JavaScript API

Overview

ARC JavaScript Overview

ARC provides the JavaScript programming environment for custom code that connects and coordinates robot skills. The ARC JavaScript API implements the ECMA-262 (ECMA 5.1) language features and adds a library of ARC-specific objects and functions for interacting with hardware, the framework, and other skills.

The engine exposes several built-in objects and root-level commands (documented below). At the end of this article we describe how to create global user-defined objects and methods.

About JavaScript

JavaScript (often abbreviated JS) is a high-level, just-in-time compiled, multi-paradigm language that conforms to the ECMA 5.1 specification. It uses curly-brace syntax, dynamic typing, prototype-based objects, and first-class functions. Although JavaScript and Java share a similar name and some syntactic elements, they are distinct languages with different designs and runtime behaviors.

JavaScript was originally used in web browsers, but modern JavaScript engines are embedded in many software systems — including Synthiam ARC — to provide scripting, automation, and integration capabilities.

ARC JavaScript API

ARC extends the standard ECMA 5.1 runtime with types and helper objects for common robot tasks: reading sensors, controlling servos and motors, accessing serial ports, working with files, networking, and more. You can use these objects directly from any script in ARC to interact with the system and other skills.

Using IntelliSense (Code Completion)

IntelliSense makes it easier to discover available classes and methods in the ARC JavaScript environment. To view built-in ARC methods:

1. Enable the IntelliSense checkbox in the code editor.
2. Type a class name, press the dot (.), and then scroll through the available methods and properties suggested by IntelliSense.

Code-completion (IntelliSense) suggestions for ARC JavaScript classes and methods.

Common ARC classes you can try with IntelliSense include:

- **ADC** – EZB analog-to-digital converter functions
- **Audio** – streaming audio functions
- **COM** – local serial COM port functions
- **Digital** – EZB digital I/O functions
- **EZB** – EZB-specific utilities (voltage, connection checks, etc.)
- **File** – reading and writing local files
- **I2C** – EZB I2C helper functions
- **Movement** – movement panel controls for directions
- **Net** – networking functions
- **Ping** – EZB ultrasonic distance sensor functions
- **PWM** – pulse-width modulation on digital I/O pins
- **Servo** – EZB servo movement functions
- **UART** – EZB hardware UART and software serial functions
- **Utility** – miscellaneous utility functions

Supported ECMA 5.1 Classes

ARC supports the ECMA 5.1 built-in objects for working with strings, numbers, arrays, and other core features. For detailed documentation and examples, refer to the linked pages on MDN or W3Schools.

- Array
- Boolean
- Console (supports time, count, error, info, log)
- Date
- Error
- Iterator
- JSON
- Map
- Math
- Number
- Object
- Proxy
- Reflect
- RegExp
- Set
- String
- Symbol

Root Commands

Root commands are global functions that do not belong to a specific class. They provide convenient access to ARC's global variable storage and to control other robot skills and UI elements.

getVar(variableName, [defaultValue])

Retrieve a value from ARC's public global variable storage. These variables are published by robot skills (Camera, Speech Recognition, Auto Position, etc.). If the variable does not exist, an optional default value can be returned.

Parameters

- `variableName` – string name of the global variable (e.g. "$Direction")

- `defaultValue` – optional value returned if the variable does not exist

Returns The stored value (or the default).

```
// Get the current direction the robot is moving
var direction = getVar("$Direction");

// Get the value of $test; if it doesn't exist, return false
var testVar = getVar("$test", false);
```

setVar(variableName, value)

Store a value in ARC's public global variable storage so other robot skills or scripts can access it via `getVar()`. Values passed with `setVar` may be serialized for storage.

Parameters

- `variableName` – string name of the global variable (e.g. "$MyValue")
- `value` – the value to store (string, number, array, etc.)

```
// Set a value of 5 to be accessible by other robot skills
setVar("$MyValue", 5);
```

setVarObject(variableName, value)

Store a JavaScript object in global storage without serialization translation so that full objects (including functions, classes or multidimensional arrays) are preserved for other scripts that retrieve them.

Parameters

- `variableName` – string name of the global variable
- `value` – the object to store

```
// Store a multidimensional array for other skills to access
var x = [[0, 1, 2], [3, 4, 5]];
setVarObject("$MyMatrix", x);
```

varExists(variableName)

Check whether a public global variable exists in ARC's global storage.

Parameters

- `variableName` – string name of the global variable

Returns boolean true if the variable exists, otherwise false.

```
// Print whether the variable exists
print(varExists("$MyValue"));
```

print(txt)

Write a value or message to the console output.

Parameters

- `txt` – the value or string to print

```
// Print a string to the console output
print("Hello World");
```

sleep(timeMs)

Pause script execution for the specified number of milliseconds.

Parameters

- `timeMs` – number of milliseconds to pause

```
// Print, pause for 5 seconds, then print again
print("Hello");
sleep(5000);
print(" World");
```

sleepRandom(minTimeMs, maxTimeMs)

Pause execution for a random duration between the specified minimum and maximum number of milliseconds.

Parameters

- `minTimeMs` – minimum time in milliseconds
- `maxTimeMs` – maximum time in milliseconds

```
// Print, pause for a random interval, then print again
print("Hello");
sleepRandom(1000, 5000);
print(" World");
```

controlCommand(controlName, command, [parameters])

Send a command to a named control or skill in the project. The control exposes commands that can be invoked by other scripts. The project cheat sheet lists available commands for interrogated controls.

```
// Start the Camera control and announce it
controlCommand("Camera", "StartCamera");
print("Camera has started");
```

showControl(controlName)

Display a user interface control and push it onto the display stack. This makes the specified control visible to the user.

```
// Show the Camera control
showControl("Camera");
```

closeControl(controlName)

Close the specified UI control and return to the previous control in the display stack. If no control name is provided, the currently displayed control will be closed.

```
// Close the active control
closeControl();
```

Variables and Scope

JavaScript variables declared in a script are private to the script engine's namespace. A variable created in one control’s JavaScript is not directly accessible from another control’s JavaScript. To share data between scripts and skills, use ARC’s global variable storage

functions (`getVar`, `setVar`, `setVarObject`).

By convention, public variables can begin with a dollar sign ($) to indicate they are meant for global access. Blockly generates JavaScript and keeps variables private by default; prefixing variable names with $ makes them public.

JavaScript Constants

ARC defines some numeric constants that are globally available in the JavaScript environment. These constants are declared automatically and are numeric values (they do not require quotes when referenced).

```
D0 ... D23
V0 ... V99
ADC0 ... ADC7
```

Custom JavaScript Extensions

ARC allows you to extend the runtime with custom objects, methods, or entire classes. You can add functions or objects that become available to your scripts. A practical example of creating custom JavaScript extensions is available in the Create Robot Skill manual.

See the example documentation: Create Robot Skill — Custom JavaScript Extension

Example screenshot demonstrating a custom JavaScript extension used in an ARC skill.

How To Program In JavaScript

Synthiam ARC lets you write JavaScript to control your robot and process data. Although JavaScript began as the language of the web, ARC uses it as a lightweight scripting language inside your project so you can read sensors, make decisions, and command robot skills.

Before you start

- **ARC has IntelliSense:** as you type, ARC shows suggestions to autocomplete functions, methods, and variables so you learn available features faster.
- **ARC uses `print()`:** use it to log messages and inspect variables so you can verify what your code is doing step-by-step.
- **JavaScript executes top-to-bottom:** your script runs in order unless you change flow with conditions, loops, or function calls.
- **Spacing is less important than braces:** in JavaScript, code blocks are defined by braces `{ }`; whitespace is mainly for readability.

---

1) Showing text with print()

When learning to program, the most important skill is being able to see what your code is doing. In ARC, `print()` writes messages to the output/log window so you can verify logic and follow execution.

Tip: If something isn't working, add more `print()` calls — print variable values and which branch or loop your code takes.

```
// Print a simple message
print("Hello from Synthiam ARC!");

// Print multiple pieces of information by building a string
var robotName = "JD";
print("Robot name is: " + robotName);
```

---

2) Variables

A **variable** is a labeled container for a value so you can reuse it later. In scripting you often use `var`. (Modern JavaScript also has `let` and `const`, but `var` keeps examples simple for ARC.)

2.1 String variables (text)

A **string** is text, enclosed in double or single quotes: `"like this"` or `'like this'`. Use strings for names, messages, or commands.

```
// Create a string variable
var greeting = "Hello!";

// Print it
print(greeting);

// Change it later (variables can be updated)
greeting = "Hi again!";
print(greeting);

// Combine strings using +
var firstName = "DJ";
var message = "Welcome, " + firstName + "!";
print(message);

// Strings can include numbers, but they remain text
var roomNumberText = "101";
print("Room (as text): " + roomNumberText);
```

Common beginner mistake: Storing `"10"` makes it text, not the number `10`. Text and numbers behave differently in comparisons and math.

2.2 Number variables

A **number** is for math: timers, distances, angles, speeds, percentages, etc. Numbers are written without quotes.

```
// Create number variables
var speed = 50;
var angle = 90;
var batteryVoltage = 7.4;

// Print numbers
print("Speed is: " + speed);
```

```
print("Angle is: " + angle);
print("Battery voltage is: " + batteryVoltage);

// Do math
var a = 10;
var b = 3;

print("a + b = " + (a + b));
print("a - b = " + (a - b));
print("a * b = " + (a * b));
print("a / b = " + (a / b));

// Update a variable using its current value
var count = 0;
count = count + 1;
count = count + 1;
print("Count is now: " + count);

// Shorthand updates
count += 5;  // same as count = count + 5
print("Count after += 5: " + count);
```

2.3 Boolean variables (true/false)

A **boolean** is either `true` or `false`. Booleans are used frequently in IF statements and loop conditions.

```
var isConnected = true;
var isMoving = false;

print("Connected? " + isConnected);
print("Moving? " + isMoving);
```

2.4 Null and undefined (empty / not set)

Sometimes you want a variable to represent "no value". JavaScript distinguishes:

- `null` — intentionally set to nothing.
- `undefined` — typically means a variable was never assigned a value.

```
var value; // not assigned yet -> undefined
print("value = " + value);

value = null; // intentionally empty
print("value = " + value);
```

---

3) Arrays (lists of values)

An **array** stores values in order. Think of it as boxes labeled 0, 1, 2, ... — the position is the **index**, and indexes start at **0**.

3.1 Creating arrays

```
// An array of numbers
var speeds = [10, 20, 30, 40];

// An array of strings
var phrases = ["Hello", "How are you?", "Goodbye"];

// An array can mix types (beginners should avoid mixing types until comfortable)
```

```
var mixed = [1, "two", true, null];

print("speeds: " + speeds);
print("phrases: " + phrases);
```

3.2 Accessing items by index

```
var colors = ["red", "green", "blue"];

print("First color (index 0): " + colors[0]);
print("Second color (index 1): " + colors[1]);
print("Third color (index 2): " + colors[2]);
```

Super common mistake: Assuming `colors[1]` is the first item. The first item is `colors[0]`.

3.3 Changing items and adding items

```
var numbers = [5, 10, 15];

// Change an existing item
numbers[1] = 999;
print("After change: " + numbers);

// Add an item to the end
numbers.push(20);
print("After push: " + numbers);

// Remove the last item
var last = numbers.pop();
print("Popped item: " + last);
print("After pop: " + numbers);

// How many items are in the array?
print("Count = " + numbers.length);
```

3.4 Looping through an array

To operate on every item, use a loop.

```
var distances = [12, 9, 7, 15];

for (var i = 0; i < distances.length; i++) {

  print("Item " + i + " is: " + distances[i]);
}
```

---

4) Objects (grouping related values)

An **object** stores related values under named properties — like fields on a form. Objects are extremely useful in robot scripting for grouping sensor readings, configuration, or state.

```
// Create an object describing a robot
var robot = {
  name: "JD",
  battery: 7.4,
  isOnline: true
};
```

```
print("Name: " + robot.name);
print("Battery: " + robot.battery);
print("Online: " + robot.isOnline);

// Update a property
robot.battery = 7.2;
print("Battery now: " + robot.battery);
```

Objects help keep related information together: sensor readings, current mode, last known position, configuration, and so on.

---

5) Conditions (IF statements)

An **IF statement** lets your script make decisions: "If this is true, do that; otherwise do something else."

5.1 Basic IF

```
var battery = 7.1;

if (battery > 7.3) {

  print("Battery is strong.");
}

if (battery <= 7.3) {

  print("Battery is getting low.");
}
```

5.2 IF / ELSE

```
var distanceCm = 25;

if (distanceCm < 20) {

  print("Obstacle is close! Stop or turn.");
} else {

  print("Path is clear enough.");
}
```

5.3 IF / ELSE IF / ELSE

```
var tempC = 22;

if (tempC < 10) {

  print("It's cold.");
} else if (tempC < 25) {

  print("It's comfortable.");
} else {

  print("It's hot.");
}
```

5.4 Comparisons you will use constantly

- `>` greater than
- `<` less than

- `>=` greater than or equal
- `<=` less than or equal
- `==` equal (performs type conversion — avoid when learning)
- `===` strict equal (recommended — compares value and type)
- `!=` not equal
- `!==` strict not equal

Beginner rule: Prefer `===` and `!==` because they avoid confusing type conversions.

```
var a = 10;
var b = "10";

print("a == b: " + (a == b));     // true (type conversion)
print("a === b: " + (a === b));   // false (different types)
```

5.5 Combining conditions with AND / OR

Use logical operators to require multiple rules.

```
var batteryOk = true;
var pathClear = false;

// AND: both must be true
if (batteryOk && pathClear) {

  print("Go forward.");
} else {

  print("Do not go forward (battery or path is not good).");
}

// OR: either can be true
var hasWifi = false;
var hasUart = true;

if (hasWifi || hasUart) {

  print("We have at least one connection method.");
} else {

  print("No connection available.");
}

// NOT: flips true/false
var isBlocked = true;

if (!isBlocked) {

  print("Not blocked.");
} else {

  print("Blocked.");
}
```

---

6) Loops (repeat code)

A **loop** repeats code. Loops are powerful but can create infinite execution if you forget an exit condition. While learning, always include a clear condition and use `print()` to monitor loop progress.

Loop safety tip: If you're not certain a loop will stop, add a counter "safety break" so it cannot run forever.

6.1 FOR loop (best for “repeat N times” or iterating an array)

A FOR loop has three parts: start, condition, and step.

```
// Print 0 through 4
for (var i = 0; i < 5; i++) {

  print("i = " + i);
}

// Count down
for (var n = 5; n > 0; n--) {

  print("Countdown: " + n);
}
print("Done!");
```

6.2 WHILE loop (best for “repeat until something changes”)

A WHILE loop repeats while its condition is true. If that condition never becomes false, the loop never ends.

```
var tries = 0;

while (tries < 3) {

  print("Try number: " + tries);
  tries++;
}

print("Finished trying.");
```

6.3 DO / WHILE loop (runs at least once)

Use DO/WHILE when you want the body to execute before checking the condition.

```
var x = 0;

do {

  print("This runs at least once. x=" + x);
  x++;
} while (x < 3);
```

6.4 break and continue

`break` exits the loop immediately. `continue` skips the remainder of the current iteration and continues with the next.

```
for (var i = 0; i < 10; i++) {

  if (i === 5) {

    print("Skipping 5");
    continue;
```

```
  }

  if (i === 8) {

    print("Stopping at 8");
    break;
  }

  print("i = " + i);
}
```

---

7) Functions (reusable code)

A **function** is a named block of code you call whenever needed. Functions reduce repetition and make scripts easier to read.

7.1 A simple function

```
function sayHello() {

  print("Hello!");
}

sayHello();
sayHello();
```

7.2 Function parameters (inputs)

Parameters let you pass information into a function so the same code can work with different values.

```
function greet(name) {

  print("Hello, " + name + "!");
}

greet("DJ");
greet("World");
```

7.3 Returning a value

Use `return` to send a result back to the caller.

```
function add(a, b) {

  return a + b;
}

var result = add(5, 7);
print("Result is: " + result);
```

7.4 Early exit with return

```
function divide(a, b) {

  if (b === 0) {

    print("Cannot divide by zero.");
```

```
    return 0;
  }

  return a / b;
}

print("10 / 2 = " + divide(10, 2));
print("10 / 0 = " + divide(10, 0));
```

---

8) Useful string tools

Strings can be measured, searched, and modified with built-in methods.

```
var text = "Synthiam ARC";

// Length
print("Length: " + text.length);

// Convert to uppercase/lowercase
print(text.toUpperCase());
print(text.toLowerCase());

// Check if it contains something
print("Contains 'ARC'? " + (text.indexOf("ARC") !== -1));

// Get a piece of the string
print("First 8 chars: " + text.substring(0, 8));
```

---

9) Useful number tools

```
// Rounding
var v = 3.14159;

print("Round: " + Math.round(v));
print("Floor: " + Math.floor(v));
print("Ceil: " + Math.ceil(v));

// Clamp a number (keep it within a range)
function clamp(value, min, max) {

  if (value < min) {

    return min;
  }

  if (value > max) {

    return max;
  }

  return value;
}

print("Clamp 150 to 0..100: " + clamp(150, 0, 100));
```

---

10) Reading errors and avoiding common mistakes

10.1 Missing quotes around text

```
// WRONG: Hello is treated like a variable name (and will likely cause an error)
 // var msg = Hello;
```

```
// RIGHT:
var msg = "Hello";
print(msg);
```

10.2 Forgetting to update the loop variable

```
// DANGEROUS: If you forget tries++, this can loop forever.
var tries = 0;

while (tries < 3) {

  print("tries=" + tries);
  tries++;
}
```

10.3 Using the wrong equality operator

```
var n = 5;

// Use === for clarity
if (n === 5) {

  print("n is exactly 5");
}
```

---

11) Debugging strategy in ARC (the practical way)

When your script doesn't behave as expected, follow a simple debugging process:

1. **Print where you are:** add `print("Reached step 1")`, `print("Reached step 2")`, etc. to trace flow.
2. **Print values:** log key variables before IF statements and inside loops.
3. **Simplify:** temporarily remove robot commands and test logic with dummy values.
4. **Change one thing at a time:** when multiple changes are made at once, it's hard to know what caused a regression.

```
// Example debugging pattern
var distanceCm = 18;

print("distanceCm=" + distanceCm);

if (distanceCm < 20) {

  print("Decision: obstacle is close");
} else {

  print("Decision: path is clear");
}
```

---

12) How ARC IntelliSense helps you

ARC's IntelliSense (autocomplete) is valuable while learning because:

- It shows function and property names so you don't have to memorize everything.
- It reduces typos, which are a major source of errors for beginners.
- It often displays parameter hints so you know what values a function expects.

Tip: When you see a dropdown while typing, use arrow keys to pick an item and press

Enter or Tab to insert it. This speeds learning and reduces mistakes.

---

13) Putting it together: “real script” style examples

The examples below show practical patterns you might use in ARC: configuration and state at the top, decisions, loops, and helper functions. Adapt these to your robot-specific ARC functions and skills.

13.1 Example: basic state + decisions

```
// Configuration
var minSafeDistanceCm = 20;

// State
var isRunning = true;
var lastDistanceCm = 0;

// Pretend we read a sensor (replace this with your ARC sensor call)
lastDistanceCm = 18;

print("Last distance: " + lastDistanceCm);

if (lastDistanceCm < minSafeDistanceCm) {

  print("Obstacle detected. Choose a different action.");
  isRunning = false;
} else {

  print("Safe to continue.");
}
```

13.2 Example: loop through a list of steps

```
// A simple list of "steps" you want to run
var steps = ["Scan", "MoveForward", "Stop", "Speak"];

// Run each step in order
for (var i = 0; i < steps.length; i++) {

  var step = steps[i];
  print("Running step: " + step);

  // Decision per step
  if (step === "Scan") {

    print("Pretend: scanning sensors...");
  } else if (step === "MoveForward") {

    print("Pretend: moving forward...");
  } else if (step === "Stop") {

    print("Pretend: stopping...");
  } else if (step === "Speak") {

    print("Pretend: speaking...");
  } else {

    print("Unknown step: " + step);
  }
}
```

13.3 Example: while loop with safety limit

```
// Repeat until a condition changes, but don't run forever
```

```
var attempts = 0;
var maxAttempts = 10;

var targetFound = false;

while (!targetFound && attempts < maxAttempts) {

  print("Searching... attempt " + attempts);

  // Pretend condition changes (replace with real sensor/vision results)
  if (attempts === 3) {

    targetFound = true;
    print("Target found!");
  }

  attempts++;
}

if (!targetFound) {

  print("Target not found after " + maxAttempts + " attempts.");
}
```

---

14) Quick reference (copy/paste friendly)

```
// String
var s = "text";

// Number
var n = 123;

// Boolean
var ok = true;

// Array
var arr = [1, 2, 3];
arr.push(4);
var first = arr[0];

// Object
var obj = { name: "JD", speed: 50 };
var name = obj.name;

// IF
if (n > 100) {

  print("big");
} else {

  print("small");
}

// FOR
for (var i = 0; i < 5; i++) {

  print(i);
}

// WHILE
var x = 0;
while (x < 3) {

  print(x);
  x++;
}

// Function
```

```
function hello(who) {

  print("Hello " + who);
}
hello("ARC");
```

---

15) Error handling with try / catch

When your script interacts with hardware, sensors, network connections, or robot skills, operations can fail. A **try / catch** block lets you handle those errors gracefully so a single failure doesn't stop the entire script.

Key idea: Put code that *might fail* inside `try`. If an error occurs, execution jumps immediately to `catch`.

15.1 Basic try / catch structure

The simplest form has two parts: `try` for the risky code and `catch` for error handling.

```
try {

  print("About to run risky code...");

  // Example: something goes wrong
  var result = unknownVariable + 1;

  print("This line will not run if an error happens above.");

} catch (ex) {

  print("An error occurred.");
  print("Error message: " + ex);
}
```

If an error occurs, JavaScript skips the rest of the `try` block and runs the `catch` block.

15.2 Why try / catch is important in ARC scripts

ARC scripts commonly interact with:

- Robot skills that may not be loaded
- Hardware that may be disconnected
- Sensors that may return invalid values temporarily
- Network connections that can drop or time out

Without `try / catch`, a single unexpected error can stop the script. With it, you can log the problem and continue safely.

15.3 Catching errors and continuing safely

A common pattern is to catch the error, log useful information, and use a safe fallback value.

```
var distanceCm = 0;

try {

  // Replace this with a real ARC sensor call
  distanceCm = GetADC(0);

  print("Distance read: " + distanceCm);

} catch (ex) {

  print("Failed to read distance sensor.");
  print("Reason: " + ex);

  // Safe fallback value
  distanceCm = 999;
}

if (distanceCm < 20) {

  print("Obstacle detected.");
} else {

  print("No obstacle detected.");
}
```

15.4 Using try / catch inside loops

When a loop runs many times, keep the failure localized to a single iteration so the loop can continue.

```
for (var i = 0; i < 5; i++) {

  try {

    print("Loop iteration: " + i);

    // Example of a risky operation
    var value = readSensorValue(i);
    print("Sensor value: " + value);

  } catch (ex) {

    print("Error on iteration " + i);
    print("Details: " + ex);

    // Continue with next iteration
  }
}
```

Best practice: Keep the `try` block as small as possible and only wrap lines that can fail. That makes bugs easier to find.

15.5 Distinguishing different error situations

The error object in `catch` often contains a message. Inspect it to handle different problems differently.

```
try {

  ControlCommand("Some Skill", "DoSomething");
```

```
} catch (ex) {

  var msg = ex.toString();

  if (msg.indexOf("not found") !== -1) {

    print("The robot skill is not loaded.");
  } else if (msg.indexOf("disconnected") !== -1) {

    print("Hardware is disconnected.");
  } else {

    print("Unknown error occurred:");
    print(msg);
  }
}
```

15.6 try / catch with functions

Place `try / catch` inside a function to protect reusable logic and keep callers simple.

```
function safeDivide(a, b) {

  try {

    if (b === 0) {

      throw "Division by zero";
    }

    return a / b;

  } catch (ex) {

    print("safeDivide error: " + ex);
    return 0;
  }
}

print("10 / 2 = " + safeDivide(10, 2));
print("10 / 0 = " + safeDivide(10, 0));
```

15.7 finally (always runs)

An optional `finally` block runs whether an error occurred or not — useful for cleanup.

```
try {

  print("Starting operation...");
  // Risky operation here

} catch (ex) {

  print("Operation failed: " + ex);

} finally {

  print("Cleanup or final steps always run.");
}
```

15.8 Common beginner mistakes with try / catch

- Wrapping the entire script in one huge `try` block — this makes debugging harder.
- Ignoring the error message instead of printing it.

- Using `try / catch` to hide logic bugs rather than fixing the root cause.

Rule of thumb: Use `try / catch` for unexpected runtime failures, not as a substitute for proper IF checks and validation.

15.9 Debugging with try / catch

When debugging, temporarily add more detail to `catch` blocks so you can inspect the error and continue testing.

```
try {

  print("Before risky code");
  riskyOperation();
  print("After risky code");

} catch (ex) {

  print("ERROR CAUGHT");
  print(ex);
  print("Script continued safely.");
}
```

Using `try / catch` correctly makes Synthiam ARC scripts more robust, safer around hardware, and easier to debug when things go wrong.

Next steps: After mastering these basics, learn the ARC-specific functions available in your project (robot skills, control commands, sensors, speech, movement, etc.). Use IntelliSense to explore available methods and `print()` to verify your script is doing exactly what you expect.

.Net CLR

Accessing the .NET CLR from ARC JavaScript

For advanced users, the .NET Common Language Runtime (CLR) is available to the ARC JavaScript engine. This lets you access the underlying ARC platform and operating system directly from JavaScript, including .NET classes, namespaces, and system services.

Use this capability when you need functionality beyond the built-in ARC APIs, but be mindful of permissions, resource management, and thread/latency considerations.

What is the CLR?

The Common Language Runtime (CLR) is the run-time environment provided by the .NET Framework. It executes code and provides services that simplify application development and execution. Code compiled to target the CLR is called managed code and benefits from runtime services that improve reliability, security, and interoperability.

Key benefits of managed code

- Cross-language integration and consistent exception handling across languages.
- Enhanced security and access control enforced by the runtime.
- Versioning and simplified deployment support.
- Component interaction model that reduces integration complexity.
- Built-in debugging, profiling, and memory management services.

.NET Framework Example

The example below demonstrates using the System.IO.StreamWriter class to write text to a log file on C:\. ARC's JavaScript engine already provides file-writing helpers; this example shows direct access to raw CLR classes and namespaces.

```
// Create an instance of the StreamWriter
var file = new System.IO.StreamWriter('c:\\log.txt');

// Write some text to the file
file.WriteLine('Hello World !');

// Close the file and dispose the object
file.Dispose();
```

Tips: Always dispose or close file and resource objects to release handles. Ensure ARC has permission to write to the target path, and consider using application-specific paths instead of root-level directories.

EZ_B Driver Example

The ARC JavaScript engine exposes the raw EZ_B driver for each connected EZ-B controller. In robot skills this collection is available as ARC.EZBManager.EZBs; in many contexts you can access it simply as EZBs. The example below moves a servo on port D2 of EZB #0 to position 20.

```
// Move the servo on port D2 to position 20 on EZB #0
EZBs[0].Servo.SetServoPosition(2, 20);
```

Note: Verify the correct EZB index and servo port for your configuration before issuing commands. Use safety checks to avoid unexpected motion.

Best practices

- Dispose CLR objects (Dispose or using patterns) to free resources promptly.
- Check file and system permissions before performing I/O operations.
- Avoid long-running or blocking calls on the main thread; use asynchronous patterns if available.
- Test CLR interactions thoroughly to ensure stability and compatibility with ARC.

Exception Handling

When running JavaScript, various runtime errors can occur. By default, JavaScript halts execution and reports an error when a problem is encountered. Errors may arise from programmer mistakes, invalid input, or unpredictable issues such as exceptions thrown by ControlCommand() when controlling other robot skills. Using try...catch (and optionally finally) allows you to handle these exceptions gracefully so your application or robot skill can

respond or recover.

Throw, and Try...Catch...Finally

The try, catch, finally, and throw statements provide structured error handling in JavaScript:

- **try** — Defines a block of code to execute and test for errors. If an exception occurs inside this block, control passes to the associated catch block.
- **catch** — Defines a block that handles any exception thrown in the try block. The catch block receives the thrown value (commonly an Error object) so you can inspect or log details.
- **finally** — Defines a block that always runs after try and catch, regardless of whether an error occurred. Use finally for cleanup tasks (closing resources, stopping motors, releasing locks, etc.).
- **throw** — Allows you to create and raise a custom exception (a string, Error object, or any value). Use throw to signal conditions that should be handled by higher-level code.

Example flow: try to execute a ControlCommand(); if it fails, catch can log the error or attempt a fallback; finally can perform cleanup actions such as resetting variables or stopping hardware safely.

The Error Object

JavaScript provides a built-in Error object that delivers information about runtime problems. When an exception is thrown as an Error (or one of its subclasses), the error value typically exposes two useful properties:

- **name** — The type or name of the error (for example, "TypeError").
- **message** — A descriptive message explaining the error (a string).

In addition, Error objects may include other properties such as `stack` in many environments, which helps with debugging by showing the call stack.

Error Object Properties

Property	Description
`name`	Sets or returns the error name (for example, `"ReferenceError"`).
`message`	Sets or returns the error message (a string describing the error).

Error Name Values

The `name` property commonly indicates one of several built-in error types. These are the standard names you may encounter:

Error Name	Description
EvalError	An error related to the `eval()` function (rare in modern code).
RangeError	A numeric value is outside an allowable range (for example, invalid array length).
ReferenceError	An invalid reference was used (such as accessing an undefined variable).
SyntaxError	Code contains a syntax error and cannot be parsed.
TypeError	An operation was performed on a value of an inappropriate type.
URIError	An error occurred during URI encoding/decoding (for example, malformed percent-encoding).

Example #1

This example forces an error using `throw` and demonstrates a simple try/catch. In a robot skill, you might replace `print(err)` with a log, an alert, or a fallback command.

```
try {
  throw "this is an error";
} catch (err) {
  print(err);
} finally {
  // Optional cleanup code here
}
```

Examples

Parse JSON

The JSON javascript class is included in the ARC javascript compiler. You can parse the JSON in javascript directly in ARC.

Here's an example parsing JSON from a string...

Code:

```
var resp = '{ "glossary": { "title": "example glossary" } }';var myObj =
JSON.parse(resp);print (myObj.glossary.title);
```

And here's an example of getting JSON from an HTTP get request...

Code:

```
var resp = Net.hTTPGet("https://ip.jsontest.com/"); var myObj = JSON.parse(resp);print
(myObj.ip);
```

ADC

get12Bit

```
ADC.get12Bit(port, [ezbIndex])
```

Parameters

port	Analog port to read from.
ezbIndex (optional)	Board index of the EZB to read from.

Returns
The value read from the specified ADC port as a 12 bit number.

Description
Returns the value read from the specified ADC port as a 12 bit number which ranges between 0 and 4095.

waitForBetween

```
ADC.waitForBetween(port, low, high, [frequencyMs], [ezbIndex], [timeoutMS])
```

Parameters

port	Analog port to read from.
low	Inclusive lower bound on value to wait for.
high	Inclusive upper bound on value to wait for.
frequencyMs (optional)	How often the ADC port is checked in milliseconds.
ezbIndex (optional)	Board index of the EZB to read from.
timeoutMS (optional)	Number of milliseconds to wait before timeout.

Returns
The value read from the specified ADC port that is between or low and high. Returns -1 if timeout.

Description
Suspends execution of the script until the value read from the specified ADC port is between low (inclusive) and high (inclusive). The value read from the ADC port ranges between 0 and 255. If frequencyMs is provided the port is checked every frequencyMs milliseconds. Otherwise the port is checked as often as possible.

waitForEquals

```
ADC.waitForEquals(port, value, [frequencyMs], [ezbIndex], [timeoutMS])
```

Parameters

port	Analog port to read from.
value	Value to wait for.
frequencyMs (optional)	How often the port is checked in milliseconds.

ezbIndex (optional)	Board index of the EZB to read from.
timeoutMS (optional)	Number of milliseconds to wait before timeout.

Returns
The value read from the specified ADC port that is equal to value. Returns -1 if timeout.

Description
Suspends execution of the script until the value read from the specified ADC port is equal to value. The value read from the ADC port ranges between 0 and 255. If frequencyMs is provided the port is checked every frequencyMs milliseconds. Otherwise the port is checked as often as possible.

waitForHigher

```
ADC.waitForHigher(port, value, [frequencyMs], [ezbIndex], [timeoutMS])
```

Parameters

port	Analog port to read from.
value	Exclusive lower bound on value to wait for.
frequencyMs (optional)	How often the port is checked in milliseconds.
ezbIndex (optional)	Board index of the EZB to read from.
timeoutMS (optional)	Number of milliseconds to wait before timeout.

Returns
The value read from the specified ADC port that higher than value. Returns -1 if timeout

Description
Suspends execution of the script until the value read from the specified ADC port is greater than value. The value read from the ADC port ranges between 0 and 255. If frequencyMs is provided the port is checked every frequencyMs milliseconds. Otherwise the port is checked as often as possible.

waitForLower

```
ADC.waitForLower(port, value, [frequencyMs], [ezbIndex], [timeoutMS])
```

Parameters

port	Analog port to read from.
value	Exclusive upper bound on value to wait for.
frequencyMs (optional)	How often the port is checked in milliseconds.
ezbIndex (optional)	Board index of the EZB to read from.
timeoutMS (optional)	Number of milliseconds to wait before timeout.

Returns
The value read from the specified ADC port that is lower than value. Returns -1 if timeout

Description
Suspends execution of the script until the value read from the specified ADC port is lower than value. The value read from the ADC port ranges between 0 and 255. If frequencyMs is provided the port is checked every frequencyMs milliseconds. Otherwise the port is checked as often as possible.

get

```
ADC.get(port, [ezbIndex])
```

Parameters

port	Analog port to read from.
ezbIndex (optional)	Board index of the EZB to read from.

Returns

The value read from the specified ADC port.

Description

Returns the value read from the specified ADC port. The value read from the ADC port ranges between 0 and 255.

Audio

getVolume

```
Audio.getVolume([ezbIndex])
```

Parameters

ezbIndex (optional)	Board index of the EZB to get volume level from.

Returns
The volume level of the specified EZB.

Description
Returns the volume level of the specified EZB. The volume level ranges between 0 (quite), 100 (loud), 200 (2x over drive).

isConnected

```
Audio.isConnected([ezbIndex])
```

Parameters

ezbIndex (optional)	Board index of the EZB to check if audio is connected.

Returns
True if audio is connected to the EZB. False otherwise.

Description
Returns true if audio is connected to the EZB. Returns false otherwise.

playAudioFile

```
Audio.playAudioFile(filename)
```

Parameters

filename	Filename of the audio file to play.

Returns
Nothing

Description
Plays the audio file at the location given by filename through the computer's audio system.

say

```
Audio.say(txt)
```

Parameters

txt	Text to say.

Returns
Nothing

Description
Speak the text given in the string txt using text-to-speech through the computer's audio system. Execution of the script will continue while the audio is playing.

saySSMLEZB

```
Audio.saySSMLEZB(ssml, [ezbIndex])
```

Parameters

ssml	Text to say.
ezbIndex (optional)	Board index of the EZB to output the audio from.

Returns
Nothing

Description
Speak the SSML document given in the string using text-to-speech through the EZB's audio system. Execution of the script will continue while the audio is playing.
Read about SSML in this support document here: https://www.w3.org/standards/history/speech-synthesis/

Example

```
// This is an example using the say-as to specify an input format and speak the date
var str = "<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-US">" +
          "<say-as type="date:mdy"> 1/29/2009 </say-as>" +
          "</speak>";

Audio.saySSMLEZB(str);
```

```
// This example includes a pause of 500ms between the two words hello and there
var str = "<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-US">" +
          "hello <break time="500ms"> there" +
          "</break></speak>";

Audio.saySSMLEZB(str);
```

saySSMLEZBWait

```
Audio.saySSMLEZBWait(ssml, [ezbIndex])
```

Parameters

ssml	Text to say.
ezbIndex (optional)	Board index of the EZB to output the audio from.

Returns
Nothing

Description
Speak the SSML document given in the string using text-to-speech through the EZB's audio system. Execution of the script will be suspended until the audio is finished playing.
Read about SSML formatted speech here: https://www.w3.org/standards/history/speech-synthesis/

Example

```
// This is an example using the say-as to specify an input format and speak the date
var str = "<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-US">" +
          "<say-as type="date:mdy"> 1/29/2009 </say-as>" +
          "</speak>";
```

```
Audio.saySSMLEZBWait(str);
```

```
// This example includes a pause of 500ms between the two words hello and there
var str = "<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-
US">" +
          "hello <break time="500ms"> there" +
          "</break></speak>";

Audio.saySSMLEZBWait(str);
```

saySSMLWait

```
Audio.saySSMLWait(ssml)
```

Parameters

ssml	Text to say.

Returns
Nothing

Description
Speak the text given in the provided SSML using text-to-speech through the computer's audio system. Execution of the script will be suspended until the audio is finished playing. Read about how to format SSML in this document here: https://www.w3.org/standards/history/speech-synthesis/

Example

```
// This is an example using the say-as to specify an input format and speak the date
var str = "<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-
US">" +
          "<say-as type="date:mdy"> 1/29/2009 </say-as>" +
          "</speak>";

Audio.saySSMLWait(str);
```

```
// This example includes a pause of 500ms between the two words hello and there
var str = "<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-
US">" +
          "hello <break time="500ms"> there" +
          "</break></speak>";

Audio.saySSMLWait(str);
```

setVolume

```
Audio.setVolume(volume, [ezbIndex])
```

Parameters

volume	Level to set the volume to.
ezbIndex (optional)	Board index of the EZB to set the volume level for.

Returns
Nothing

Description
Set the volume level of the EZB to volume. The volume level ranges between 0 (quite), 100 (loud), 200 (2x over drive).

soundNote

```
Audio.soundNote(note, length, [signalType], [ezbIndex])
```

Parameters

note	Note to play as a string from "C1" to "Bb3".
length	How long to play the note for in milliseconds.
signalType (optional)	Type of signal to use when playing the note as a string ("Sine"
ezbIndex (optional)	Board index of the EZB to output the audio from.

Returns
Nothing

Description
Plays the specified note for length milliseconds. Notes are inputted as strings which range from "C1" to "Bb3". The note is played using the signal specified by signalType. If signalType is not provided a sine wave is used. Accepted signal types are "Sine", "Square", "Triangle", "Pulse", "Sawtooth", "WhiteNoise", "GaussNoise", "DigitalNoise". This function does not wait for the note to end to continue executing the script. In case of playing multiple notes in sequence, call the sleep() function to ensure the notes don't play over previous notes.

speakVolume

```
Audio.speakVolume(volume)
```

Parameters

volume	Level to set the text-to-speech volume to.

Returns
Nothing

Description
Sets the text-to-speech volume level of the computer to volume. The volume level ranges between 0 and 100.

stop

```
Audio.stop([ezbIndex])
```

Parameters

ezbIndex (optional)	Board index of the EZB to stop playing audio.

Returns
Nothing

Description
Stops all audio currently playing on the EZB.

stopAudioFile

```
Audio.stopFile()
```

Returns
Nothing

Description
Stops the playback of the audio file started by playAudioFile(filename) that is playing through the computer's audio system.

waitForSpeech

```
Audio.waitForSpeech(timeout, s1, ...)
```

Parameters

timeout	Time to wait in seconds before stopping waiting for speech.
s1...	Strings to wait for that are detected in the microphone.

Returns
The phrase heard if the timeout did not occur in a lowercase string.

Description
Suspends execution of the script until one of the strings specified by s1,.... is detected in the microphone, or a timeout occurs after timeout seconds. If a timeout occurs, the method will return "timeout".
***Note: the returned string is in lowercase**

Example

```
// Specify the words in the function
response = Audio.waitForSpeech(10, "One", "two", "Three");

print("Heard: " + response);
```

```
// Use an array of words to listen for
var words = ["One", "two", "Three"];

response = Audio.waitForSpeech(10, words);

print("Heard: " + response);
```

setVolumePC

```
Audio.setVolumePC(volume)
```

Parameters

volume	Level to set the PC speaker volume to between 0-255

Returns
Nothing

Description
Set the volume level of the default audio output device (Soundcard). The volume level ranges between 0 and 255. With 0 (mute) and 255 is loudest

waitForSpeechRange

```
Audio.waitForSpeechRange(timeout, start, end, [increment])
```

Parameters

timeout	Time to wait in seconds before stopping waiting for speech.
start	numeric start value of the range
end	numeric end value of the range
increment	the increment of the range (optional)

Returns

The value within the range as a number or the word "timeout" if timeout is reached.

Description

Suspends execution of the script until one of the numbers within the range (start, end) is detected in the microphone by speech, or a timeout occurs after timeout seconds. If a timeout occurs, the method will return "timeout."
The returned value is a number (i.e., 23, 10, 5) and not a string (i.e., twenty-three, ten, five).

Each option will be displayed if the number of options within the range is less than 10. For example, if the start is 10 and the end is 20, there will be 10 options for each option to be displayed.

If there are more than ten options, the range will be displayed.

Large Number Ranges

This style of speech recognition populates the pre-defined responses it expects to hear. In this case, a range of numbers.
This dramatically increases the accuracy of the speech recognition system by limiting the detected phrase/words to the list.
By providing the list, the speech recognition system will generate definitions of the sound waveforms it is looking for.

While this method dramatically improves detection accuracy, it also can consume a lot of PC resources if an extensive number range is used.
For example, a range higher than 100 is considered an extensive range. You may notice a significant delay every time this function is called, and that is due to the vast range.

A recommended solution is to use the waitForAnyNumberSpeech() command.

An additional solution is to prompt for multiple digits that will generate a more significant number when added together with simple math. For example, if you require 1000 positions, consider prompting for each digit.

Example

```
// Get a number from the user within the range between 10 and 20
response = Audio.waitForSpeechRange(10, 10, 20);
Audio.say("You selected " + response);
```

```
// Get a number from the user within the range between 10 and 20 incremented by 2 (every even number)
response = Audio.waitForSpeechRange(10, 10, 20, 2);
Audio.say("You selected " + response);
```

waitForAnySpeech

```
Audio.waitForAnySpeech(timeout, prompt)
```

Parameters

timeout	Time to wait in seconds before stopping waiting for speech.
prompt	The text prompt to display to the user while listening for speech

Returns
The phrase heard if the timeout did not occur in a lowercase string.

Description
Suspends execution of the script until any speech is detected in the microphone or a timeout occurs after timeout seconds. If a timeout occurs, the method will return "timeout."
***Note: the returned string is in lowercase**
This command uses the entire dictionary of speech recognition words.
Generally, using this dictionary size will have a very low recognition quality. This may work okay if you are feeding a chatbot (such as OpenAI or PandoraBot).
But, if you are looking for specific phrases, this will most likely not work, and you should use the waitForSpeech() command instead.
Using this command for open conversations is discouraged.

Example

```
// Listen for any response from the user
response = Audio.waitForAnySpeech(30, "Say anything to me");

print("Heard: " + response);
```

waitForAnyNumberSpeech

```
Audio.waitForAnyNumberSpeech(timeout, prompt)
```

Parameters

timeout	Time to wait in seconds before stopping waiting for speech.
prompt	The text prompt to display to the user while listening for speech

Returns
The number heard if the timeout did not occur in a lowercase string.

Description
Suspends execution of the script until any number is detected in the microphone or a timeout occurs after timeout seconds. If a timeout occurs, the method will return "timeout" in lowercase. The number can include a minus sign and a decimal point.

Example

```
// Listen for any response from the user
response = Audio.waitForAnyNumberSpeech(30, "Give me a number");
```

```
print("Heard: " + response);
```

saySSML

```
Audio.saySSML(ssml)
```

Parameters

ssml	Text to say.

Returns
Nothing

Description
Speak the formatted SSML given using text-to-speech through the computer's audio system. Execution of the script will continue while the audio is playing.
This support document explains SSML here: https://www.w3.org/standards/history/speech-synthesis/

Example

```
// This is an example using the say-as to specify an input format and speak the date
var str = "<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-
US">" +
          "<say-as type="date:mdy"> 1/29/2009 </say-as>" +
          "</speak>";

Audio.saySSML(str);
```

```
// This example includes a pause of 500ms between the two words hello and there
var str = "<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-
US">" +
          "hello <break time="500ms"> there" +
          "</break></speak>";

Audio.saySSML(str);
```

sayEZB

```
Audio.sayEZB(txt, [ezbIndex])
```

Parameters

txt	Text to say.
ezbIndex (optional)	Board index of the EZB to output the audio from.

Returns
Nothing

Description
Speak the text given in the string txt using text-to-speech through the EZB's audio system. Execution of the script will continue while the audio is playing.

sayEZBWait

```
Audio.sayEZBWait(txt, [ezbIndex])
```

Parameters

txt	Text to say.
ezbIndex(optional)	Board index of the EZB to output the audio from.

Returns
Nothing

Description
Speak the text given in the string txt using text-to-speech through the EZB’s audio system. Execution of the script will be suspended until the audio is finished playing.

COM

available

```
COM.available(port)
```

Parameters

port	COM port name as a string.

Returns
True if the specified COM port is available. False otherwise.

Description
Returns true if the specified COM port is available. Returns false otherwise. port is provided as a string (e.g. “COM3”).

availablePorts

```
COM.availablePorts()
```

Returns
Array of available COM ports.

Description
Returns an array of available COM ports. Available COM ports are represented in the array as strings (e.g. “COM3”).

clearInputBuffer

```
COM.clearInputBuffer(port)
```

Parameters

port	COM port name as a string.

Returns
Nothing

Description
Clears all data from the input buffer of the specified COM port.

close

```
COM.close(port)
```

Parameters

port	COM port name as a string.

Returns
Nothing

Description
Closes the specified COM port.

isPortOpen

```
COM.isPortOpen(port)
```

Parameters

port	COM port name as a string.

Returns
True if the COM port is open. False otherwise.

Description
Returns true if the specified COM port is open. Returns false otherwise.

open

```
COM.open(port, baud)
```

Parameters

port	COM port name as a string.
baud	Baud rate of the port

Returns
Nothing

Description
Opens the specified COM port with a baud rate of baud. This function must be called before calling any other read or write functions to the COM port.

readAllText

```
COM.readAllText(port)
```

Parameters

port	COM port name as a string.

Returns
All data in the COM port as a string.

Description
Reads all data from the specified COM port and returns it as a string.

readByte

```
COM.readByte(port)
```

Parameters

port	COM port name as a string.

Returns
The first byte read from the COM port.

Description
Reads and returns one byte from the specified COM port.

readLine

```
COM.readLine(port)
```

Parameters

port	COM port name as a string.

Returns
One line of data from the COM port as a string.

Description
Reads all data from the specified COM port until an endline character is reached. Value is returned as a string.

write

```
COM.write(port, byte/byteArray)
```

Parameters

port	COM port name as a string.
byte/byteArray	Byte or byte array to write to the COM port.

Returns
Nothing

Description
Writes one byte given by byte to the specified COM port. If a byte array is provided instead, the byte array given by byteArray will be written to the specified COM port.

writeString

```
COM.writeString(port, str)
```

Parameters

port	COM port name as a string.
str	String to write to the COM port.

Returns
Nothing

Description
Writes the string given by str to the specified COM port.

writeStringNewLine

```
COM.writeStringNewLine(port, str)
```

Parameters

port	COM port name as a string.
str	String to write to the COM port.

Description
Writes the string given by str with a newline character appended to the end to the specified COM port.

read

```
COM.read(port, bytesToRead)
```

Parameters

port	COM port name as a string.

bytesToRead	Number of bytes to read from the COM port.

Returns
A byte array of length bytesToRead containing the data read from the COM port.

Description
Reads bytesToRead bytes from the COM port and returns it in a byte array.

Digital

get

```
Digital.get(port, [ezbIndex])
```

Parameters

port	Digital port to read from.
ezbIndex (optional)	Board index of the EZB to read from.

Returns
True if the specified digital port is 1 (high). False if the digital port is 0 (low).

Description
Reads and returns the digital value from the specified digital port.

set

```
Digital.set(port, value, [ezbIndex])
```

Parameters

port	Digital port to set the value of.
value	Value to set the digital port to as a Boolean or integer.
ezbIndex (optional)	Board index of the EZB to use.

Returns
Nothing

Description
Sets the value of the specified digital port to value.

setRandom

```
Digital.setRandom(port, [ezbIndex])
```

Parameters

port	Digital port to set the value of.
ezbIndex (optional)	Board index of the EZB to use.

Returns
The random value that the digital port was set to.

Description
Randomly sets the value of the specified digital port to either 0 or 1.

toggle

```
Digital.toggle(port, [ezbIndex])
```

Parameters

port	Digital port to toggle.
ezbIndex (optional)	Board index of the EZB to use.

Returns
The value toggled to.

Description
Toggles the digital port. If the digital port was 0 then it is set to 1. If the digital port was 1 then it is set to 0.

wait

```
Digital.wait(port, valueToWaitFor, [frequencyMs], [ezbIndex], [timeoutMS])
```

Parameters

port	Digital port to set the value of.
valueToWaitFor	Digital value to wait for as boolean or integer.
frequencyMs (optional)	How often the port is checked in milliseconds.
ezbIndex (optional)	Board index of the EZB to read from.
timeoutMS (optional)	Number of milliseconds to wait before timeout.

Returns
Nothing

Description
Suspends execution of the script until the value read from the specified digital port is equal to valueToWaitFor. If frequencyMs is provided the port is checked every frequencyMs milliseconds. Otherwise the port is checked as often as possible.

EZB

getCPUTemp

```
EZB.getCPUTemp([ezbIndex])
```

Parameters

ezbIndex (optional)	Board index of the EZB to read from.

Returns
CPU temperature of the EZB in degrees celsius.

Description
Returns the CPU temperature of the EZB in degrees Celsius.

getVoltage

```
EZB.getCPUTemp([ezbIndex])
```

Parameters

ezbIndex (optional)	Board index of the EZB to read from.

Returns
Input voltage of the EZB in volts.

Description
Returns the input voltage of the EZB in volts.

isConnected

```
EZB.isConnected(ezbIndex)
```

Parameters

ezbIndex	Board index of the EZB to check if connected.

Returns
True if the EZB at board index ezbIndex is connected. False otherwise.

Description
Checks if the EZB at board index ezbIndex is connected or not.

Example

```
if (EZB.isConnected(0)) {
  print("EZB is connected.");
} else {
  print("EZB is not connected.");
}
```

File

append

```
File.append(filename, byte/byteArray)
```

Parameters

filename	Path of file to append to.
byte/byteArray	Byte or byte array to append to the file.

Returns
Nothing

Description
Appends the byte given by byte to the file. If a byte array is provided instead, the byte array given by byteArray will be appended to the file.

appendString

```
File.appendString(filename, str)
```

Parameters

filename	Path of file to append to.
str	String to append to the file.

Returns
Nothing

Description
Appends the string given by str to the file.

appendStringLine

```
File.appendStringLine(filename, str)
```

Parameters

filename	Path of file to append to.
str	String to append to the file.

Returns
Nothing

Description
Appends the string given by str with a newline character appended to the end to the file.

delete

```
File.delete(filename)
```

Parameters

filename	Path of file to delete.

Returns
Nothing

Description
Deletes the file at the path specified by filename.

exists

```
File.exists(filename)
```

Parameters

filename	Path of file to check if exists.

Returns
True if the file at path filename exists. False otherwise.

Description
Checks if the file at path filename exists.

getLength

```
File.getLength(filename)
```

Parameters

filename	Path of file to get length of.

Returns
Number of characters in the file.

Description
Returns the number of characters contained in the file.

getReadPosition

```
File.getReadPosition(filename)
```

Parameters

filename	Path of file to get read position of.

Returns
Read position of the file as the number of characters that have been read since it was opened.

Description
Returns the read position of the file as the number of characters that have been read since it was opened.

isFileOpenForReading

```
File.isFileOpenForReading(filename)
```

Parameters

filename	Path of file to check if open for reading.

Returns
True if file is open for reading. False otherwise.

Description
Checks if the file at filename is open for reading or not. If a read function has been called on the file this will return true. If the file has not been opened for reading, or the file is closed using the readClose funciton then this will return false. A file will remain open for reading even after the script finishes executing unless the file is closed using the readClose function.

isReadEnd

```
File.isReadEnd(filename)
```

Parameters

filename	Path of file to check if at end.

Returns
True if the read position of the file is as at the end of the file. False otherwise.

Description
Checks if the read position of the file at filename is at the end of the file.

readAllText

```
File.readAllText(filename)
```

Parameters

filename	Path of file to read.

Returns
All text contained within the file as a string.

Description
Reads and returns all the text contained with the file at filename as a string.

readByte

```
File.readByte(filename)
```

Parameters

filename	Path of file to read from.

Returns
The ASCII code representation for the character at the current read position of the file.

Description
Reads and returns the ASCII code representation (one byte of data) of the character located at the current read position of the file, and then advances the read position by one character.

readChar

```
File.readChar(filename)
```

Parameters

filename	Path of file to read from.

Returns
The character at the current read position of the file.

Description
Reads and returns the character located at the current read position of the file, and then advances the read position by one character.

readClose

```
File.readClose(filename)
```

Parameters

filename	Path of file to close.

Returns
Nothing

Description
Closes the file at path filename that was being read from.

readLine

```
File.readLine(filename)
```

Parameters

filename	Path of file to read from.

Returns
The string from the current read position to the next endline character.

Description
Reads and returns all characters as a string from the current read position to the next endline character. The returned string does not contain the endline character. The read position advances to be immediately after the endline character.

readRandomLine

```
File.readRandomLine(filename)
```

Parameters

filename	Path of file to read from.

Returns
A random line from the file as a string.

Description
Reads and returns a random line from the file, which starts from the end of one line and ends at the next endline character. This function does not affect the read position of the file.

readReset

```
File.readReset(filename)
```

Parameters

filename	Path of file to reset read position of.

Returns
Nothing

Description
Resets the read position of the file to 0.

setReadPosition

```
File.setReadPosition(filename, position)
```

Parameters

filename	Path of file to set read position of.

position	Read position to set to.

Returns
Nothing

Description
Sets the read position of the file to position.

I2C

read

```
I2C.read(address, bytesToExpect, [ezbIndex])
```

Parameters

address	Address of the device to read from as a hexadecimal number (e.g.0x5e).
bytesToExpect	Number of bytes to read from the device.
ezbIndex (optional)	Board index of the EZB to read from.

Returns
A byte array of length bytesToExpect containing the bytes read from the device.

Description
Reads bytesToExpect bytes from the device at address and returns them as a byte array.

readByte

```
I2C.readByte(address, [ezbIndex])
```

Parameters

address	Address of the device to read from as a hexadecimal number (e.g.0x5e).
ezbIndex (optional)	Board index of the EZB to read from.

Returns
One byte read from the device.

Description
Reads one byte of data from the device at address.

readString

```
I2C.readString(address, bytesToExpect, [ezbIndex])
```

Parameters

address	Address of the device to read from as a hexadecimal number (e.g.0x5e).
bytesToExpect	Number of bytes to read from the device (1 byte = 1 character).
ezbIndex (optional)	Board index of the EZB to read from.

Returns
A string of length bytesToExpect.

Description
Reads a string of data of length bytesToExpect from the device at address.

setClockSpeed

```
I2C.setClockSpeed(speed, [ezbIndex])
```

Parameters

speed	Clock speed to set the I2C interface to.
ezbIndex (optional)	Board index of the EZB to set the I2C clock speed of.

Returns
Nothing

Description
Sets the clock speed of the I2C interface. The default speed is 100000, which is 100khz. Many devices support faster speeds, up to 400000 (400khz).

write

```
I2C.write(address, byte/byteArray, [ezbIndex])
```

Parameters

address	Address of the device to read from as a hexadecimal number (e.g.0x5e).
byte/byteArray	Byte or byte array to write to the device.
ezbIndex (optional)	Board index of the EZB to write to.

Returns
Nothing

Description
Writes byte to the device at address. A byte array can instead be provided to write to the device at address.

writeString

```
I2C.write(address, str, [ezbIndex])
```

Parameters

address	Address of the device to read from as a hexadecimal number (e.g.0x5e).
str	String to write to the device.
ezbIndex (optional)	Board index of the EZB to write to.

Returns
Nothing

Description
Writes the string given by str to the device at address.

Movement

down

```
Movement.down([timeOut])
```

Parameters

timeOut (optional)	Duration to move down for in milliseconds.

Returns
Nothing

Description
Moves the robot down using the project's Movement Panel skill. The Movement Panel skill must have the capability to move the robot down. If timeOut is provided the robot will move down for timeOut milliseconds. Otherwise it will continue to move down until a different Movement function is called.

forward

```
Movement.forward([speed], [timeOut])
```

Parameters

speed (optional)	Speed to move at.
timeOut (optional)	Duration to move forward for in milliseconds.

Returns
Nothing

Description
Moves the robot forward using the project's Movement Panel skill. The speed can be specified by providing speed as a value between 0 and 255. If timeOut is provided the robot will move forward for timeOut milliseconds. Otherwise it will continue to move forward until a different Movement function is called.

land

```
Movement.land()
```

Returns
Nothing

Description
Lands the robot using the project's Movement Panel skill. The Movement Panel skill must have the capability to land the robot.

getSpeed

```
Movement.getSpeed()
```

Returns
The global movement speed value as a value between 0 and 255.

Description
Returns the global movement speed value as a value between 0 and 255. The global movement speed value is the speed used for the left and right wheels by the project's

Movement Panel skill if it has the capability to do so.

getSpeedLeft

```
Movement.getSpeedLeft()
```

Returns
The global movement speed value of the left wheel as a value between 0 and 255.

Description
Returns the global movement speed value of the left wheel as a value between 0 and 255. The global movement speed value of the left wheel is the speed used for the left wheel by the project's Movement Panel skill if it has the capability to do so.

getSpeedRight

```
Movement.getSpeedRight()
```

Returns
The global movement speed value of the right wheel as a value between 0 and 255.

Description
Returns the global movement speed value of the right wheel as a value between 0 and 255. The global movement speed value of the right wheel is the speed used for the right wheel by the project's Movement Panel skill if it has the capability to do so.

left

```
Movement.left([speed], [timeOut])
```

Parameters

speed (optional)	Speed to move at.
timeOut (optional)	Duration to move left for in milliseconds.

Returns
Nothing

Description
Moves the robot left using the project's Movement Panel skill. The speed can be specified by providing speed as a value between 0 and 255. If timeOut is provided the robot will move left for timeOut milliseconds. Otherwise it will continue to move left until a different Movement function is called.

reverse

```
Movement.reverse([speed], [timeOut])
```

Parameters

speed (optional)	Speed to move at.
timeOut (optional)	Duration to move reverse for in milliseconds.

Returns
Nothing

Description
Moves the robot reverse using the project's Movement Panel skill. The speed can be specified by providing speed as a value between 0 and 255. If timeOut is provided the robot will move reverse for timeOut milliseconds. Otherwise it will continue to move reverse until

a different Movement function is called.

right

```
Movement.right([speed], [timeOut])
```

Parameters

speed (optional)	Speed to move at.
timeOut (optional)	Duration to move right for in milliseconds.

Returns
Nothing

Description
Moves the robot right using the project's Movement Panel skill. The speed can be specified by providing speed as a value between 0 and 255. If timeOut is provided the robot will move right for timeOut milliseconds. Otherwise it will continue to move right until a different Movement function is called.

rollLeft

```
Movement.rollLeft([timeOut])
```

Parameters

timeOut (optional)	Duration to roll left for in milliseconds.

Returns
Nothing

Description
Rolls the robot left using the project's Movement Panel skill. The Movement Panel skill must have the capability to roll the robot left. If timeOut is provided the robot will roll left for timeOut milliseconds. Otherwise it will continue to roll left until a different Movement function is called.

rollRight

```
Movement.rollRight([timeOut])
```

Parameters

timeOut (optional)	Duration to roll right for in milliseconds.

Returns
Nothing

Description
Rolls the robot right using the project's Movement Panel skill. The Movement Panel skill must have the capability to roll the robot right. If timeOut is provided the robot will roll right for timeOut milliseconds. Otherwise it will continue to roll right until a different Movement function is called.

setSpeed

```
Movement.setSpeed(speed/speedLeft, [speedRight])
```

Parameters

speed/speedLeft	Global Movement Speed value for left and right wheel. If

	speedRight is provided this value will be used for the global Movement Speed value of the left wheel only.
speedRight (optional)	Global movement speed value for the right wheel.

Returns
Nothing

Description
Sets the global movement speed value for the left and right wheels. If speedLeft and speedRight are provided the values for the left and right wheels will be set respectively.

Example

```
// Set both left and right speeds to 127 (half speed)
Movement.setSpeed(127);
```

```
// Set the left speed to 100 and right speed to 200 (slightly turn left)
Movement.setSpeed(100, 200);
```

setSpeedLeft

```
Movement.setSpeedLeft(speed)
```

Parameters

speed	Global movement speed value for the left wheel.

Returns
Nothing

Description
Sets the global movement speed value for the left wheel. **speed** is a value between 0 and 255.

setSpeedRight

```
Movement.setSpeedRight(speed)
```

Parameters

speed	Global movement speed value for the right wheel.

Returns
Nothing

Description
Sets the global movement speed value for the right wheel. speed is a value between 0 and 255.

stop

```
Movement.stop()
```

Returns
Nothing

Description
Stops the robot using the project's Movement Panel skill.

takeoff

```
Movement.takeoff()
```

Returns
Nothing

Description
Tells the robot to takeoff using project's Movement Panel skill. The Movement Panel skill must have takeoff capability.

up

```
Movement.up([timeOut])
```

Parameters

timeOut (optional)	Duration to move up for in milliseconds.

Returns
Nothing

Description
Moves the robot up using the project's Movement Panel skill. The Movement Panel skill must have the capability to move the robot up. If timeOut is provided the robot will move up for timeOut milliseconds. Otherwise it will continue to move up until a different Movement function is called.

waitForDown

```
Movement.waitForDown()
```

Returns
Nothing

Description
Suspends execution of the script until down is executed from the project's movement panel either by the user or from another script.

waitForForward

```
Movement.waitForForward()
```

Returns
Nothing

Description
Suspends execution of the script until forward is executed from the project's movement panel either by the user or from another script.

waitForLeft

```
Movement.waitForLeft()
```

Returns
Nothing

Description
Suspends execution of the script until left is executed from the project's movement panel either by the user or from another script.

waitForReverse

```
Movement.waitForReverse()
```

Returns
Nothing

Description
Suspends execution of the script until reverse is executed from the project's movement panel either by the user or from another script.

waitForRight

```
Movement.waitForRight()
```

Returns
Nothing

Description
Suspends execution of the script until right is executed from the project's movement panel either by the user or from another script.

waitForStop

```
Movement.waitForStop()
```

Returns
Nothing

Description
Suspends execution of the script until stop is executed from the project's movement panel either by the user or from another script.

waitForUp

```
Movement.waitForUp()
```

Returns
Nothing

Description
Suspends execution of the script until up is executed from the project's movement panel either by the user or from another script.

fromString

This is a slug for fromString. Can use custom movement names

Net

hTTPGet

```
Net.hTTPGet(url, [timeout])
```

Parameters

url	URL to send HTTP Get request to.
timeout (optional)	Timeout in milliseconds.

Returns
HTTP contents from the provided URL as a string.

Description
Sends an HTTP Get request to url. If timeout is specified the request will timeout after timeout milliseconds. Suspends script execution until the request completes, or the request times out.

hTTPPost

```
Net.hTTPPost(url, postData, [timeout])
```

Parameters

url	URL to send HTTP POST request to.
postData	Data to send as part of POST request as a string.
timeout (optional)	Timeout in milliseconds.
headerNames (optional)	Array string of header names to append to the post request
headerValues (optional)	Array string of header values to append to the post request (must match count of headerNames)

Returns
The HTTP POST request response as a string.

Description
Sends an HTTP POST request to url with postData stored in the request body. If timeout is specified the request will timeout after timeout milliseconds. Suspends script execution until the request completes, or the request times out.

Example

```
var resp = Net.hTTPPost(
 "https://postman-echo.com/post",
 "Some text");

 print(resp);
```

```
// post to an echo server
var resp = Net.hTTPPost(
 "https://postman-echo.com/post",
 "banana",
 1000,
 ["customHeaderTitle"],
 ["customHeaderValue"]);

 print(resp);
```

isInternetAvailable

```
Net.isInternetAvailable()
```

Returns
True if the computer is connected to the internet. False otherwise.

Description
Checks the internet connection of the computer.

sendUDP

```
Net.sendUDP(hostname, port, byteArray)
```

Parameters

hostname	Hostname of receiver as a string.
port	Port to use.
byteArray	Data to send as a byte array.

Returns
The length of the byteArray being sent.

Description
Sends byteArray over the specified port to the hostname using UDP.

Example

```
// 1. This corresponds to the ASCII codes for "Hello"
var data = [72, 101, 108, 108, 111];

// 2. Call Net.sendUDP with the hostname, port, and your byteArray
Net.sendUDP("10.0.0.10", 8870, data);
```

```
// 1. Call Net.sendUDP with the hostname, port, and your byteArray(hello)
Net.sendUDP("10.0.0.10", 8870, [72, 101, 108, 108, 111]);
```

Ping

get

```
Ping.get(triggerPort, echoPort, [ezbIndex])
```

Parameters

triggerPort	Trigger port used by the ultrasonic distance sensor.
echoPort	Echo port used by the ultrasonic distance sensor.
ezbIndex (optional)	Board index of the EZB to use.

Returns
The value measured from the ultrasonic distance sensor as a number between 0 and 255.

Description
Pings the ultrasonic distance sensor once and returns the value measured.

waitForBetween

```
Ping.waitForBetween(triggerPort, echoPort, low, high, [ezbIndex])
```

Parameters

triggerPort	Trigger port used by the ultrasonic distance sensor.
echoPort	Echo port used by the ultrasonic distance sensor.
low	Inclusive lower bound on value to wait for.
high	Inclusive upper bound on value to wait for.
ezbIndex (optional)	Board index of the EZB to use.

Returns
The value measured from the ultrasonic distance sensor that is between low and high as a number between 0 and 255.

Description
Suspends execution of the script until the value read from the ultrasonic distance sensor is between low (inclusvie) and high (inclusive). The value read from the ultrasonic distance sensor is returned when the script resumes execution.

waitForEquals

```
Ping.waitForEquals(triggerPort, echoPort, distance, [ezbIndex])
```

Parameters

triggerPort	Trigger port used by the ultrasonic distance sensor.
echoPort	Echo port used by the ultrasonic distance sensor.
distance	Value to wait for.
ezbIndex (optional)	Board index of the EZB to use.

Returns
The value measured from the ultrasonic distance sensor that is equal to distance as a number between 0 and 255.

Description
Suspends execution of the script until the value read from the ultrasonic distance sensor is equal to distance. The value read from the ultrasonic distance sensor is returned when the script resumes execution.

waitForHigher

```
Ping.waitForHigher(triggerPort, echoPort, distance, [ezbIndex])
```

Parameters

triggerPort	Trigger port used by the ultrasonic distance sensor.
echoPort	Echo port used by the ultrasonic distance sensor.
distance	Exclusive lower bound on value to wait for
ezbIndex (optional)	Board index of the EZB to use.

Returns
The value measured from the ultrasonic distance sensor that is higher than distance as a number between 0 and 255.

Description
Suspends execution of the script until the value read from the ultrasonic distance sensor is higher than distance. The value read from the ultrasonic distance sensor is returned when the script resumes execution.

waitForLower

```
Ping.waitForLower(triggerPort, echoPort, distance, [ezbIndex])
```

Parameters

triggerPort	Trigger port used by the ultrasonic distance sensor.
echoPort	Echo port used by the ultrasonic distance sensor.
distance	Exclusive upper bound on value to wait for.
ezbIndex (optional)	Board index of the EZB to use.

Returns
The value measured from the ultrasonic distance sensor that is lower than distance as a number between 0 and 255.

Description
Suspends execution of the script until the value read from the ultrasonic distance sensor is lower than distance. The value read from the ultrasonic distance sensor is returned when the script resumes execution.

PWM

get

```
PWM.get(port, [ezbIndex])
```

Parameters

port	Port to get PWM duty cycle from.
ezbIndex (optional)	Board index of the EZB to get PWM duty cycle from.

Returns
The PWM duty cycle percentage of the specified port as a value between 0 and 100.

Description
Returns the PWM duty cycle percentage of the specified port. Duty cycle percentage is returned as a value between 0 and 100.

set

```
PWM.set(port, dutyCycle, [ezbIndex])
```

Parameters

port	Port to use.
dutyCycle	Duty cycle percentage to set PWM duty cycle to.
ezbIndex (optional)	Board index of the EZB to get PWM duty cycle from.

Returns
Nothing

Description
Sets the PWM duty cycle percentage of the specified port to dutyCycle which is a value between 0 and 100.

setRandom

```
PWM.setRandom(port, lowCycle, highCycle, [ezbIndex])
```

Parameters

port	Port to use.
lowCycle	Inclusive lower bound on duty cycle percentage to set PWM duty cycle to.
highCycle	Exclusive upper bound on duty cycle percentage to set PWM duty cycle to.
ezbIndex (optional)	Board index of the EZB to get PWM duty cycle from.

Returns
The random value that the PWM was set to.

Description
Sets the PWM duty cycle percentage of the specified port to a value randomly chosen between lowCycle (inclusive) and highCycle (exclusive). lowCycle and highCycle are values between 0 and 100.

Servo

decrement

```
Servo.decrement(port, count, [ezbIndex])
```

Parameters

port	Servo port to use.
count	Degrees to decrement by.
ezbIndex (optional)	Board index of the EZB to use.

Returns
Nothing

Description
Decrements the position of the servo at the specified port by count. Servo position is between 0 and 180.

getPosition

```
Servo.getPosition(port, [ezbIndex])
```

Parameters

port	Servo port to use.
ezbIndex (optional)	Board index of the EZB to use.

Returns
The commanded position of the servo at the specified port as a value between 1 and 180.

Description
Returns the position of the servo at the specified port as a value between 1 and 180. This function only returns the position the servo was commanded to move to, not the actual position of the servo. This does not query a smart servo to get the position. It only returns the last position the servo was moved to.
If you wish to get the servo position of a bi-directional servo, use the getPositionRealtime() command

getPositionRealtime

```
Servo.getPositionRealtime(port, [ezbIndex])
```

Parameters

port	Servo port to use.
ezbIndex (optional)	Board index of the EZB to use.

Returns
The actual position of the servo at the specified port as a value between 1 and 180.

Description
Returns the position of the servo at the specified port. This function returns the physical position of the servo returned by the servo. This function was meant to work with servos which accept bi-directional communication (e.g. Dynamixel, LewanSoul).

- If this function is called on a pwm servo or a servo that does not support bi-direction

communication, the last set servo position will be returned
• If the servo being queried has a communication issue, the last set position will be returned

getSpeed

```
Servo.getSpeed(port, [ezbIndex])
```

Parameters

port	Servo port to get speed from.
ezbIndex (optional)	Board index of the EZB to use.

Returns
The speed that the specified port is set to as a value

Description
Returns the speed that the specified port is set to as a value

increment

```
Servo.increment(port, count, [ezbIndex])
```

Parameters

port	Servo port to use.
count	Degrees to increment by.
ezbIndex (optional)	Board index of the EZB to use.

Returns
Nothing

Description
Increments the position of the servo at the specified port by count. Servo position is between 0 and 180.

release

```
Servo.release(port, [ezbIndex])
```

Parameters

port	Servo port to release.
ezbIndex (optional)	Board index of the EZB to use.

Returns
Nothing

Description
Stops the servo at the specified port from holding its position.

releaseAll

```
Servo.releaseAll([ezbIndex])
```

Parameters

ezbIndex (optional)	Board index of the EZB to use.

Returns
Nothing

Description
Stops all servos at all ports from holding their position.
*Warning: this will affect all ports and reset their state. That means if you using some ports for digital i/o or UART, those ports will be reset. This resets ALL ports, not some. If you wish to only release servos on specific ports, create a script using Servo.release(port) instead of this.
i.e.

```
Servo.release(d0);
Servo.release(d1);
Servo.release(d3);
```

setMaxPositionLimit

```
Servo.setMaxPositionLimit(port, position, [ezbIndex])
```

Parameters

port	Servo port to set max position limit of.
position	Max position limit.
ezbIndex (optional)	Board index of the EZB to use.

Returns
Nothing

Description
Sets the global maximum position that the servo at the specified port can move to
the position. The position is a value between 0 and 180 (or the global max servo position if overridden)
While robot skills may have their own Min/Max servo options, this sets the global value. This is used for safety to ensure servos don't move outside the specified range.

setMinPositionLimit

```
Servo.setMinPositionLimit(port, position, [ezbIndex])
```

Parameters

port	Servo port to set min position limit of.
position	Min position limit.
ezbIndex (optional)	Board index of the EZB to use.

Returns
Nothing

Description
Sets the global minimum position that the servo at the specified port can move to the position. The position is a value between 0 and 180 (or the global max servo position if overridden).
While robot skills may have their own Min/Max servo options, this sets the global value. This is used for safety to ensure servos don't move outside the specified range.

setPosition

```
Servo.setPosition(port, position, [ezbIndex])
```

Parameters

port	Servo port to set position of.
position	Position to move the servo to.
ezbIndex (optional)	Board index of the EZB to use.

Returns
Nothing

Description
Commands the servo to move to position. position is a value between 1 and 180.

setPositionRandom

```
Servo.setPositionRandom(port, lowPosition, highPosition, [ezbIndex])
```

Parameters

port	Servo port to set position of.
lowPosition	Inclusive lower bound on random position value.
highPosition	Exclusive upper bound on random position value.
ezbIndex (optional)	Board index of the EZB to use.

Returns
The random position that the servo was commanded to move to.

Description
Commands the servo to move to a random position between lowPosition (inclusive) and highPosition (exclusive). lowPosition and highPositionis are values between 1 and 180.

setSpeed

```
Servo.setSpeed(port, speed, [ezbIndex])
```

Parameters

port	Servo port to use.
speed	Speed to set the servo port to.
ezbIndex (optional)	Board index of the EZB to use.

Returns
Nothing

Description
Sets the speed of the servo port to speed. speed is a value between 0 (fastest) and 10 (slowest).
This tutorial explains how to re-initialize the servo speeds during startup: Setting Servo speeds and Initialization Script Tutorial - Tutorials - Community - Synthiam

waitForMove

```
Servo.waitForMove(port, [ezbIndex], [timeoutMS])
```

Parameters

port	Servo port to wait for movement.
ezbIndex (optional)	Board index of the EZB to use.
timeoutMS (optional)	Number of milliseconds to wait before timeout.

Returns
The position of the servo at the specified port that has changed. Returns -1 if timeout.

Description
Suspends execution of the script until the commanded position of the servo at the specified port changes. The commanded position can be changed either by the user from another skill, or another script.

waitForPositionEquals

```
Servo.waitForPositionEquals(port, position, [ezbIndex], [timeoutMS])
```

Parameters

port	Servo port to use.
position	Position to wait for.
ezbIndex (optional)	Board index of the EZB to use.
timeoutMS (optional)	Number of milliseconds to wait before timeout.

Returns
Current servo position. Returns -1 if timeout

Description
Suspends execution of the script until the commanded position of the servo at the specified port is equal to position. The commanded position can be changed either by the user from another skill, or another script.

waitForPositionHigher

```
Servo.waitForPositionHigher(port, position, [ezbIndex], [timeoutMS])
```

Parameters

port	Servo port to use.
position	Exclusive lower bound on the position to wait for.
ezbIndex (optional)	Board index of the EZB to use.
timeoutMS (optional)	Number of milliseconds to wait before timeout.

Returns
The position of the servo at the specified port that is higher than position. Returns -1 if timeout.

Description
Suspends execution of the script until the commanded position of the servo at the specified port is higher than position. The commanded position can be changed either by the user from another skill, or another script.

waitForPositionLower

```
Servo.waitForPositionLower(port, position, [ezbIndex], [timeoutMS])
```

Parameters

port	Servo port to use.
position	Exclusive upper bound on the position to wait for.
ezbIndex (optional)	Board index of the EZB to use.
timeoutMS (optional)	Number of milliseconds to wait before timeout.

Returns
The position of the servo at the specified port that is lower than position. Returns -1 if timeout

Description
Suspends execution of the script until the commanded position of the servo at the specified port is lower than position. The commanded position can be changed either by the user from another skill, or another script.

setVelocity

```
setVelocity(port, velocity, ezb index);
```

Parameters

port	The servo port
velocity	The velocity value
ezb index (optional)	The ezb index

Description
Set the velocity of a servo. The servo and servo driver must support this feature. For example, check the Dynamixel robot skill because some of their servos support the Velocity feature.

Example

```
// Set the velocity of the servo v1 to 10. By default, this will use EZB #0 because no ezb is specified
Servo.setVelocity(v1, 10);
```

```
// Specify the velocity of servo V1 on ezb index #3
Servo.setVelocity(v1, 10, 3);
```

setAcceleration

```
setAcceleration(port, value, ezb index);
```

Parameters

port	The servo port
value	The Acceleration value
ezb index (optional)	The ezb index

Description
Set the acceleration of a servo. The servo and servo driver must support this feature. For example, check the Dynamixel robot skill because some of their servos support the Acceleration feature.

Example

```
// Set the Acceleration of the servo v1 to 10. By default, this will use EZB #0 because no
ezb is specified
Servo.setAcceleration(v1, 10);
```

```
// Specify the Acceleration of servo V1 on ezb index #3
Servo.setAcceleration(v1, 10, 3);
```

IsReleased

```
IsReleased(port, ezbIndex)
```

Parameters

port	The servo port
ezbIndex (optional)	The ezb index

Returns
boolean (true of false)

Description
Returns the released state of the servo (torque off or on)

Example

```
if (IsReleased(d0))
  print("Servo d0 is released");
```

setFineTuneOffset

```
setFineTuneOffset(port, offset, ezbIndex)
```

Parameters

port - Servo port (i.e. d0	v2
offset - Value of the offset. Can be a positive or negative value	
ezbIndex - The ezb to apply this offset value (optional)	

Description
Sets the global fine-tune offset in positions of the specified servo. For example, if you set the offset to +10, every servo position set by other robot skills or script commands will add 10 positions. If you query the servo position, it'll return 10, not 20.
One of the uses for this could be used to maintain servos upright from an IMU, such as in this post.

Example

```
// Set the servo D0 to an offset of +20 positions
setFineTuneOffset(D0, 20);
```

```
// Set the servo D0 to an offset of -15 positions
setFineTuneOffset(D0, -15);
```

getFineTuneOffset

```
getFineTuneOffset(port, ezbIndex)
```

Parameters

port - Servo port (i.e. d0	v2
ezbIndex - The ezb to apply this offset value (optional)	

Description
Gets the global fine-tuned offset in positions of the specified servo. The value is set by either loading a fine tune servo profile or calling setServoFileTune()
One of the uses for this could be used to maintain servos upright from an IMU, such as in this post.

Example

```
// Get the servo D0 offset and print it to the output
print("The offset of D0 is " + getFineTuneOffset(D0));
```

getVelocity

```
Servo.getVelocity(port, [ezbIndex])
```

Parameters

port	Servo port to get velocity from.
ezbIndex (optional)	Board index of the EZB to use.

Returns
The velocity that the specified port is set to as a value

Description
Returns the velocity that the specified port is set to as a value

getAcceleration

```
Servo.getAcceleration(port, [ezbIndex])
```

Parameters

port	Servo port to get acceleration from.
ezbIndex (optional)	Board index of the EZB to use.

Returns
The acceleration that the specified port is set to as a value

Description
Returns the acceleration that the specified port is set to as a value

UART

hardwareUartAvailable

```
UART.hardwareUartAvailable(uartIndex, [ezbIndex])
```

Parameters

uartIndex	Index of UART to check.
ezbIndex (optional)	Board index of the EZB to use.

Returns
Number of bytes available in the UART receive buffer at the specified UART.

Description
Returns the number of bytes available in the UART receive buffer at the specified UART given by uartIndex. The UART must be initialized first.

hardwareUartRead

```
UART.hardwareUartRead(uartIndex, bytesToRead, [ezbIndex])
```

Parameters

uartIndex	Index of UART to read from.
bytesToRead	Number of bytes to read from the UART.
ezbIndex (optional)	Board index of the EZB to use.

Returns
A byte array of length bytesToRead containing data received from the specified UART.

Description
Returns a byte array of length bytesToRead containing data received from the specified UART. The UART must be initialized first.

hardwareUartReadAvailable

```
UART.hardwareUartReadAvailable(uartIndex, [ezbIndex])
```

Parameters

uartIndex	Index of UART to read from.
ezbIndex (optional)	Board index of the EZB to use.

Returns
A byte array containing all available data received from the specified UART.

Description
Returns a byte array containing all available data received from the specified UART. The UART must be initialized first.

hardwareUartReadString

```
UART.hardwareUartReadString(uartIndex, bytesToRead, [ezbIndex])
```

Parameters

uartIndex	Index of UART to read from.
bytesToRead	Number of bytes to read from the UART.
ezbIndex (optional)	Board index of the EZB to use.

Returns
A string of length bytesToRead containing data received from the specified UART.

Description
Returns a string of length bytesToRead containing data received from the specified UART. The UART must be initialized first.

hardwareUartReadStringAvailable

```
UART.hardwareUartReadStringAvailable(uartIndex, [ezbIndex])
```

Parameters

uartIndex	Index of UART to read from.
ezbIndex (optional)	Board index of the EZB to use.

Returns
A string containing all available data received from the specified UART.

Description
Returns a string containing all available data received from the specified UART. The UART must be initialized first.

hardwareUartWrite

```
UART.hardwareUartWrite(uartIndex, byte/byteArray, [ezbIndex])
```

Parameters

uartIndex	Index of UART to write to.
byte/byteArray	Byte or byte array to write to the UART.
ezbIndex (optional)	Board index of the EZB to use.

Returns
Nothing

Description
Writes byte to the specified UART. A byte array can instead be provided to write to the specified UART. The UART must be initialized first.

Example

```
// Initialize the first uart (#0) to 9600 baud
UART.initHardwareUart(0, 9600);

// Write the array of bytes to the first UART
UART.hardwareUartWrite(0, [0xff, 0xf0, 0x30]);
```

```
// Initialize the first uart (#0) to 9600 baud
UART.initHardwareUart(0, 9600);

// Write a single byte to the first UART
UART.hardwareUartWrite(0, 0xff);
```

```
var str = [0xff, 0xfe, 0xfa, 0x35, 0xaf];
```

```
// write the amount of data we're gonna send
UART.hardwareUartWrite(0, str.length);

// now send the data
UART.hardwareUartWrite(0, str);
```

hardwareUartWriteString

```
UART.hardwareUartWriteString(uartIndex, str, [ezbIndex])
```

Parameters

uartIndex	Index of UART to write to.
str	String to write to the UART.
ezbIndex (optional)	Board index of the EZB to use.

Returns
Nothing

Description
Writes string given by str to the specified UART. The UART must be initialized first.

Example

```
var str = "some random amount of data";

// send the data to UART #0 on the first EZB
UART.hardwareUartWriteString(0, str);
```

initHardwareUart

```
UART.initHardwareUart(uartIndex, baud, [ezbIndex])
```

Parameters

uartIndex	Index of UART to initialize.
baud	Baud rate to set the UART to.
ezbIndex (optional)	Board index of the EZB to use.

Returns
Nothing

Description
Initializes the UART with the specified baud rate. The UART will stay initialized until the EZB is power cycled. The UART must be initialized using this command before you can READ or WRITE to the interface.
Once you have initialized the UART interface with this command, you may now use the other UART commands to read and write to the interface.

sendSerial

```
UART.sendSerial(port, baud, byte/byteArray, [ezbIndex])
```

Parameters

port	Digital port to send data over.
baud	Baud rate to send data at.

byte/byteArray	Byte or byte array to send.
ezbIndex (optional)	Board index of the EZB to use.

Returns
Nothing

Description
Sends a byte or a byte array of data over the specified digital port with a baud rate of baud.

sendSerialString

```
UART.sendSerialString(port, baud, str, [ezbIndex])
```

Parameters

port	Digital port to send data over.
baud	Baud rate to send data at.
str	String to send.
ezbIndex (optional)	Board index of the EZB to use.

Returns
Nothing

Description
Sends a string of data over the specified digital port with a baud rate of baud.

Utility

browser

```
Utility.browser(url)
```

Parameters

url	URL to open.

Returns
Nothing

Description
Opens the provided URL in the computer's default browser.

checkForUpdate

```
Utility.checkForUpdate()
```

Returns
True if an ARC update is available. False otherwise.

Description
Checks if an ARC update is available.

clearGlobalVariable

```
Utility.clearGlobalVariable(globalVariableName)
```

Parameters

globalVariableName	Name of the global variable to clear as a string.

Returns
Nothing

Description
Deletes the specified global variable.

closeControl

```
Utility.closeControl()
```

Returns
Nothing

Description
Used for mobile devices and the Interface Builder only, this function will close the current control, the same as pressing the BACK button on your device.

defineGlobalVariable

```
Utility.defineGlobalVariable(globalVariableName, size, [defaultValue])
```

Parameters

globalVariableName	Name of the global variable to define as a string. Must start with '$'

size	Size of the array to create.
defaultValue (optional)	Default value to fill the array.

Returns
Nothing

Description
Creates a global variable array with name globalVariableName and size size. If defaultValue is provided the array will be filled with defaultValue. Otherwise it will be filled with empty strings.

exec

```
Utility.exec(filename, [arguments])
```

Parameters

filename	File path of the executable file to execute.
arguments (optional)	Optional arguments to pass to the executable file as a string.

Returns
Nothing

Description
Execute the executable file at file path filename with optional arguments arguments.

fillGlobalArray

```
Utility.fillGlobalArray(globalVariableName, defaultValue)
```

Parameters

globalVariableName	Name of the global array as a string.
defaultValue	Value to fill the array with.

Returns
Nothing

Description
Fills the global array with name globalVariableName with the value defaultValue.

getBit

```
Utility.getBit(value, bitPosition)
```

Parameters

value	Number to get bit from.
bitPosition	Position to get bit from.

Returns
The bit (1 or 0) at position bitPosition in the number value.

Description
Returns the bit at position bitPosition in the number value. Position is 0 indexed starting from the least significant bit.

getGlobalArraySize

```
Utility.getGlobalArraySize(globalVariableName)
```

Parameters

globalVariableName	Name of the global array as a string.

Returns
Returns The size of the global array.

Description
Returns the size of the global array with the name globalVariableName.

getRandom

```
Utility.getRandom(min, max)
```

Parameters

min	Inclusive lower bound on the random integer.
max	Exclusive upper bound on the random integer.

Returns
Random integer between min (inclusive) and max (exclusive).

Description
Returns a random integer between min (inclusive) and max (exclusive).

getRandomUnique

```
Utility.getRandomUnique(min, max)
```

Parameters

min	Inclusive lower bound on the random integer.
max	Exclusive upper bound on the random integer.

Returns
Random integer between min (inclusive) and max (exclusive) and unique from the last returned integer.

Description
Returns a random integer between min (inclusive) and max (exclusive). This function will never return the same integer twice.

loadProject

```
Utility.loadProject(filename)
```

Parameters

filename	File path of project to load.

Returns
Nothing

Description
Loads the project at file path filename.

map

```
Utility.map(input, inputMin, inputMax, outputMin, outputMax)
```

Parameters

input	Value to map.
inputMin	Minimum value the input can be.
inputMax	Maximum value the input can be.
outputMin	Minimum value the output can be.
outputMax	Maximum value the output can be.

Returns
input mapped onto the provided range of outputs.

Description
Maps input onto the provided range of outputs. (e.g. if input is halfway between inputMin and inputMax, then this function will return the value halfway between ouputMin and outputMax).

The formula for calculating the return value is shown below.

setBits

```
Utility.setBits(b7, b6, b5, b4, b3, b2, b1, b0)
```

Parameters

b7	The bit in the 8th position (most significant bit).
b6	The bit in the 7th position.
b5	The bit in the 6th position.
b4	The bit in the 5th position.
b3	The bit in the 4th position.
b2	The bit in the 3rd position.
b1	The bit in the 2nd position.
b0	The bit in the 1st position (least significant bit).

Returns
The number in decimal that is equal to the binary number (b7 b6 b5 b4 b3 b2 b1 b0).

Description
Converts the bits given by b7, b6, b5, b4, b3, b2, b1, b0 into decimal and returns the number.

showControl

```
Utility.showControl(controlName)
```

Parameters

controlName	Name of the control to show.

Returns
Nothing

Description
Used for mobile devices and the Interface Builder only, this function will open the control with name controlName into the foreground.

showDesktop

```
Utility.showDesktop(desktopNumber)
```

Parameters

desktopNumber	Virtual desktop to show.

Returns
Nothing

Description
Switches to virtual desktop desktopNumber. desktopNumber can be either 1, 2 or 3.

sleepPCHibernate

```
Utility.sleepPCHibernate(force, wake)
```

Parameters

force	Boolean to force the computer to hibernate or not. If true other applications have no say in the decision.
wake	Boolean to allow the computer will wake up on Wake events or not.

Returns
Nothing

Description
Puts the computer in hibernation mode. If force is true, other applications will have no say in the decision to hibernate. If wake is true, the computer will be allowed to wake up on Wake events.

sleepPCSuspend

```
Utility.sleepPCSuspend(force, wake)
```

Parameters

force	Boolean to force the computer to sleep or not. If true other applications have no say in the decision.
wake	Boolean to allow the computer will wake up on Wake events or not.

Returns
Nothing

Description
Puts the computer in sleep mode. If force is true, other applications will have no say in the decision to sleep. If wake is true, the computer will be allowed to wake up on Wake events.

waitUntilTime

```
Utility.waitUntilTime(hour, minute)
```

Parameters

hour	Hour to wait until in 24 hour format.
minute	Minute to wait until.

Returns
Nothing

Description
Suspends execution of the script until the time (from the computer's clock) reaches the specified time.

shutdownPC

```
Utility.shutdownPC();
```

Description
Shuts down the PC. Does not prompt to save the project. Forces windows to shut down all applications.

Navigation

updateLocation

```
Navigation.updateLocation(x, y, degreesHeading, [confidence]);
```

Parameters

x	Cartesian X (cm) coordinate of the robot
y	Cartesian Y (cm) coordinate of the robot
degreesHeading	Direction in degrees (0-359) that the robot is facing relative to the starting position
confidence	[optional] the confidence (0-255) of accuracy of the coordinate location

Description

Send the cartesian coordinate of the robot to the NMS (navigation messaging system).

Example

```
// Send to the NMS that the robot is moving forward 30 cm over 30 seconds

for (var x = 0; x < 30; x++) {

  print("Position X:" + x);
  Navigation.updateLocation(x, y, degreesHeading);
  sleep(1000);
}
```

updateScan

```
updateScan(distance, confidence, degree)
```

Parameters

distance	the distance in CM of the detected obstacle
confidence	the confidence (0-255) of the detected obstacle
degree	the degree (0-359) of the obstacle

Description

Programmatically send the detected obstacle distance based on the current cartesian coordinate of the robot to the NMS (navigation messaging system).

SetNavigationStatusToStopCancel

```
SetNavigationStatusToStopCancel()
```

Description

Instruct the navigator in NMS Level #1 to stop/cancel navigating. The variable $NavigationStatus will also update to StoppedCancelled.
*Note: the NMS Level #1 robot skill must support this function. Verify with its manual that this feature is supported. There should also be an option in the respective robot skill for support pausing with close distances.
Example NMS Level #1 controls are:

- EZ-Slam

- The Navigator
- The Better Navigator
- Other navigation robot skills

Example

```
// Stop navigating if the ping sensor detects an object less than 100 distance
if (Ping.get(d0, d0) < 100)
  Navigation.SetNavigationStatusToStopCancel();
```

SetNavigationStatusToPause

```
SetNavigationStatusToPause()
```

Description

Instruct the navigator in NMS Level #1 to pause navigating. The variable $NavigationStatus will also update to Paused.
*Note: the NMS Level #1 robot skill must support this function. Verify with its manual that this feature is supported. There should also be an option in the respective robot skill for support pausing with close distances.
Example NMS Level #1 controls are:

- EZ-Slam
- The Navigator
- The Better Navigator
- Other navigation robot skills

Example

```
// Pause navigating if the ping sensor detects an object less than 100 distance and continue
if greater
if (Ping.get(d0, d0) < 100)
  Navigation.SetNavigationStatusToPause();
else
  Navigation.SetNavigationStatusToNavigating();
```

SetNavigationStatusToNavigating

```
SetNavigationStatusToNavigating()
```

Description

Instruct the navigator in NMS Level #1 to continue navigating if previously paused. The variable $NavigationStatus will also update to Navigating.
*Note: the NMS Level #1 robot skill must support this function. Verify with its manual that this feature is supported. There should also be an option in the respective robot skill for support pausing with close distances.
Example NMS Level #1 controls are:

- EZ-Slam
- The Navigator
- The Better Navigator
- Other navigation robot skills

Example

```
// Pause navigating if the ping sensor detects an object less than 100 distance and continue
if greater
if (Ping.get(d0, d0) < 100)
  Navigation.SetNavigationStatusToPause();
else
  Navigation.SetNavigationStatusToNavigating();
```

UI

waitForUserInput

```
UI.waitForUserInput(timeout, prompt)
```

Parameters

timeout	Time to wait in seconds before stopping waiting for their text input.
prompt	The text prompt to display to the user while waiting for their text input

Returns

The lowercase text entered by the user, or the word timeout.

Description

Suspends execution of the script and prompts the user to enter text or a timeout occurs after timeout seconds. If a timeout occurs, the method will return "timeout."
***Note: the returned string is in lowercase**

Example

```
// Wait for any input from the user
response = UI.waitForUserInput(30, "Type anything to me");

print("You entered: " + response);
```