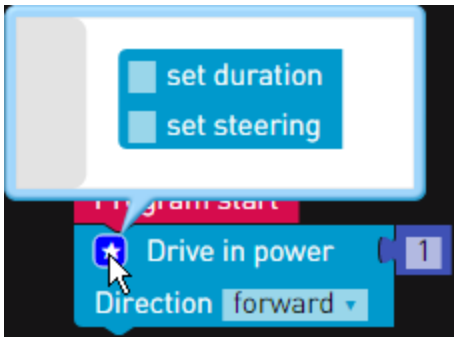
5.3.3.1. Drive Block

The Drive block has a blue star icon that allows you to set the duration or steering for the robot to drive.

To configure the duration or steering of the Drive block:

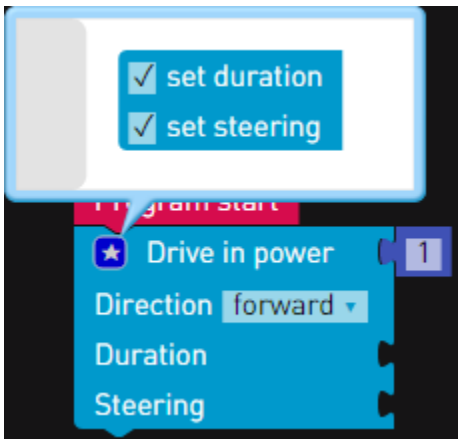
1. Click the Drive block's  blue star icon.

   A pop-up window appears.



2. Select the desired options that you want to add to the block.

   The options are dynamically added to the block.



3. Click the blue star icon to close the pop-up window.

   Repeat steps 1 and 2 and deselect the options to remove them from the block.

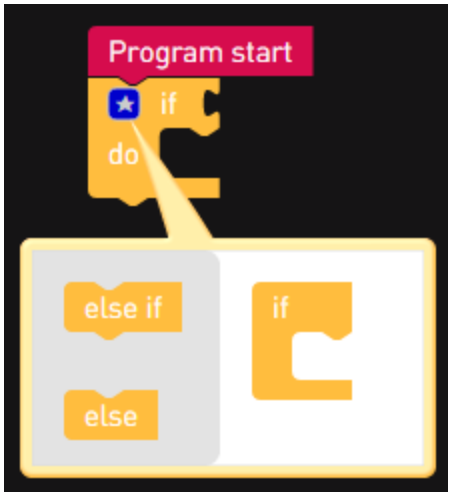
5.3.3.2. If..Do

The If..Do block has a blue star icon that allows you to add else or else if conditions to the block.

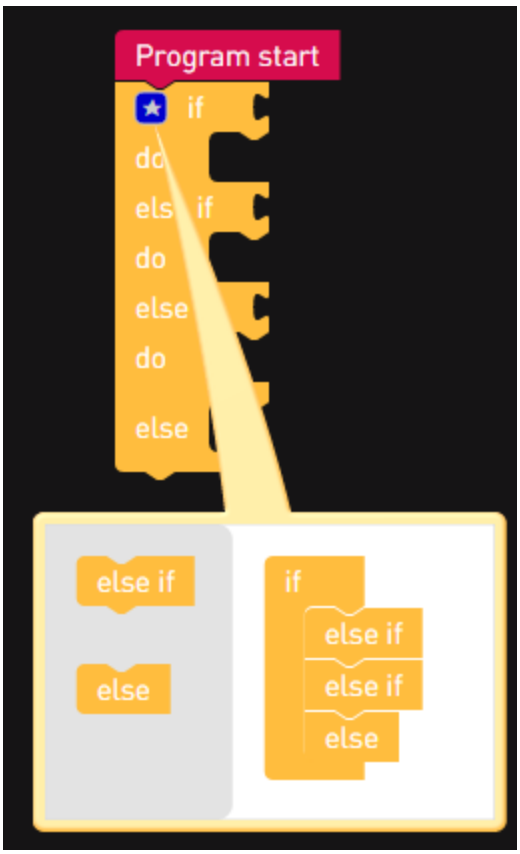
To add conditions to the If..Do block:

1. Click the If..Do block's blue star icon.

   A pop-up window appears.



2. Click either an else if block or an else block in the left gray area of the pop-up window, and drag and release it in place in the block on the right white area of the pop-up window. The conditions are dynamically added to the block. You may add several conditions.



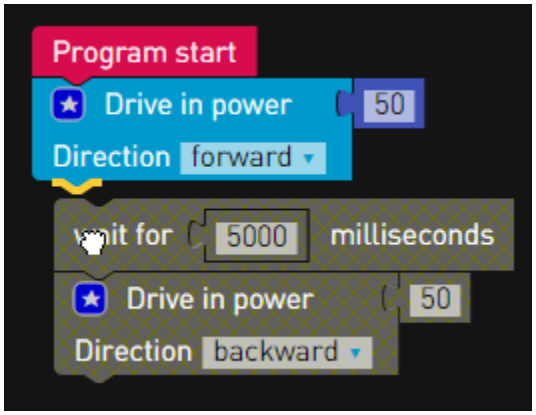
Click an else if or else block from the right white area of the pop-up window and drag and release it in the left gray area of the pop-up window to remove it from the block.

3. Click the blue star icon to close the pop-up window.

5.4. DRAGGING BLOCKS

Click and drag a block to move it within a program.

Clicking and dragging a block that is not the last block will drag all connected blocks that are attached below and beside it.



5.5. DELETING BLOCKS

There are two ways to delete blocks as described in the following subsections.

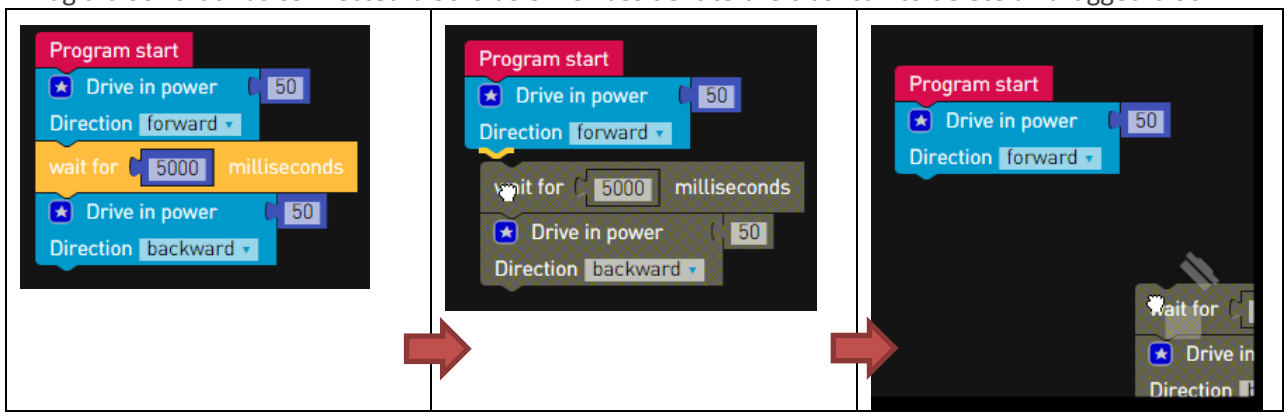
5.5.1. Trashcan

5.5.1.1. Deleting a Single Block

Drag a block onto the trashcan and release it to delete the block.

5.5.1.2. Deleting a Group of Blocks

Drag a block that has connected blocks below or beside it to the trashcan to delete all dragged block.



5.5.2. Context Menu

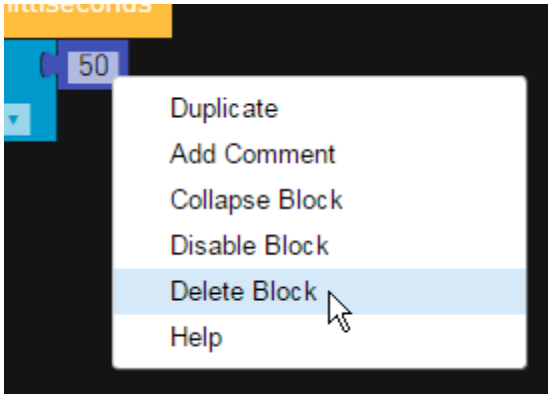
Right-clicking a block opens a context menu, whose options will depend on the block clicked.

5.5.2.1. Deleting a Single Block

To delete a block using its context menu:

1. Right-click a block.

2. Click **Delete Block**.

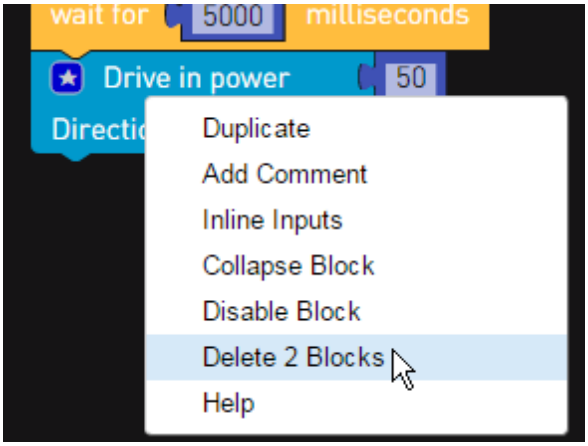


The block is deleted.

5.5.2.2. Deleting a Group of Blocks

To delete a group of blocks using a context menu:

1. Right-click a block that has one or more block connected horizontally.
2. Click Delete 2 blocks.



The group of blocks is deleted.

5.6. COMMENTS

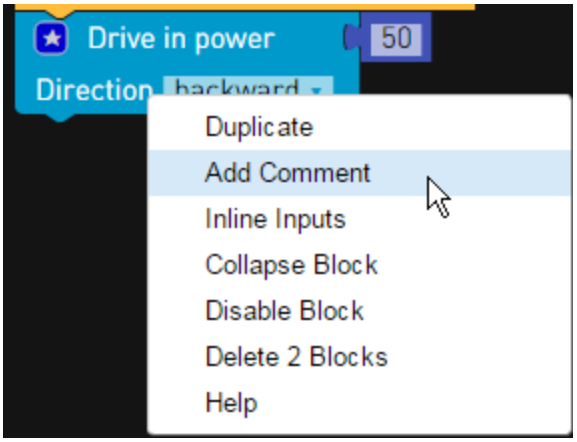
Adding comments to a program is good practice and very useful for explaining the program to others, or reminding yourself of the purpose of how the program works.

5.6.1. Adding a Comment

To add a comment to a block:

1. Right click a block.

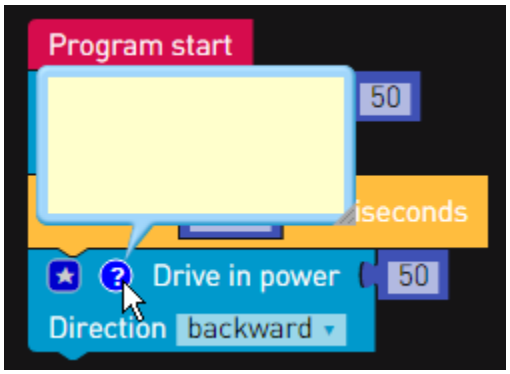
2. Click **Add Comment**.



A Comment icon is added to the block.

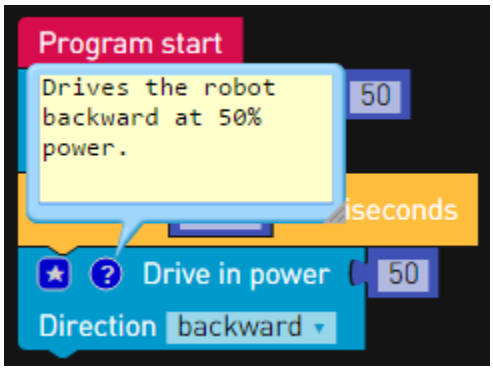
3. Click the Comment icon.

   A pop-up window appears. in which you can add comments to the block.



4. Click in the pop-up window.

5. Type a comment.



6. Click the Comment icon again.

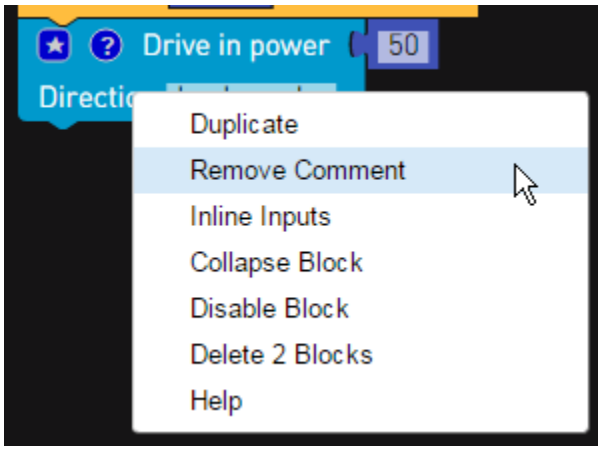
   The comment pop-up is closed.

You can view the comment again by clicking the Comment icon.

5.6.2. Removing a Comment

To remove a comment:

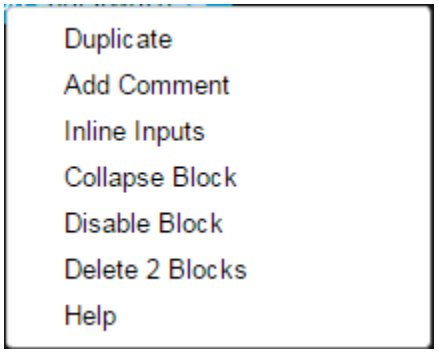
1. Right-click a block that has a comment.
2. Click **Remove Comment**.



The comment is removed.

5.7. CONTEXT OPTIONS

This section describes the options in a context menu that are displayed when right-clicking a block.



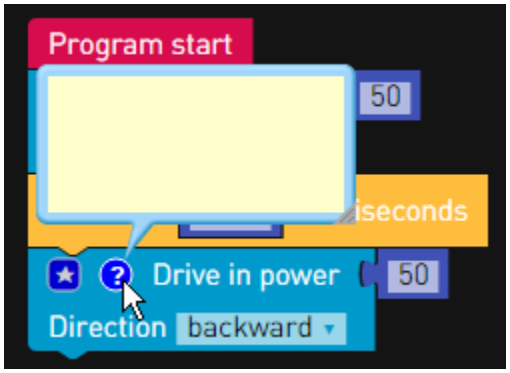
5.7.1. Duplicate

Click **Duplicate** to duplicate a block and all those horizontally connected to the block that you right-clicked.

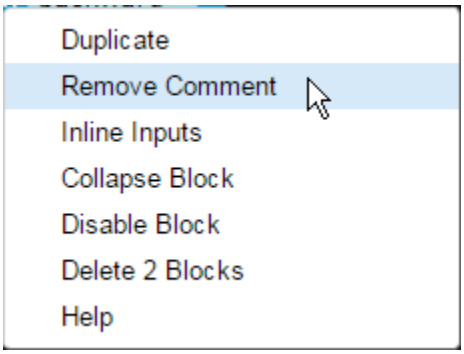
5.7.2. Add Comment / Remove Comment

Click **Comment** to add a Comment icon to the block.

Click the Comment icon to display a pop-up window in which you can add comments to the block. For more information, refer to section 5.6.1 Adding a Comment, above.



When a block has a Comment, the block's context menu has a Remove Comment option instead. Click Remove Comment to delete the comment.

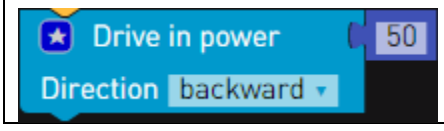
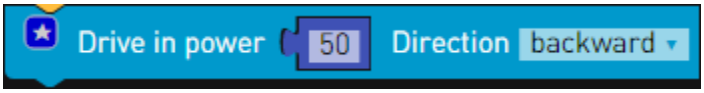


For more information, refer to section 5.6.2 Removing a Comment, above.

5.7.3. Inline Inputs / External Inputs

This option changes the block to either include input blocks within the current block, or to append the input blocks on the right of the current block.

The context menu option changes depending on the current setting.

External Inputs	Inline Inputs
	

5.7.4. Collapse Block / Expand Block

This option collapses or expands the current block and those that are horizontally attached.

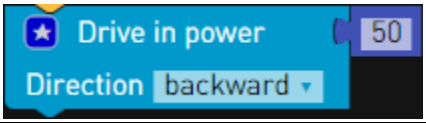
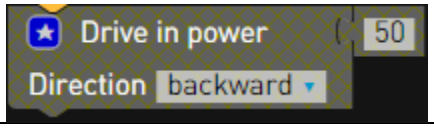
Expand Block	Collapse Block
	

It is useful to collapse blocks when a program is long and difficult to read. Collapsing blocks shortens a program so that it is easier to read and see the entire program. On the other hand, while blocks are collapsed, information is hidden.

The context menu option changes depending on the current setting.

5.7.5. Disable Block / Enable Block

This option disables or enables the current block and those that are horizontally attached.

Enable Block	Disable Block
	

Disabled blocks are ignored by the program. Instead of deleting blocks that you do not want but may need later, you can disable them and enable them when needed.

The context menu option changes depending on the current setting.

5.7.6. Delete Block

Click **Delete Block** to delete a block and those that are horizontally attached. For more information, refer to section 5.5 Deleting Blocks, above.

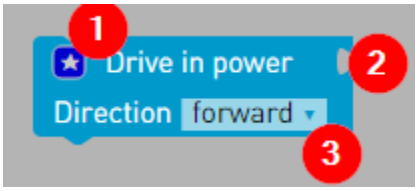
5.7.7. Help

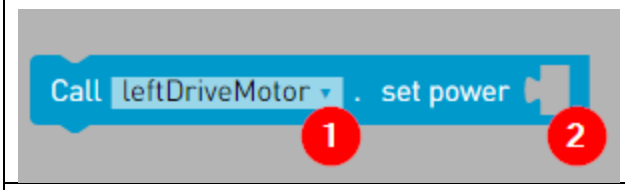
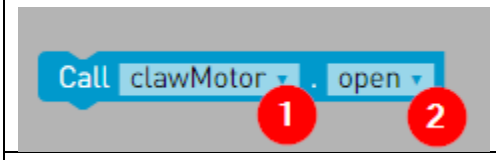
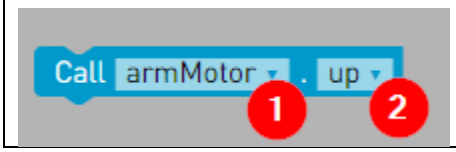
Click **Help** to display information about the block and an example program that uses the block.

5.8. BLOCK LIBRARY

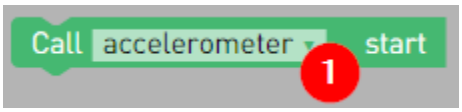
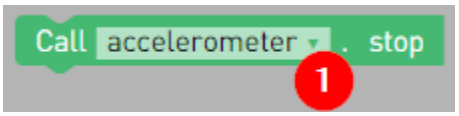
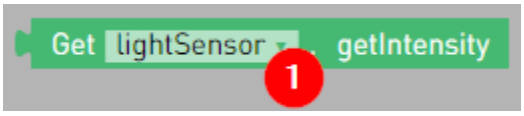
The Block Library contains the blocks that are used to write programs, divided by category. This section describes each block.

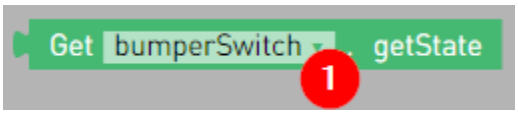
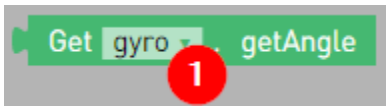
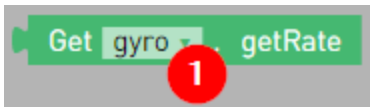
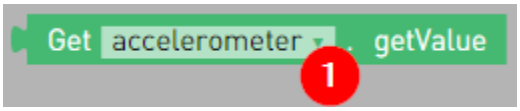
5.8.1. Motor Blocks

Block	Description	Configuration
	Drives a robot (rotates both motors).	1. Click the blue star icon to add additional parameters: • **Duration**: How long the robot should drive for (milliseconds). • **Steering**: The steering angle for the drive. 0 drives the robot straight.

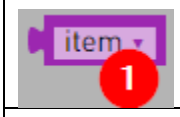
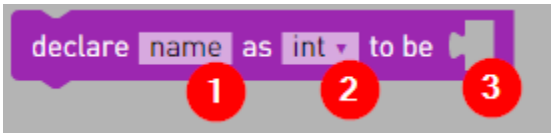
Block	Description	Configuration
		For more information, refer to section 5.3.3.1 Drive Block, above. 2. Add a number block to define the power (%) for the robot to drive at. 3. Select the direction to drive in (forward/backward).
	Rotates a single motor.	1. Select which motor to rotate. 2. Add a number block to define the power (%) for motor rotation.
	Opens or closes a ClawMotor.	1. Select which ClawMotor to control. 2. Select either open or close.
	Raises or lowers an armMotor.	1. Select which armMotor to control. 2. Select either up or down.

5.8.2. Sensor Blocks

Block	Description	Configuration
	Initializes a sensor. Sensors must be initialized before they can be used in a program.	1. Select which sensor to initialize.
	Stops a sensor. Use this block when the sensor is no longer needed.	1. Select which sensor to stop.
	Checks the distance of an object in front of the robot's ultrasonic distance sensor.	1. Select the ultrasonic sensor to check if there is more than one. 2. Select whether to detect distance in inches or meters.
	Checks the light intensity detected by the robot's light sensor.	1. Select the light sensor to check if there is more than one.

Block	Description	Configuration
	Checks the state of the robot's bumper switch (pressed/not pressed).	1. Select the bumper switch to check if there is more than one.
	Checks the angle of the robot's gyro, relative to its position when the program started.	1. Select the gyro to check if there is more than one.
	Checks the rate at which the gyro changes angle.	1. Select the gyro to check if there is more than one.
	Checks the acceleration of the robot using its accelerometer.	1. Select the accelerometer to check if there is more than one.

5.8.3. Variables Blocks

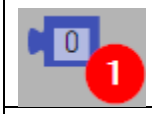
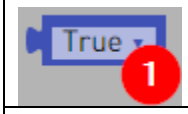
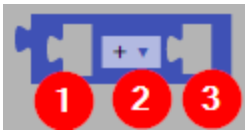
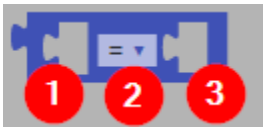
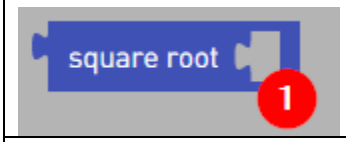
Block	Description	Configuration
	Calls a variable.	1. Select which variable to call.
	Creates a variable.	1. Define a name for the variable. 2. Select the variable type (int, float, String, bool). 3. Insert a number block to define the value for the variable.
	Changes the value of a variable.	1. Select the variable to change. 2. Insert a number block to define a new value for the variable.

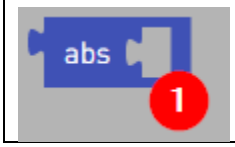
5.8.4. Output Blocks

Block	Description	Configuration
	Prints a message to the top line of the robot's LCD display.	1. Add a text or number block. Its contents will be printed to the robot's LCD display.

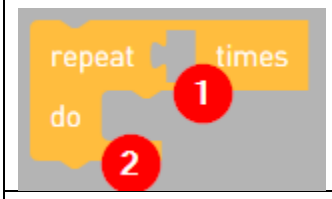
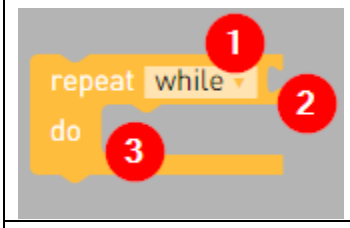
Block	Description	Configuration
	Prints a message to a specific line of the robot's LCD display.	1. Select which LCD display to print to if there is more than one. 2. Add a text or number block. Its contents will be printed to the robot's LCD display. 3. Select which line of the LCD display to print to (top/bottom).

5.8.5. Data Blocks

Block	Description	Configuration
	Contains a number.	1. Type a numerical value. The value can be any real number including negative or rational numbers.
	Contains text.	1. Type text in the field.
	Represents a Boolean value (True/False).	1. Select True or False.
	Performs a mathematical operation between two numbers.	1. Insert a number block. 2. Select an operation (addition, subtraction, multiplication, exponentiation). 3. Insert a number block.
	Checks the relationship between two numbers and returns a Boolean value.	1. Insert a number block. 2. Select a relationship to check (equal, not equal, less than, less than or equal to, more than, more than or equal to). 3. Insert a number block.
	Negates a number.	1. Add a number block. The result of the two blocks together is the negative value of the number block.
	Checks two conditions and returns a Boolean value.	1. Insert a number block. 2. Select the conditions (and, or, not, exclusive or). 3. Insert a number block.
	Performs a square root operation.	1. Insert a number block. The result of the two blocks is the square root of the number block.
	Performs exponentiation.	1. Insert a number block for the base. 2. Insert a number block for the exponent.

Block	Description	Configuration
	Returns the absolute value of a number.	1. Insert a number block.

5.8.6. Control Flow Blocks

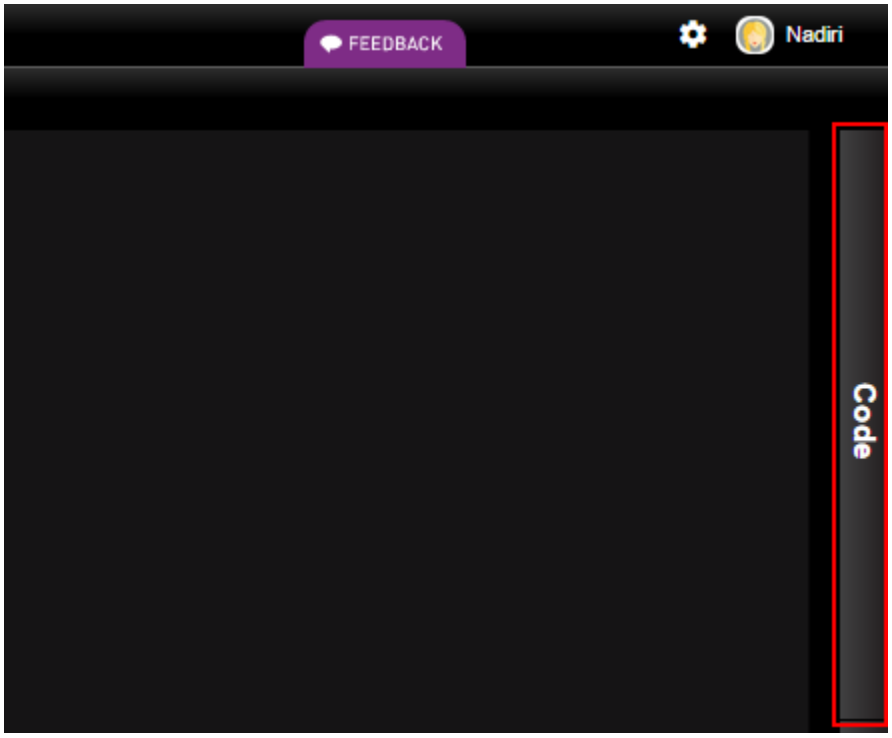
Block	Description	Configuration
	Repeats a single block or multiple blocks in a loop a defined number of times.	1. Insert a number block to define the number of times to repeat the loop. 2. Insert a single block or multiple blocks to repeat in a loop.
	Repeats a single block or multiple blocks in a loop while or until a condition is true.	1. Select while to repeat the loop while a condition is true, or until to repeat the loop while the condition is false. 2. Add a condition to check. 3. Insert a single block or multiple blocks to repeat in a loop.
	Repeats a single block or multiple blocks in a loop if a condition is true. Alternate conditions can be added as described in section 5.3.3.2 If..Do, above.	1. Add a condition to check. 2. Insert a single block or multiple blocks to repeat in a loop.
	Pauses program execution for a defined time. This block does not affect the robots current behavior. It pauses the program from continuing to execute the next commands until the defined time has expired.	1. Insert a number block for the time (milliseconds) to pause.

5.9. CODE PANE

The Code pane displays the C++ code behind the Blocks program that you write. Any changes made in a Blocks program will dynamically change the C++ code in the Code Pane.

5.9.1. Expanding the Code Pane

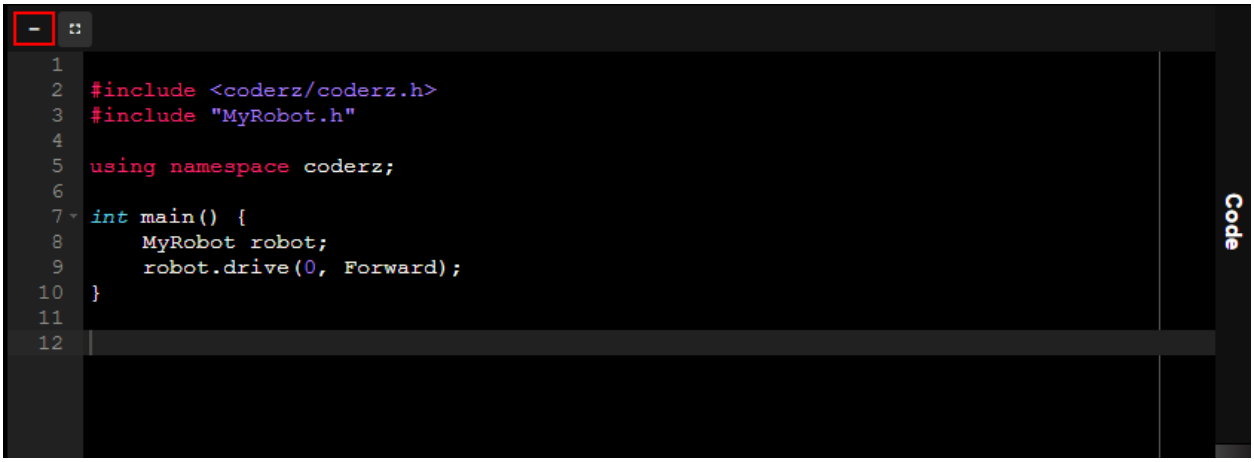
To expand the Code pane, on the top-right of the Blocks interface, click the collapsed Code pane.



5.9.2. Collapsing the Code Pane

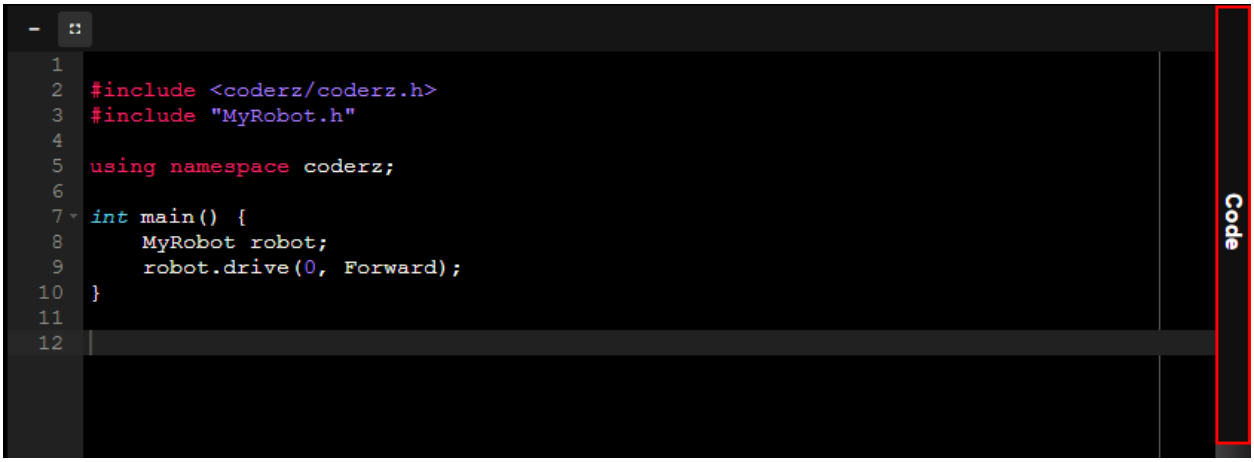
To collapse the Code pane, click one of the following:

- The Code pane minimize button.



or

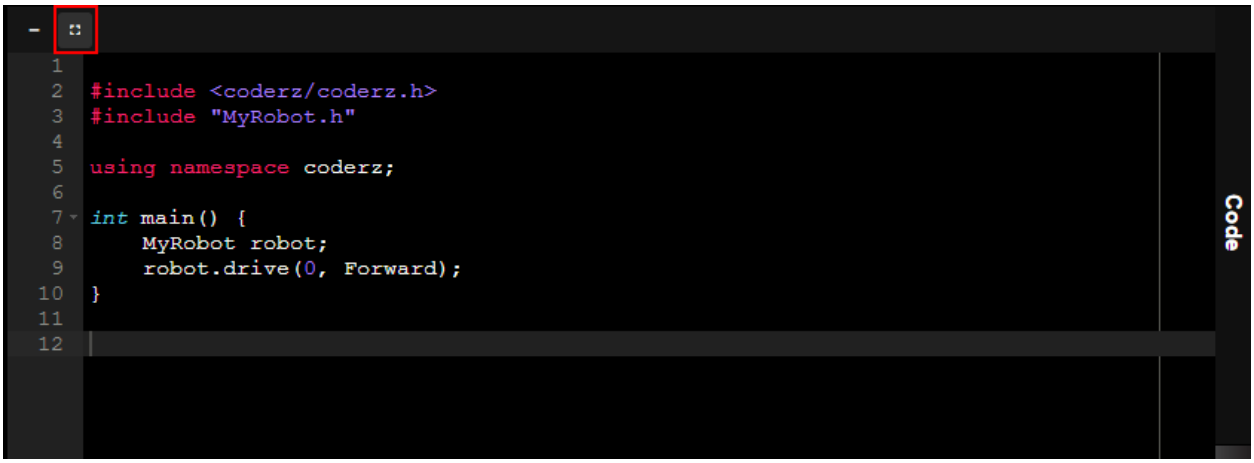
- The Code bar.



5.9.3. Maximizing / Restoring the Code Pane

The Code pane can be maximized. Doing so hides the Project Workspace area completely.

To maximize the Code pane, click the Full Screen button.



To restore the Code pane to its original width, click the Full Screen button again.

6. Code Interface (C++ / Java)

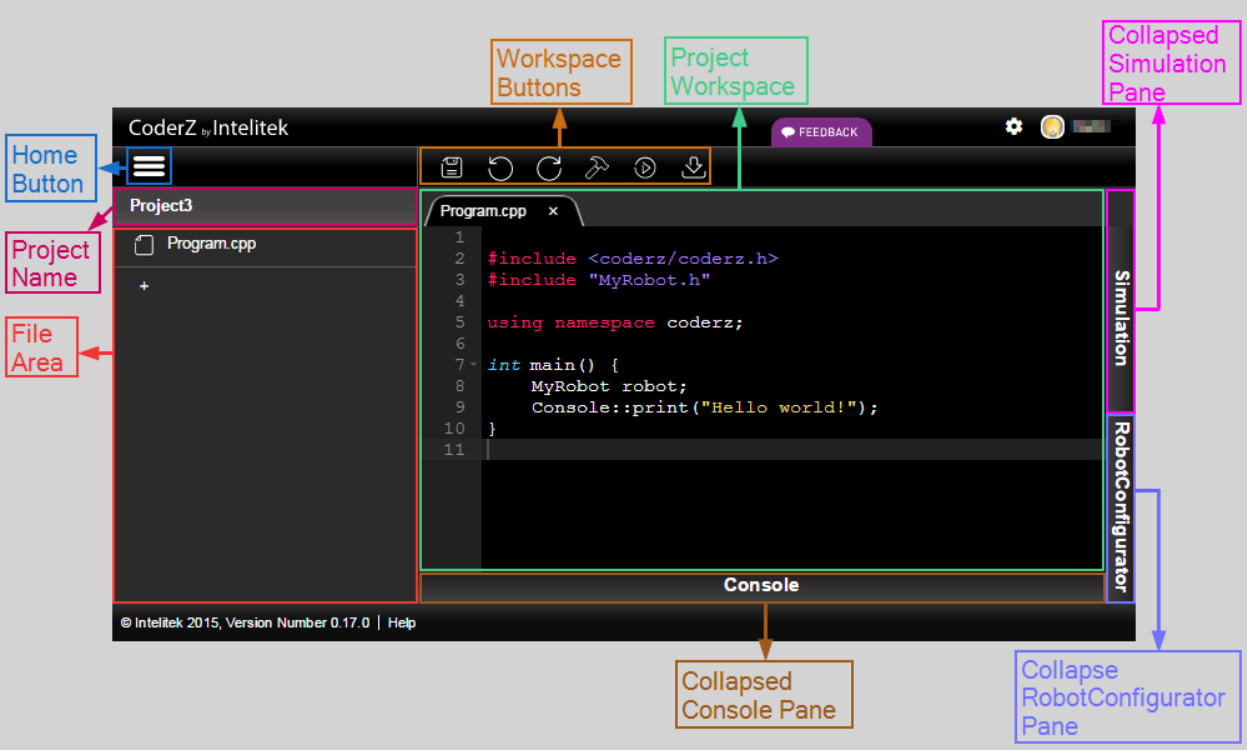
The C++ and Java interfaces are similar. Images below are from the C++ interface, but the Java interface has the same elements.

The descriptions in this section are true for both interfaces.

This section describes the following:

- Overview
- Managing Files
- Commenting
- Search
- Autocomplete
- C++ Commonly Used Functions
- C++ Sample Programs
- C++ Library
- Java Commonly Used Functions
- Java Sample Programs

6.1. OVERVIEW



This section describes the elements of the code interface. Refer to the image above for the location of these elements.

The code interface contains the following elements:

Name	Description

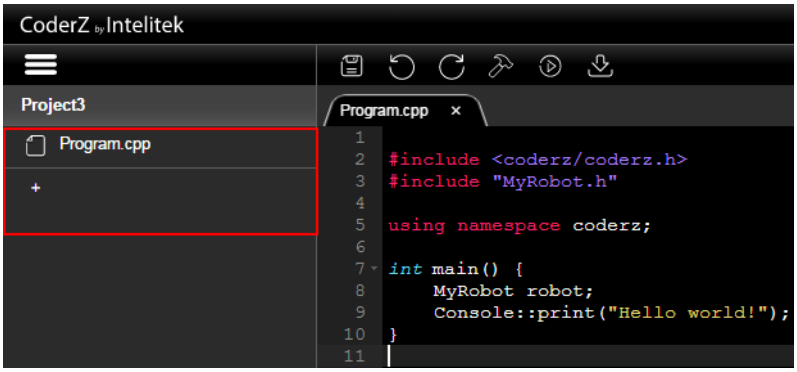
Name	Description
Home Button	Click the home button to return to the Welcome page of the Home interface.
Project Name	The name of the project is displayed in the Project Name area. Hover over the Project Name area to display the trashcan and pencil icons for deleting and renaming the project. For more information, refer to section 4.3 Renaming a Project and section 4.4 Deleting a Project above.
File Area	The File area contains all programs in the project.
Workspace Buttons	The following Workspace buttons are above the Project Workspace:  Save: Saves the open project.  Undo: Undoes an action.  Redo: Redoes an action that was undone.  Compile: Compiles the program. After clicking this button, the Console is displayed with information about compilation. If there are problems with the program, details of the problems are written to the console. If there are no problems with the program, a success message is written to the console.  Run Simulation: Compiles the program and runs it in simulation. After clicking this button, the Simulation Pane expands and the simulated robot runs the program.  Download to Robot: Downloads the program to a physical robot. The robot must be connected via a USB cable to the PC. For more information, refer to section Downloading to Robot, below.
Project Workspace	Programs are written in the Project Workspace.
Collapsed Simulation Pane	Expands the Simulation pane. The Simulation Pane contains a simulation of the robot as configured in the RobotConfigurator that can run programs. For more information, refer to section 7.1 Simulation, below.
Collapsed RobotConfigurator Pane	Expands the RobotConfigurator Pane. In this pane you can configure your robot and its ports. For more information, refer to section 7.2 RobotConfigurator, below.

Name	Description
Collapsed Console Pane	Expands the Console Pane. Program information including compilation details and program running details is written to the console. For more information, refer to section 7.3 Console, below.

6.2. MANAGING FILES

A C++ or Java project can contain multiple files.

Project files appear as a list in the File area.



This section describes the following:

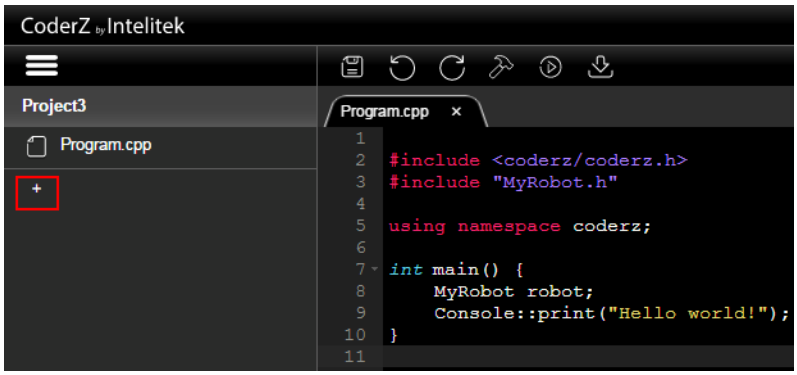
- Adding Files
- Renaming a File
- Opening/Closing a File
- Deleting a File

6.2.1. Adding Files

When a file is added, a .cpp and a .hpp file are created.

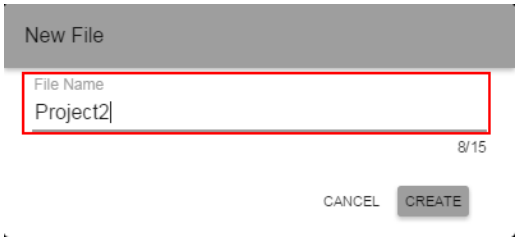
To add a file to a project:

1. Below the list of existing files in the File Area, click the **+** icon.

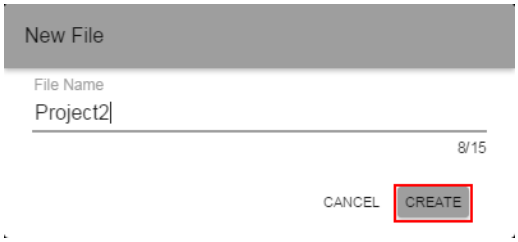


The New File dialog box appears.

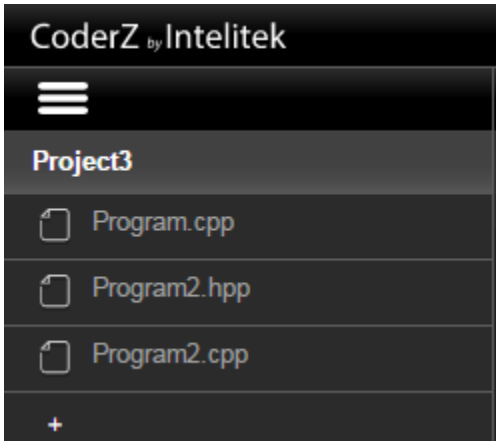
2. In the File Name field, name the file.



3. Click **Create**.



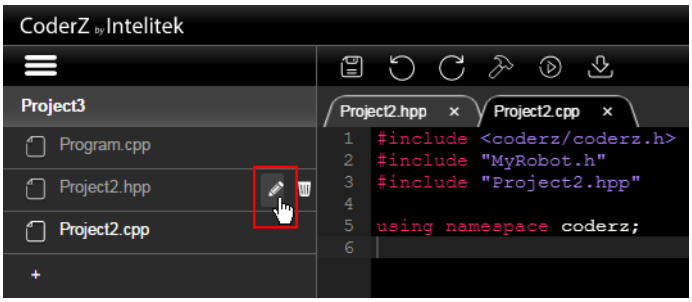
The file is created and the .cpp and .hpp file appear in the Files Area.



6.2.2. Renaming a File

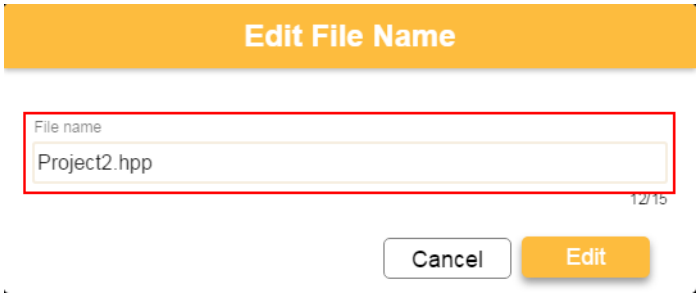
To rename a file:

1. In the File Area, hover over the filename and click the pencil icon.

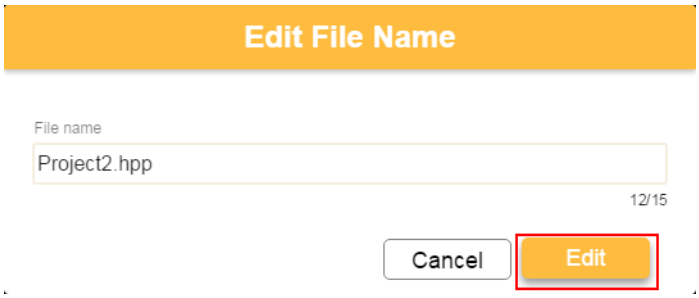


The Edit File Name dialog box appears.

2. In the File name field, rename the file.



3. Click **Edit**.

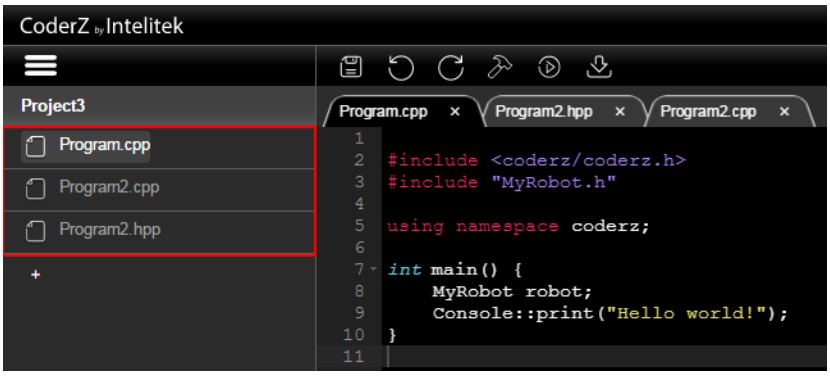


The file is renamed.

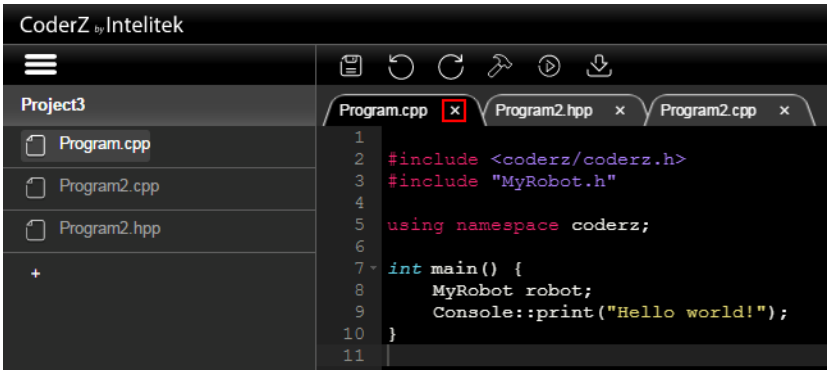
6.2.3. Opening/Closing a File

Open files appear as tabs at the top of the Project Workspace.

To open a file, in the File Area click the name of a file.



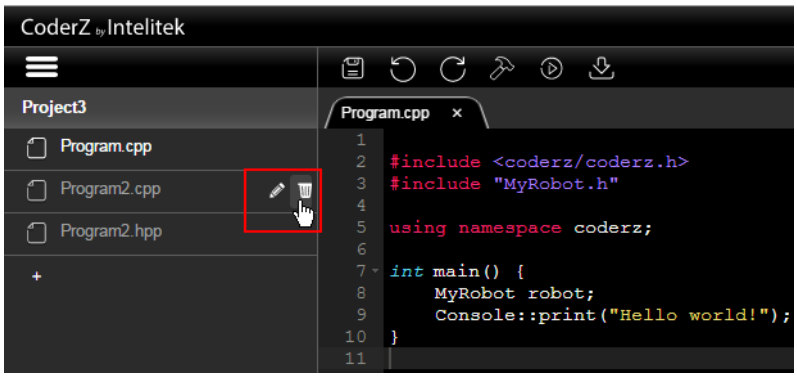
To close a file, on the file's tab, click the x.



6.2.4. Deleting a File

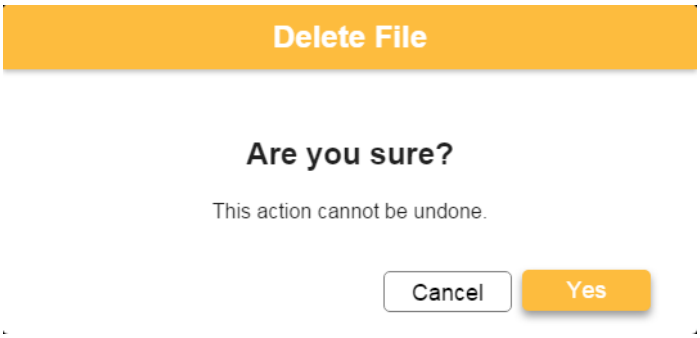
To delete a file:

1. In the File Area, hover over the filename and click the trashcan icon.



   The Delete File dialog box appears.

2. Click **Yes**.



   The file is deleted.

6.3. COMMENTING

Adding comments to a program is good practice and very useful for explaining the program to others, or reminding yourself of the purpose of how the program works.

Comments are ignored when the program is executed.

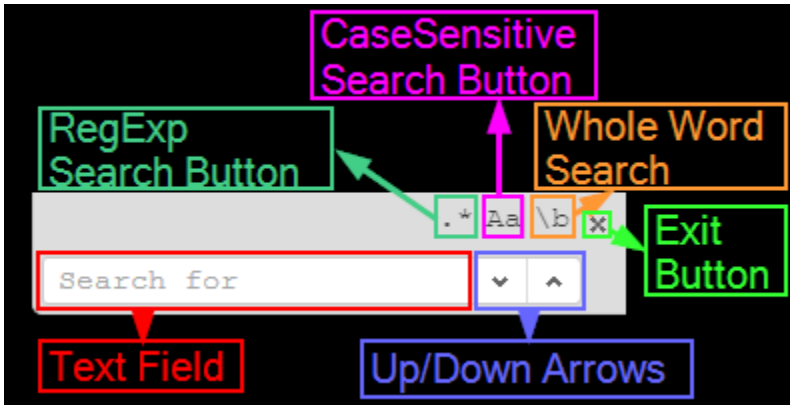
To add a comment, add two forward slashes followed by the comment.

```
Program.cpp ×
 1
 2  #include <coderz/coderz.h>
 3  #include "MyRobot.h"
 4
 5  using namespace coderz;
 6
 7  int main() {
 8      MyRobot robot;
 9      Console::print("Hello world!"); // Prints "Hello World" to the console
10  }
11
```

6.4. SEARCH

You can search within the code in the Project Workspace using the Search box. This section describes the Search box and how to search for a String.

6.4.1. Search Box



This section describes the elements of the Search box. Refer to the image above for the location of these elements.

The Search box contains the following elements:

Name	Description
Text Field	In the Text Field, enter the String to search for.
Up/Down Arrows	Click ˅ to search upward in the program or ^ to search downward.
Exit Button	Click X to close the Search box.
RegExp Search Button	Click the RegExp Search button to allow regular expressions in the search. For example, an asterisk can be used to represent any character.
CaseSensitive Search Button	Click the CaseSensitive Search button to make the search case sensitive.

Name	Description
Whole Word Search	Click the Whole Word Search button to only search for whole words. When enabled, if the search String is part of a word, it will be ignored in a search.

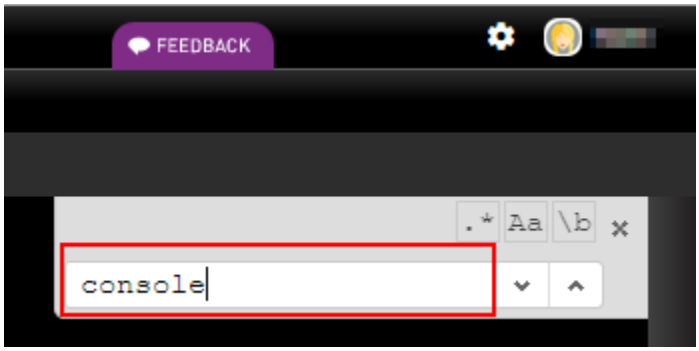
6.4.2. Search for a String

To search for a String:

1. Press **Control** and **F** simultaneously.

   The Search box appears.

2. Type a search string in the field.



3. Press ˅ to search upward in the program or ^ to search downward.

6.5. AUTOCOMPLETE

When adding the first of any of the following pairs to a program, the second is automatically added:

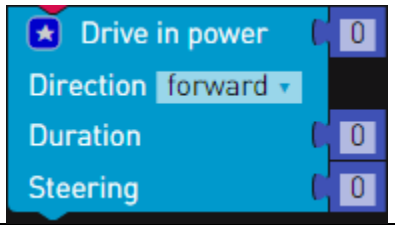
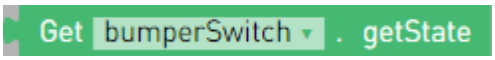
- Round brackets: ( )
- Braces: { }
- Quotation marks: “ ”

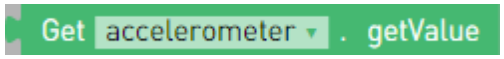
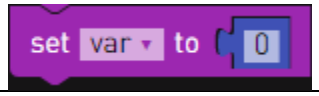
6.6. C++ COMMONLY USED FUNCTIONS

This section provides the C++ code for commonly used functions and their parameters.

The table below shows blocks from the Blocks interface and the equivalent C++ code.

A reference to the class in the C++ Library (Section 6.8 C++ Library) where you can find more details and descriptions is provided for each function.

Category	Block Function and Parameter Blocks	Equivalent C++ Code and Parameters
Motor		robot.drive(0, Forward, 0, 0); **See Class: Robot**
Motor		robot.getLeftDriveMotor().setPower(0); **See Class: Robot**
Motor		robot.getClawMotor().open(); **See Class: Claw**
Motor		robot.getArmMotor().up(); **See Class: Arm**
Sensor		robot.getAccelerometer().start(); **See Class: Accelerometer**
Sensor		robot.getAccelerometer().stop(); **See Class: Accelerometer**
Sensor		.reset();
Sensor		robot.getUltrasonic().getDistanceInches(); **See Class: UltraSonicSensor**
Sensor		robot.getLightSensor().getIntensity(); **See Class: LightSensor**
Sensor		robot.getBumperSwitch().getState(); **See Class: Switch**
Sensor		robot.getGyro().getAngle(); **See Class: Gyro**
Sensor		robot.getGyro().getRate(); **See Class: Gyro**

Sensor		robot.getAccelerometer().getValue(); **See Class: DigitalInput**
Sensor		.setValue(0);
Variables		item;
Variables		int var2 = 0;
Variables		var2 = 0;
Output		Console::print(0);
Output		robot.getLCD().setText(LCDLine::Top, "Hello World!"); **See Class: LCD**

Control Flow		for (int __index__1 = 0; __index__1 < 2; __index__1++) {
		}
		}

Control Flow		while (true) {}
Control Flow		if () {} else if (false) {} else {}
Control Flow		waitMilli(1000);

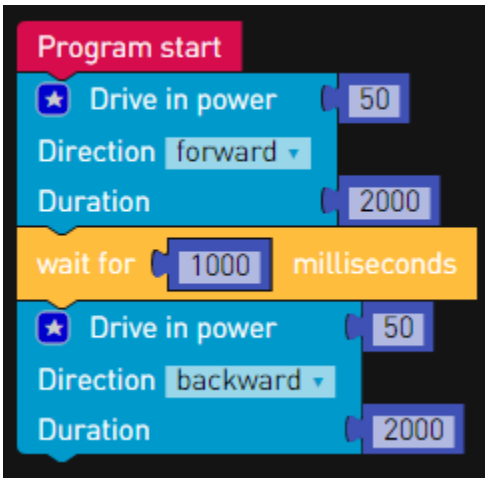
6.7. C++ SAMPLE PROGRAMS

This section provides some sample programs that demonstrate the implementation of some common CoderZ C++ functions.

Program 1

Description: The robot drives forward at 50% power for 2 seconds, pauses for 1 second and then reverses at 50% power for 2 seconds.

Blocks Program



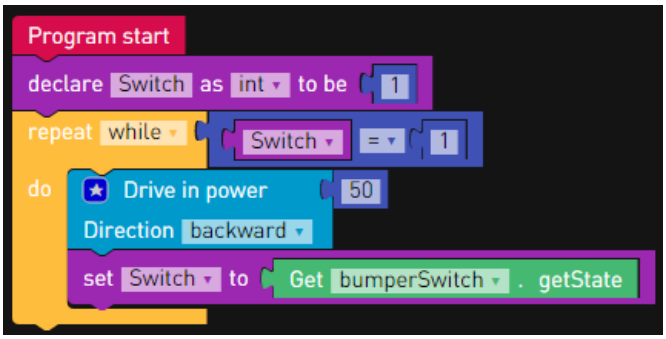
Equivalent C++ Program

```
#include <coderz/coderz.h>
#include "MyRobot.h"

using namespace coderz;

int main() {
   MyRobot robot;
   robot.drive(50, Forward, 2000);
   waitMilli(1000);
   robot.drive(50, Backward, 2000);
}
```

Program 2

Description: The state of the robot's Bumper Switch is written to a variable. While the Bumper Switch is not pressed, the robot reverses.

Blocks Program



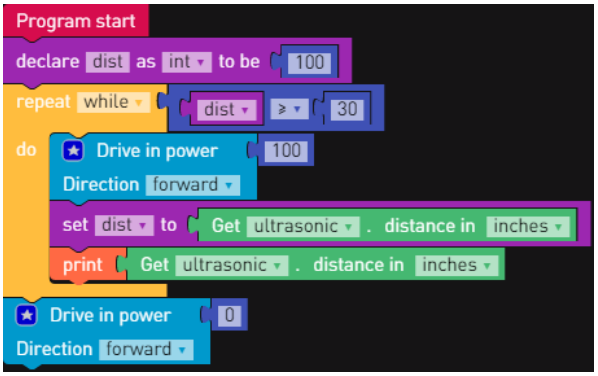
Equivalent C++ Program

```
#include <coderz/coderz.h>
#include "MyRobot.h"

using namespace coderz;

int main() {
   MyRobot robot;
   int Switch = 1;
   while (Switch == 1) {
      robot.drive(50, Backward);
      Switch = robot.getBumperSwitch().getState();
   }
}
```

Program 3

Description: The distance of an object from the robot's ultrasonic distance sensor is written to a variable. While the robot is more than 30 inches away from an object, the robot drives forward. Once it reaches 30 inches from an object, the robot stops.

Blocks Program



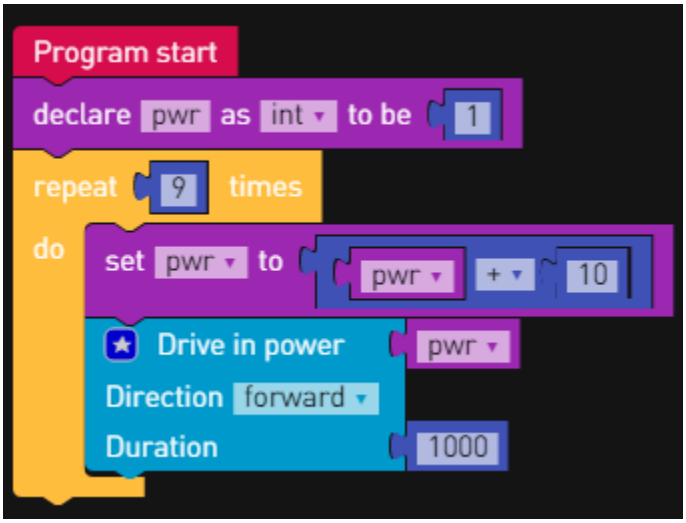
Equivalent C++ Program

```
#include <coderz/coderz.h>
#include "MyRobot.h"

using namespace coderz;

int main() {
   MyRobot robot;
   int dist = 100;
   while (dist >= 30) {
      robot.drive(100, Forward);
      dist = robot.getUltrasonic().getDistanceInches();

Console::print(robot.getUltrasonic().getDistanceInches())
;
   }
   robot.drive(0, Forward);
}
```

Program 4

Description: A variable is declared. A loop is repeated 9 times, in which the value of the variable is increased by 10 and the robot drives forward for one second at the power of the variable. The result is that the robot drives for 9 seconds, each second driving faster.

Blocks Program



Equivalent C++ Program

```
#include <coderz/coderz.h>
#include "MyRobot.h"

using namespace coderz;

int main() {
   MyRobot robot;
   int pwr = 1;
   for (int __index__1 = 0; __index__1 < 9; __index__1++) {
      pwr = pwr + 10;
      robot.drive(pwr, Forward, 1000);
   }
}
```

6.8. C++ LIBRARY

This section describes the classes and functions in the CoderZ C++ library.

Class: Accelerometer	
Accelerometer (PortNumber): Represents an accelerometer.	Parameters PortNumber: Insert the port number of the accelerometer.
getAccelerationInMilliGs (): Returns the acceleration being experienced in a single axis. An Accelerometer measures the acceleration being experienced in a single axis. Most accelerometers can measure two separate axes, labeled x and y, which should be treated as two separate sensors. An accelerometer returns values in milliG's where G is the acceleration due to gravity. The accelerometer can also be used as a tilt sensor, by measuring the component of acceleration in the vertical axis and comparing the value to one G.	
start (): Initiates an accelerometer for use in a program.	
stop (): Stops an accelerometer when it is no longer used in a program.	

Class: AnalogInput	
AnalogInput (PortNumber): Represents an analog input.	Parameters PortNumber: Insert the port number of the analog input.
getValue (): Returns the value of an analog input.	
getEndpoint (): Returns the end point of an analog input.	
Class: Arm	
Arm (PortNumber motorPortNumber): Represents an arm.	Parameters PortNumber: Insert the port number of the arm. motorPortNumber: Insert the port number of the motor.
up (): Raises an arm.	
down (): Lowers an arm.	
Class: BasicString	
BasicString (C const *c, Size size=0)	
BasicString (BasicString< C > const &other)	
BasicString< C > & **operator=** (BasicString< C > const &other)	
void **reset** ()	
void **push_back** (char const *s, Size size=0)	
void **push_back** (char c)	
Size **size** () const	
C const * **data** () const	
C * **data** ()	
C & **operator**[] (Size s)	
C const & **operator**[] (Size s) const	
Class: Battery	
virtual float **getVoltage** () const =0: Returns the battery voltage.	
Class: BumperSwitch	
BumperSwitch (PortNumber): Represents a bumper switch.	Parameters PortNumber: Insert the port number of the analog input.
bool **isPressed** () const INTELICODE_OVERRIDE: Returns a Boolean value for the state of a bumper switch; False if the bumper switch is pressed and True if it is not pressed.	

Class: Claw	
Claw (PortNumber motorPortNumber): Represents a claw.	Parameters PortNumber: Insert the port number of the claw. motorPortNumber: Insert the port number of the motor.
void **open** (): Opens the claw.	
void **close** (): Closes the claw.	
Class: Connection	
Connection (Port port, Peripheral *peripheral, PeripheralEndpoint endpoint)	
Connection (Connection const &other)	
bool **isValid** () const	
Port const & **getPort** () const	
PeripheralEndpoint const & **getPeripheralEndpoint** () const	
Peripheral & **getPeripheral** () const	
Class: DigitalInput	
DigitalInput (PortNumber): Represents a digital input.	Parameters PortNumber: Insert the port number of the digital input.
virtual bool **getValue** () const: Returns a Boolean value of a digital input.	
PeripheralEndpoint const & **getEndpoint** () const	
Class: DigitalOutput	
DigitalOutput (PortNumber): Represents a digital output.	Parameters PortNumber: Insert the port number of the digital output.
void **setOutput** (bool): Sets the output of a digital output.	Parameters bool: Insert a Boolean value for the digital output.
PeripheralEndpoint const & **getEndpoint** () const INTELICODE_OVERRIDE	
Class: DrivableRobot	
virtual void **drive** (float speed)=0: Sets the specified speed on all of the robot's wheels.	Parameters float speed: Insert the speed at which to drive the robot. A negative value drives the robot backward.
void **stop** (): Stops all of the robot's wheels.	
void **driveTimed** (float speed, unsigned int milliseconds): Sets the specified speed on all of the robot's wheels for a defined duration. Program execution will pause while the robot drives.	Parameters float speed: Insert the speed at which to drive the robot. A negative value drives the robot backward. unsigned int milliseconds: Insert an integer to define the time in milliseconds for the robot to drive.
Class: Gyro	
Gyro (PortNumber): Represents a gyro.	Parameters PortNumber: Insert the port number of the gyro.
int **getAngle** () const: Returns the angle of a gyro relative to its initial position.	
float **getRate** () const: Returns the rate of change of angle of a gyro.	
void **start** (): Initiates a gyro.	
void **stop** (): Stops a gyro when it is no longer needed in a program.	
void **setType** (unsigned int type): Sets the scaling factor so the output values match the connected sensor.	
void **setDeadband** (char deadband): Allows the user to control the Deadband to reduce noise or increase resolution.	Parameters PortNumber: Insert a character for the gyro's deadband.
Class: IntegratedMotorEncoder	
IntegratedMotorEncoder (PortNumber): Represents an integrated motor encoder.	Parameters PortNumber: Insert the port number of the motor encoder.

virtual unsigned char **initialize** (): Initializes an integrated motor encoder for use in a program.	
virtual void **setValue** (long presetValue): Sets the value of an integrated motor encoder.	Parameters long presetValue: Insert a value to set the integrated motor encoder to.
virtual long **getValue** (): Returns the value of an integrated motor encoder.	
virtual float **getSpeed** (): Returns the speed of an integrated motor encoder.	
virtual void **reset** (): Resets the value of an integrated motor encoder.	
Class: InterruptWatcher	
InterruptWatcher (PortNumber digitalPortNumber): Represents an interrupt watcher.	Parameters PortNumber: Insert the port number of the interrupt watcher. digitalPortNumber: Insert the digital port number of the interrupt watcher.
virtual void **start** (Direction): Initializes an interrupt watcher.	Parameters Direction: Insert a value to define the direction of the interrupt watcher.
virtual bool **getValue** (): Returns the value of an interrupt watcher.	
virtual void **stop** (): Stops an interrupt watcher when it is no longer needed in a program.	
Class: Joystick	
void **arcade2Motors** (uint8_t joystickIndex, uint8_t handleForwardReverse, uint8_t handleRotation, PortNumber leftMotorPort, bool bIsleftInvertDirection, PortNumber rightMotorPort, bool bIsrightInvertDirection): Assigns arcade control of two motors to a joystick.	
void **arcade4Motors** (uint8_t joystickIndex, uint8_t handleForwardReverse, uint8_t handleRotation, PortNumber leftFrontMotorPort, bool bIsLeftFrontInvertDirection, PortNumber rightFrontMotorPort, bool bIsRightFrontInvertDirection, PortNumber leftRearMotorPort, bool bIsLeftRearInvertDirection, PortNumber rightRearMotorPort, bool bIsRightRearInvertDirection): Assigns arcade control of four motors to a joystick.	
void **tank2Motors** (uint8_t joystickIndex, uint8_t leftMotorHandle, uint8_t rightMotorHandle, PortNumber leftMotorPort, bool bIsLeftInvertDirection, PortNumber rightMotorPort, bool bIsRightInvertDirection): Assigns tank control of two motors to a joystick.	
void **tank4Motors** (uint8_t joystickIndex, uint8_t leftMotorsHandle, uint8_t rightMotorsHandle, PortNumber leftFrontMotorPort, bool bIsLeftFrontInvertDirection, PortNumber rightFrontMotorPort, bool bIsRightFrontInvertDirection, PortNumber leftRearMotorPort, bool bIsLeftRearInvertDirection, PortNumber rightRearMotorPort, bool bIsRightRearInvertDirection): Assigns arcade control of four motors to a joystick.	
void **joystickToMotor** (uint8_t joystickIndex, uint8_t handleForwardReverse, PortNumber MotorPort, bool bIsInvertDirection): Assigns control of a motor to a joystick.	
void **joystickToMotorAndLimit** (uint8_t joystickIndex, uint8_t handleForwardReverse, PortNumber MotorPort, bool bIsInvertDirection, PortNumber positiveLimitSwitch, PortNumber negativeLimitSwitch): Assigns control of a motor to a joystick.	
void **joystickToServo** (uint8_t joystickIndex, uint8_t handleForwardReverse, PortNumber MotorPort, bool bIsInvertDirection): Assigns control of a servo to a joystick.	
void **joystickToDigitalOutput** (uint8_t joystickIndex, uint8_t controlButton, uint8_t buttonStatus, PortNumber outputPort): Assigns control of a digital output to a joystick.	
int **getJoystickAnalog** (uint8_t joystickIndex, uint8_t handle)	
int **getJoystickAccelerometer** (uint8_t joystickIndex, uint8_t axe)	

Class: LCD	
void **setText** (LCDLine::Value, String const &message): Prints text to a specific line on the LCD display.	Parameters LCDLine::Value: Insert a value to define which line of the LCD display to print to. String const &message: Insert a String of text to print to the LCD display.
void **setText** (LCDLine::Value, char const *message): Prints a numerical value to the LCD display.	Parameters LCDLine::Value: Insert a value to define which line of the LCD display to print to. char const *message: Insert a number value to print to the LCD display.
void **setBackLight** (bool): Turns the LCD display's backlight on or off.	Parameters bool: Insert a Boolean value to turn the LCD display's backlight on or off.
LCDButtonsState **getButtonsState** () const: Returns the state of all three LCD buttons.	
bool **isButton1Pressed** () const: Returns a Boolean value for the state of Button 1.	
bool **isButton2Pressed** () const: Returns a Boolean value for the state of Button 2.	
bool **isButton3Pressed** () const: Returns a Boolean value for the state of Button 3.	
PeripheralEndpoint const & **getEndpoint** () const INTELICODE_OVERRIDE	
Class: LCDButtonsState	
Class: LCDLine	
enum Value { Top, Bottom }	Parameters Top, Bottom: Define which line of the LCD display.
Class: LightSensor	
LightSensor (PortNumber analogPortNumber): Represents a light sensor.	Parameters PortNumber: Insert the port number of the light sensor. analogPortNumber: Insert the analog port number of the light sensor.
AnalogValue **getIntensity** () const: Returns the intensity detected by a light sensor.	
Class: LimitSwitch	
LimitSwitch (PortNumber): Represents a limit switch.	Parameters PortNumber: Insert the port number of the limit switch.
bool isPressed () const INTELICODE_OVERRIDE: Returns a Boolean value for the state of the limit switch.	
Class: LineFollower	
LineFollower (PortNumber analogPortNumber): Represents a line follower.	Parameters PortNumber: Insert the port number of the line follower. analogPortNumber: Insert the analog port number of the line follower.
AnalogValue getIntensity () const: Returns the intensity detected by a line follower.	
Class: Map	
void **insert** (Key const &k, Value const &v)	
void **insert** (Mapping const &m)	
iterator **begin** ()	
iterator **end** ()	
const_iterator **begin** () const	
const_iterator **end** () const	
Value & **operator[]** (Key const &key)	
Value const & **operator[]** (Key const &key) const	
const_iterator **find** (Key const &k) const	
iterator **find** (Key const &k)	

Vector< Key > **keys** () const	
Class: Motor	
Motor (PortNumber motorPortNumber): Represents a motor.	Parameters PortNumber: Insert the port number of the motor. motorPortNumber: Insert the motor port number of the motor.
virtual void **setPower** (int speed): Sets the power of a motor.	Parameters int speed: Insert a value to define the power of a motor.
virtual void **setPowerTimed** (int speed, unsigned int milliseconds): Sets the power of a motor for a defined duration.	Parameters int speed: Insert a value to define the power of a motor. milliseconds: Insert a value to define int unsigned int milliseconds: Insert an integer to define the time in milliseconds for the motor to rotate.
virtual void **stop** (): Stops a motor.	
PeripheralEndpoint const & **getEndpoint** () const	
Class: OpticalEncoder	
OpticalEncoder (PortNumber): Represents an optical encoder.	Parameters PortNumber: Insert the port number of the optical encoder.
virtual void **start** (): Initializes an optical encoder.	
virtual void **stop** (): Stops an optical encoder when it is no longer needed in a program.	
virtual long **getValue** (): Returns the value of an optical encoder.	
virtual void **setValue** (long lPresetValue): Sets the value of an optical encoder.	Parameters long lPresetValue: Insert a value to set the optical encoder to.
virtual void **reset** (): Resets the value of an optical encoder.	
Class: OpticalQuadEncoder	
OpticalQuadEncoder (int iChannelA, int iChannelB): Represents an optical quad encoder.	Parameters int iChannelA: Insert the value of iChannelA. int iChannelB: Insert the value of iChannelB.
virtual void **start** (bool bInvertDirection): Initializes an optical quad encoder.	Parameters bool bInvertDirection: Insert a Boolean value to set the direction of the optical quad encoder.
virtual void **stop** (): Stops an optical quad encoder when it is no longer needed in a program.	
virtual long **getValue** (): Returns the value of an optical quad encoder.	
virtual void **setValue** (long lSetValue): Sets the value of an optical quad encoder.	Parameters long lSetValue: Insert a value to set the optical quad encoder to.
virtual void **reset** (): Resets an optical quad encoder.	
Class: Peripheral	
enum **Type** { **TypePeripheral**, **TypeWheel** }	
virtual **Type**	
virtual **Vector**	
< PeripheralEndpoint > **getEndpoints** () const =0	
Class: PeripheralEndpoint	
PeripheralEndpoint (ConnectionType type, char const *name="port")	
PeripheralEndpoint (PeripheralEndpoint const &other)	
char const * **getName** () const	
ConnectionType **getConnectionType** () const	
Class: Port	
Port (PortNumber portNumber=0, ConnectionType type=ConnectionTypeInvalid): Represents a port.	

Port (Port const &p): Represents a port.	
bool **isValid** () const: Returns a Boolean value for whether or not a port is valid.	
void **setPortNumber** (PortNumber portNumber): Sets a port number.	
void **setType** (ConnectionType type): Sets a port connection type.	
PortNumber **getPortNumber** () const: Returns a port number.	
ConnectionType **getConnectionType** () const: Returns a connection type.	
Class: Potentiometer	
Potentiometer (PortNumber analogPortNumber): Represents a potentiometer.	Parameters PortNumber: Insert the port number of the potentiometer. analogPortNumber: Insert the analog port number of the potentiometer.
Class: Robot	
Peripheral & **getPeripheralByPort** (Port)	
Connection const & **getConnectionByPort** (Port)	
Connection const & **getConnectionByPort** (PortNumber, ConnectionType)	
void **drive** (int wheelPower, Direction direction=Forward, int timeMilli=-1, int steering=0): Drives the robot at a defined power and for a defined duration.	Parameters int wheelPower: Insert a value to define the power at which to drive the robot. Direction direction=Forward: Define the direction the robot should drive in, forward or backward. int timeMilli=-1: Insert an integer to define the time in milliseconds for the robot to drive. int steering=0: Insert an integer to define the steering angle for the drive. A value of 0 drives the robot straight.
void **connect** (Port port, Peripheral *peripheral, PeripheralEndpoint endpoint)	
void **connect** (SinglePortPeripheral *peripheral)	
Vector< Connection > const & **getConnections** () const	
T & **getPeripheralByPortAs** (Port port)	
Class: RuntimeException	
RuntimeException (char const *what)	
char const * **what** () const INTELICODE_OVERRIDE	
Class: Servo	
Servo (PortNumber port): Represents a servo.	Parameters PortNumber port: Insert the port number of the servo.
virtual void **setSpeed** (int speed): Sets the speed of a servo.	Parameters int speed: Insert a value to define the speed of the servo.
virtual void **stop** (): Stops a servo.	
virtual int **getSpeed** (): Returns the speed of a servo.	
Class: SimulationRobot	
SmartMotor & **getLeftMotor** ()	
SmartMotor & **getRightMotor** ()	
DigitalOutput & **getLed** ()	
UltrasonicSensor & **getUltrasonicSensor** ()	
LightSensor & **getLightSensor** ()	
LCD & **getLCD** ()	
BumperSwitch & **getBumperSwitch** ()	
Accelerometer & **getAccelerometer** ()	
Gyro & **getGyro** ()	

LineFollower & **getLineFollower** ()	
void **drive** (float speed) INTELICODE_OVERRIDE	
Class: SinglePortPeripheral	
SinglePortPeripheral (**Port** port)	
Port const & **getPort** () const	
PortNumber **getPortNumber** () const	
virtual **PeripheralEndpoint** const & **getEndpoint** () const =0	
Vector< **PeripheralEndpoint** > **getEndpoints** () const INTELICODE_OVERRIDE	
Class: SmartMotor	
SmartMotor (PortNumber): Represents a smart motor.	
SmartMotor (Port)	
void **rotateDegrees** (int degrees, float speed=1)	
void **rotateTimes** (float times, float speed=1)	
Class: SmartTasks	
void **multiSmartMotorsTimeControl** (int motorsCount, PortNumber motor1, PortNumber motor2, PortNumber motor3, PortNumber motor4, bool InvertMotor1, bool InvertMotor2, bool InvertMotor3, bool InvertMotor4, int iPower, int iMilliseconds)	
void **multiSmartMotorsRotateDegrees** (int motorsCount, int masterMotorIndex, PortNumber motor1, PortNumber motor2, PortNumber motor3, PortNumber motor4, bool InvertMotor1, bool InvertMotor2, bool InvertMotor3, bool InvertMotor4, int iPower, int iDegrees)	
void **multiSmartMotorsRotations** (int motorsCount, int masterMotorIndex, PortNumber motor1, PortNumber motor2, PortNumber motor3, PortNumber motor4, bool InvertMotor1, bool InvertMotor2, bool InvertMotor3, bool InvertMotor4, int iPower, int iRotations)	
Class: Speaker	
void **playAudioFile** (char const *fileName, bool repeat=false, bool wait=false): Plays an audio file on the robot's speaker.	Parameters char const *fileName: Insert the filename of the audio file to play. bool repeat=false: Insert a Boolean value to set the audio to repeatedly play in a loop or not. bool wait=false:
void **playAudioTone** (int frequency, seconds duration, bool wait=false) : Plays an audio tone on the robot's speaker.	Parameters int frequency: Insert an integer to define the frequency at which to play the audio tone. seconds duration: Insert a value to define the number of seconds for which to play the audio tone. bool wait=false:
void **stop** (): Stops audio that is being played on the robot's speaker.	
void **setMute** (bool): Mutes the robot's speaker.	Parameters bool: Insert a Boolean value to mute or unmute the speaker.
Class: Switch	
Switch (PortNumber p): Represents a switch.	Parameters PortNumber p: Insert the port number of the switch.
bool **getState** () const: Returns the state of a switch.	
virtual bool **isPressed** () const =0: Returns a Boolean value to describe the state of a switch as pressed or unpressed.	

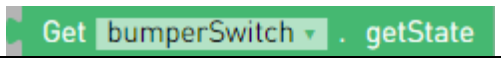
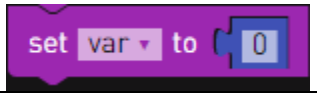
Class: UltrasonicSensor	
UltrasonicSensor (PortNumber inputDigitalPort, PortNumber outputDigitalPort): Represents an ultrasonic sensor.	Parameters PortNumber inputDigitalPort: Insert the digital input port number of the ultrasonic sensor. PortNumber outputDigitalPort: Insert the digital output port number of the ultrasonic sensor.
meters **getDistance** () const: Returns the distance of an object from an ultrasonic sensor in meters.	
float **getDistanceMeters** () const: Returns the distance of an object from an ultrasonic sensor in meters.	
float **getDistanceInches** () const: Returns the distance of an object from an ultrasonic sensor in inches.	
void **start** (): Initializes an ultrasonic sensor.	
void **stop** (): Stops an ultrasonic sensor when it is no longer needed in a program.	
Port **getInputPort** () const: Returns the input port of an ultrasonic sensor.	
Port **getOutputPort** () const: Returns the output port of an ultrasonic sensor.	
Vector< **PeripheralEndpoint** > **getEndpoints** () const INTELICODE_OVERRIDE	
Class: Vector	
Vector (Vector const &other)	
iterator **begin** ()	
iterator **end** ()	
const_iterator **begin** () const	
const_iterator **end** () const	
void **push_back** (T const &t)	
const_iterator **find** (T const &t) const	
bool **contains** (T const &t) const	
T & **operator[]** (Size i)	
T const & **operator[]** (Size i) const	
Size **size** () const	
T * **data** ()	
T const * **data** () const	
Class: WheelBase	
WheelBase (PortNumber motorPortNumber, Direction=Forward): Represents a wheel base.	Parameters PortNumber motorPortNumber: Insert the motor port number of the wheel base. Direction=Forward: Set the direction of the wheel base as forward or backward.
void **setDirection** (Direction): Sets the direction of a wheel base.	Parameters Direction: Set the direction of the wheel base as forward or backward.
Peripheral::Type **getType** () const INTELICODE_OVERRIDE	
void **setDirection** (char const *direction): Sets the direction of a wheel base.	
Direction **getDirection** () const: Returns the direction of a wheel base.	
void **setPower** (int power) INTELICODE_OVERRIDE: Sets the power of a wheel base.	Parameters Direction: Insert a value to define the power of the wheel base.

6.9. JAVA COMMONLY USED FUNCTIONS

This section provides the Java code for commonly used functions and their parameters.

The table below shows blocks from the Blocks interface and the equivalent Java code.

A reference to the class in the Java Library (Section _____) where you can find more details and descriptions is provided for each function.

Category	Block Function and Parameter Blocks	Equivalent Java Code and Parameters
Motor		robot.getLeftDriveMotor().setPower(0);
Motor		robot.getClawMotor().open();
Motor		robot.getArmMotor().up();
Motor		.rotateDegrees(0, 0);
Sensor		robot.getAccelerometer().start();
Sensor		robot.getAccelerometer().stop();
Sensor		.reset();
Sensor		robot.getUltrasonic().getDistanceInches();
Sensor		robot.getLightSensor().getIntensity();
Sensor		robot.getBumperSwitch().getState();
Sensor		robot.getGyro().getAngle();
Sensor		robot.getGyro().getRate();
Sensor		robot.getAccelerometer().getValue();
Sensor		.setValue(0);
Variables		item;
Variables		int var2 = 0;
Variables		var2 = 0;

Output		Console.print(0);
Output		robot.getLCD().setText(LCDLine.Top, "Hello World!");

Control Flow		for (int __index__1 = 0; __index__1 < 2; __index__1++) {
		}
		}

Control Flow		while (true) {}
Control Flow		if () {} else if (false) {} else {}

6.10. JAVA SAMPLE PROGRAMS

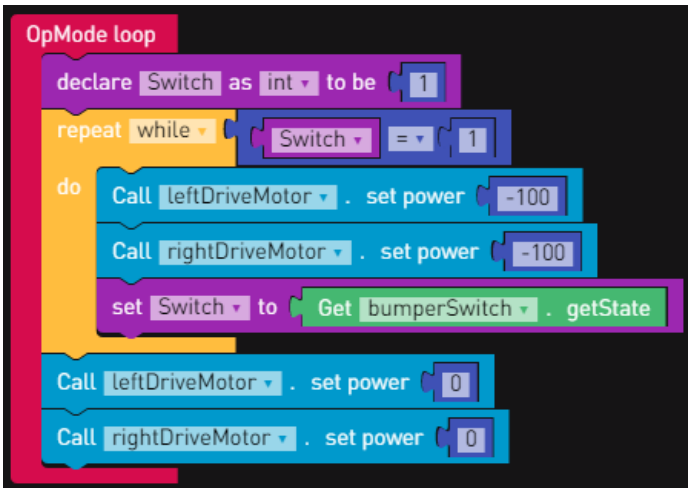
This section provides some sample programs that demonstrate the implementation of some common CoderZ Java functions.

Note: Only the loop() methods, that contain the robot behavior, are included below.

Program 1

Description: The state of the robot's Bumper Switch is written to a variable. While the Bumper Switch is not pressed, the robot reverses.

Blocks Program



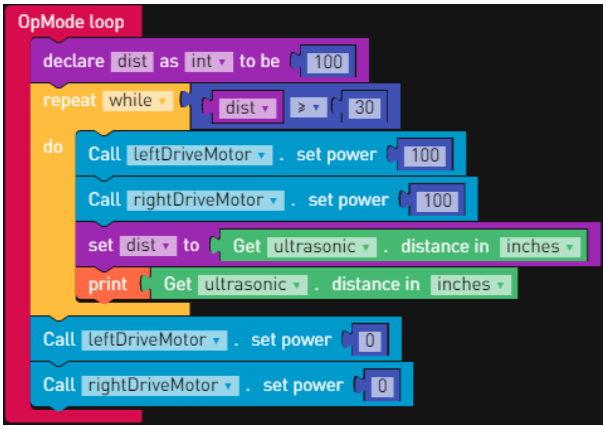
Equivalent C++ Program

```
public void loop() {
        int Switch = 1;
        while (Switch == 1) {
           robot.getLeftDriveMotor().setPower(-100);
           robot.getRightDriveMotor().setPower(-100);
           Switch = robot.getBumperSwitch().getState();
        }
        robot.getLeftDriveMotor().setPower(0);
        robot.getRightDriveMotor().setPower(0);
    }
```

Program 2

Description: The distance of an object from the robot's ultrasonic distance sensor is written to a variable. While the robot is more than 30 inches away from an object, the robot drives forward. Once it reaches 30 inches from an object, the robot stops.

Blocks Program



Equivalent C++ Program

```
public void loop() {
        int dist = 100;
        while (dist >= 30) {
          robot.getLeftDriveMotor().setPower(100);
          robot.getRightDriveMotor().setPower(100);
          dist = robot.getUltrasonic().getDistanceInches();

Console.print(robot.getUltrasonic().getDistanceInches());
        }
        robot.getLeftDriveMotor().setPower(0);
        robot.getRightDriveMotor().setPower(0);
    }
```

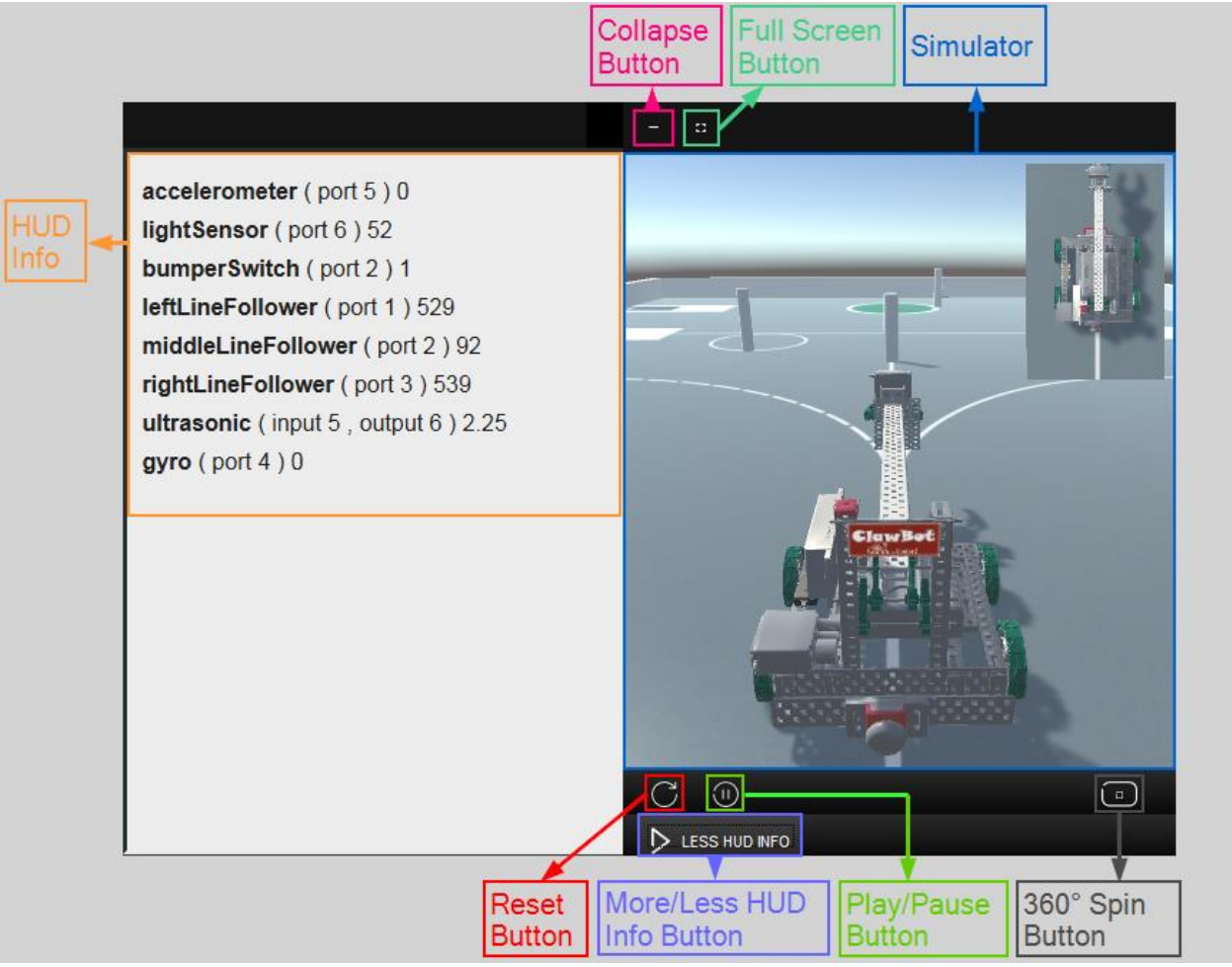
7. Project Interface Panes

This section describes the panes that are common to all project interfaces.

7.1. SIMULATION

The Simulation Pane contains a simulation of the robot as configured in the RobotConfigurator that can run programs.

7.1.1. Overview



This section describes the elements of the Simulation pane. Refer to the image above for the location of these elements.

The Simulation pane contains the following elements:

Name	Description
HUD Info	Displays sensor values. This area can be shown or hidden by clicking the More/Less HUD Info button.
Collapse Button	Collapses the Simulation pane.

Name	Description
Full Screen Button	Maximizes the Simulation pane and hides the Project Workspace area completely. To restore the Simulation pane to its original width, click the Full Screen button again.
Simulator	The Simulator contains the actual simulation of the robot. For information about controlling the Simulator, refer to section 7.1.2 Control, below. The Simulator scene can be changed using the RobotConfigurator. For more information, refer to section 7.2.1 Overview, below.
Reset Button	Resets the Simulator.
More/Less HUD Info button	Shows or hides the HUD Info area.
Play/Pause Button	Plays or pauses a program running in the Simulator.
360° Spin Button	Spins the Simulation view around the robot.

7.1.2. Control

This section describes the following control operations in the Simulator:

- Zoom
- Pan
- Reset

7.1.2.1. Zoom

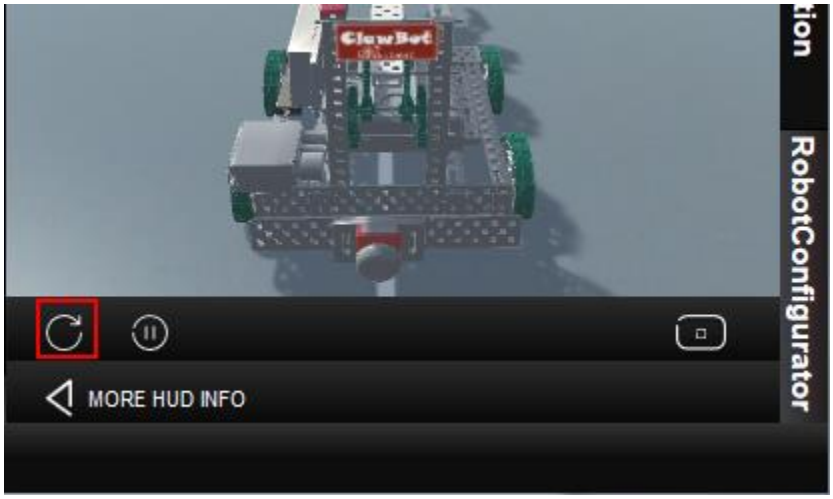
To zoom in or out in the Simulator, with the curser on the Simulator scroll the mouse scroll wheel up and down.

7.1.2.2. Pan

To pan in the Simulator, click in the Simulator and move your mouse around.

7.1.2.3. Reset

To reset the Simulator, click the Reset button.



7.1.3. HUD Info

The HUD Info area displays sensor values. It is useful to have the HUD Info area open and to pay attention to the sensor values while a program is running. It can help to fine-tune a program or to check why a program is not running as expected.

To show or hide the HUD Info area, click the More/Less HUD Info button.

7.2. ROBOTCONFIGURATOR

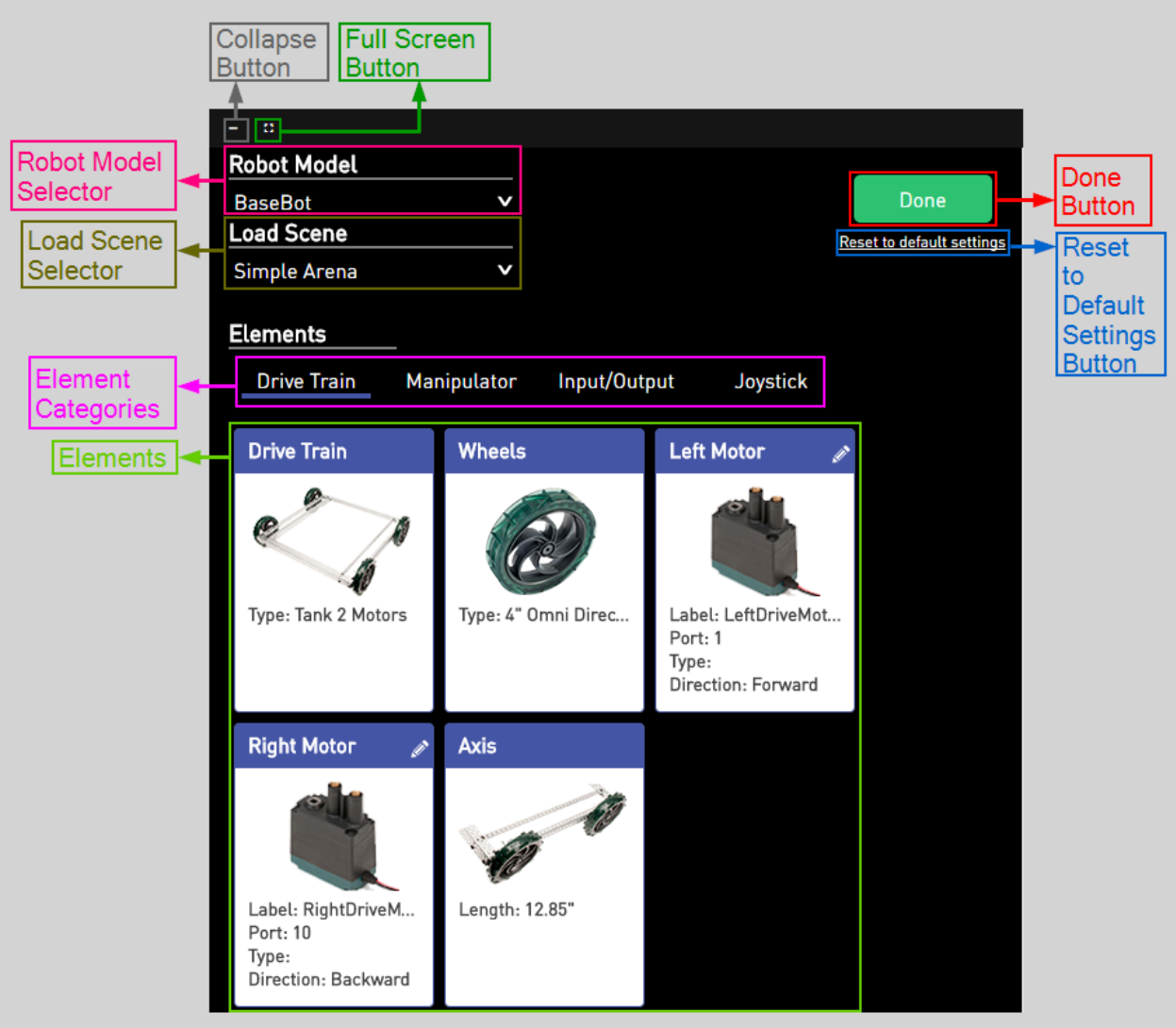
The default robot for a program is the BaseBot.

A project must be configured for your physical robot so that the program knows what elements it has and where relevant, what ports they are connected to. You can configure your robot using the RobotConfigurator.

This section describes the following:

- Overview
- Adding an Element
- Editing an Element
- Deleting an Element

7.2.1. Overview



This section describes the elements of the RobotConfigurator pane. Refer to the image above for the location of these elements.

The RobotConfigurator pane contains the following elements:

Name	Description
Collapse Button	Collapses the RobotConfigurator pane.
Full Screen Button	Maximizes the RobotConfigurator pane, while hiding the Project Workspace area completely. To restore the RobotConfigurator pane to its original width, click the Full Screen button again.
Robot Model Selector	Select your robot from the robot builds in the drop-down menu.
Load Scene Selector	Select a scene for the Simulation from the scenes in the drop-down menu.
Element Categories	Elements are divided by category. Click a category to display its elements.

Name	Description
Elements	Each Element Category's elements appear when the category name is clicked. Some elements have the following icons on the top-right:  • Click the trashcan icon to delete the element. • Click the pencil icon to configure the element's name and the port that it is connected to. Some Element Categories have an  icon beside its elements. Click the icon to add a new element.
Done Button	Saves the configuration.
Reset to Default Settings Button	Resets the configuration to its default.

7.2.2. Adding an Element

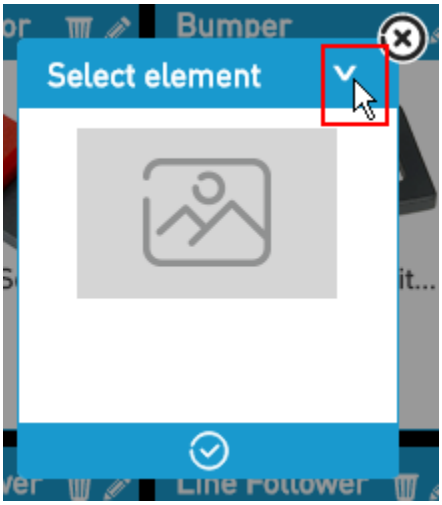
For some Element Categories you can add elements.

To add an element:

1. Click the  icon.

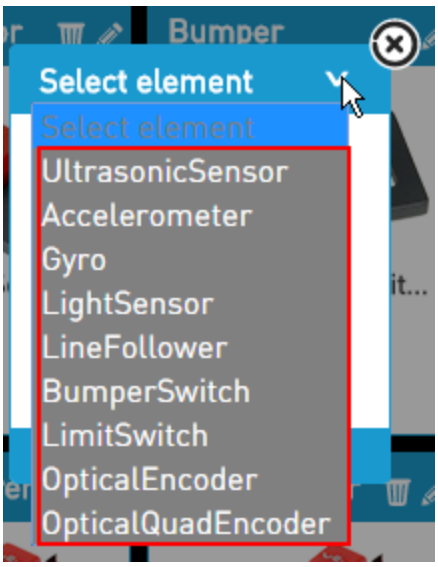
   The Select element window appears.

2. Click the ˅ icon.



   A drop-down list appears.

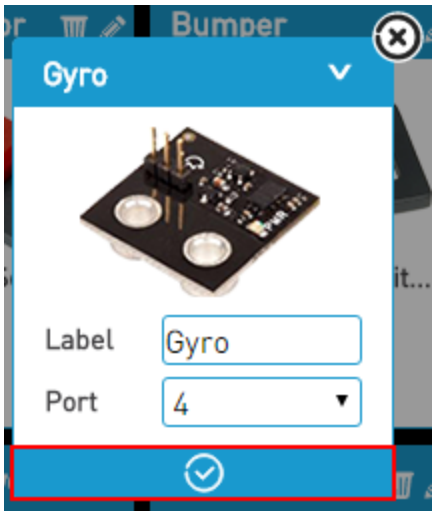
3. Select an element.



4. In the Label field, name the element.

5. In the Port field, select the port that the element is connected to on your physical robot.

6. Click the  icon.

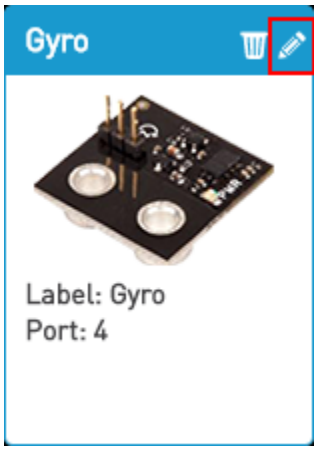


The element is added.

7.2.3. Editing an Element

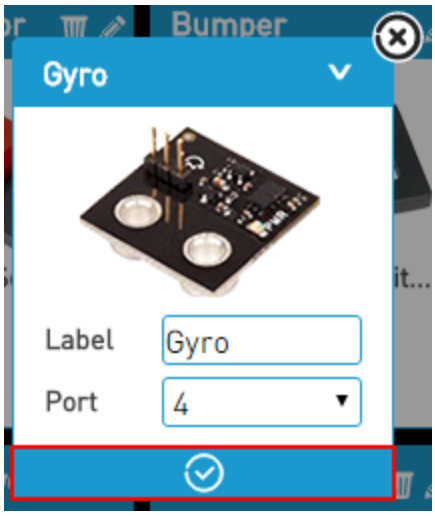
To edit an element:

1. On the top-right of an element, click the pencil icon.



2. Make the required changes to the element.

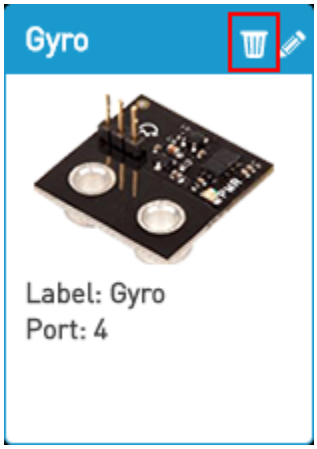
3. Click the  icon.



The changes are saved.

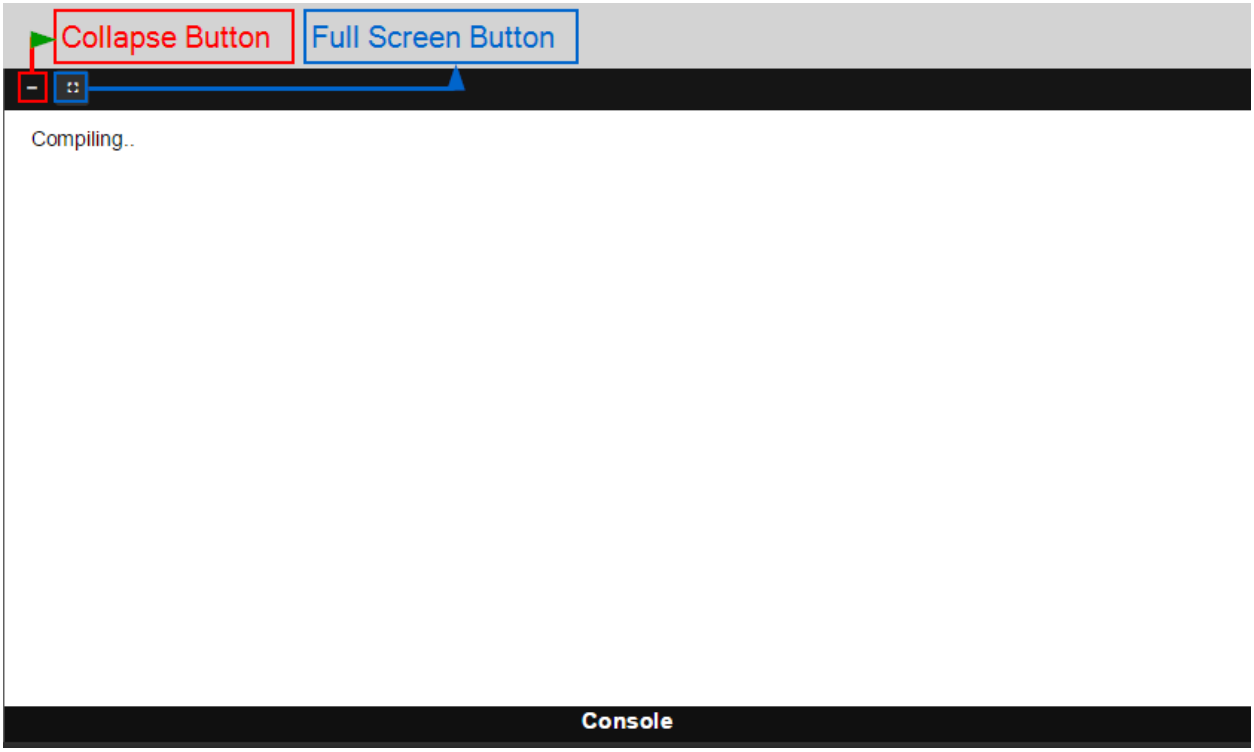
7.2.4. Deleting an Element

To delete an element, on the top-right of an element, click the trashcan icon.



7.3. CONSOLE

Program information including compilation details and program running details is written to the console.



This section describes the elements of the Console pane. Refer to the image above for the location of these elements.

The Console pane contains the following elements:

Name	Description
Collapse Button	Collapses the Console pane.
Full Screen Button	Maximizes the Console pane and hides the Project Workspace area completely. To restore the Console pane to its original size, click the Full Screen button again.

8. Downloading a Program to a Robot

Programs can be downloaded to a robot from any of the three project interfaces (Blocks, C++, Java).

To download a program to a robot:

1. Connect a robot to your PC. The robot does not need to be on.
2. Click the  Download to Robot button.

The program compiles and is downloaded to your robot.

The first time you download a program, you will be prompted to download an application called the Robot Downloader. Download and unzip the Zip file, and then run the setup to install the application.

9. Classes

9.1. CREATING A CLASS

9.2. DASHBOARD