Programming

Overview

Event-based Programming in ARC

ARC uses an event-based programming model, where robot behaviors are implemented as scripts that run in response to events. Triggers can include the camera beginning to track an object, a joystick button press, a Robot Skill function call, or a ControlCommand() message. Each Robot Skill contains event scripts that implement its intended behavior. ARC also includes several programming options, allowing users to choose the approach that best fits their experience level and project goals.

Synthiam designed ARC to scale from beginner to advanced users. The goal is to make robot programming more accessible while increasing the likelihood of successful development and deployment.

Programming Modes

Choose a programming method based on your experience level and the complexity of your project.

RoboScratch

Very easy

RoboScratch visual programming interface for beginners.

RoboScratch is a block-based interface built into ARC for learning programming fundamentals. Users add function blocks to a workspace to create a sketch. Blocks include triggers such as when the camera sees an object or speech is detected, along with action blocks such as movement commands. By linking blocks together, users can teach robots to perform simple tasks without typing code.

- RoboScratch

Blockly

Easy

Blockly uses graphical blocks to represent functions, logic, and subroutines.

Blockly introduces programming concepts through draggable blocks that represent functions, logic, and control flow. It gives users access to advanced capabilities and allows them to create more complex behaviors visually, without needing to learn syntax or type traditional code.

- Blockly

Scripting

Intermediate

ARC scripting options include JavaScript, EZ-Script, and Python.

ARC supports text-based scripting in JavaScript, EZ-Script, and Python. Scripts written in different languages can interact within the same project. Using the ControlCommand messaging system, Robot Skills can send events and triggers to one another, making it possible to create modular and coordinated robot behaviors.

- EZ-Script
- Python
- JavaScript

Programming

Advanced

Compiled Robot Skills and libraries can interface directly with the ARC API.

For advanced development, users can compile programs and libraries that interface with the ARC API. Robot Skills can be created in managed CLR languages such as C#, or in native languages such as C and C++. These skills can be packaged for distribution, published to the Robot Skill Store, or integrated into custom solutions.

- Create a Robot Skill

Block Programming

Block-based programming separates executable actions into modular blocks that can be dragged, connected, and arranged to define behavior. Blocks often include icons and labels for readability, making this approach ideal for beginners, educators, and students.

- RoboScratch
- Blockly

Text-Based Programming

Text-based programming languages provide more detailed control and are appropriate when projects require greater complexity, custom algorithms, or integration with external libraries and services.

- EZ-Script
- Python
- JavaScript

Robot Skill Development

Custom Robot Skills allow users and companies to add services, features, and hardware support to ARC. Robot Skills are typically developed using CLR-supported languages such as C#, or native languages when appropriate. To learn how to build and package a Robot Skill, see the Create Robot Skill documentation.

Choosing a Programming Language

When editing a Robot Skill configuration in ARC, the code dialog includes tabs for selecting the language used by that code block. Projects can use multiple languages at the same time, so you are not limited to one language across the entire project. Choose the language that best fits the task and the developer's skill set.

The ARC code dialog allows a language to be selected for each code block or script.

Programming Guide

This guide provides a comprehensive and beginner-friendly explanation of some of the most important programming concepts used throughout Synthiam ARC scripting, especially when working with JavaScript or Python inside ARC. Understanding how classes, methods, and case sensitivity work will help you write more reliable scripts, avoid common errors, and better interpret the examples in the ARC documentation.

What is a Class?

In both JavaScript and Python, a `class` is a reusable template or blueprint used to create objects. An object created from a class is known as an *instance* of that class. The class defines the structure and capabilities of those instances, including:

- **Properties (or attributes)**, which store information relevant to the object. For example, a servo might store its current position, speed, or minimum/maximum limits.
- **Methods**, which are functions attached to the class that define what the object can do — such as move, read a sensor, or change configuration.

Classes allow you to organize code logically, create predictable behaviors, and reuse common functionality across many parts of your project. In Synthiam ARC, numerous features — such as servos, sensors, robot movement, and audio control — are exposed as classes, each containing various methods you can call to interact with your robot hardware.

What is a Method?

A `method` is a function that belongs to a class. It defines an action or behavior the class can perform. Methods often operate on the data stored within an instance of that class, which means they can read or update property values to change how the object behaves.

For example, a method in a camera class might capture an image, change a setting such as exposure, or start video tracking. A servo-related method might adjust angle, set a speed, release torque, or retrieve position data.

Methods are central to interacting with robot hardware in ARC. When you call a method, you're sending a structured instruction to the ARC system telling it exactly what you want that component of your robot to do.

Class.Method() Notation

In the Synthiam ARC documentation and examples, you'll frequently encounter expressions such as `Servo.setServoPosition()`. This format is known as **dot notation**. The dot (`.`) acts as a separator that tells the programming language, "This method belongs to this class."

For example:

- `Servo` is the class — it represents the entire concept of a servo motor within ARC.
- `setServoPosition` is the method — it's the specific action you want to perform.

When combined as `Servo.setServoPosition(90)`, you are instructing ARC to execute the `setServoPosition` method on the `Servo` class and pass in a value of `90` as a parameter. In this example, the parameter represents the exact position you want the servo to rotate to.

This class–method relationship exists throughout all ARC scripting features. Any time you see class-style names followed by a dot and a function name, you are calling a method that belongs specifically to that class.

Case Sensitivity

Both JavaScript and Python, which are used in Synthiam ARC, treat uppercase and lowercase letters as completely different characters. This is known as **case sensitivity**. This means that the following names are not interchangeable — they are considered entirely different identifiers:

- `servo`
- `Servo`
- `SERVO`

If the ARC documentation specifies a class called `Servo`, then you **must** write it with that exact casing. Changing even a single letter — for example writing `servo` — will cause an error and prevent your script from running.

This same rule applies to:

- class names
- method names
- variable names
- arguments
- built-in functions

Case sensitivity is one of the most frequent causes of issues for beginner programmers, so always double-check your capitalization when calling ARC classes and methods.

Example in JavaScript

```
class Servo {

// A method defined on the Servo class that sets the servo's position.
static setServoPosition(position) {

```
console.log(`Setting servo position to ${position}`);
```

}
}

// Correct way to call the method using proper capitalization
Servo.setServoPosition(90);

// Incorrect - case sensitivity matters!
// servo.setservoPosition(90); // This will cause an error because both the class and method names are wrong
```

Example in Python

```
class Servo:

```
# A static method that takes a position and prints a message.
@staticmethod
def setServoPosition(position):
    print(f"Setting servo position to {position}")
```

Correct way to call the method with exact case

Servo.setServoPosition(90)

Incorrect - case sensitivity matters!

Servo.setservoposition(90) # This will fail because the method name does not match exactly
```

Whenever you are writing scripts in Synthiam ARC, always refer to the official documentation for the exact class and method names. Following correct capitalization ensures that your scripts run without errors and that ARC can properly communicate with the components of your robot.

Script Robot Skills

The Scripting category in Synthiam ARC empowers users to create custom scripts and behaviors for robots. It includes scripting languages and tools that allow users to define complex and customized robot actions, expanding the flexibility and capabilities of robotic systems.
Here are the manuals for
JavaScript,
Python,
EZ-Script,
Blockly,
and RoboScratch.

Nearly all robot skills can execute a script. However, there are robot skills specific to scripting. These scripting robot skills allow you to run programs or customize features, such as creating a custom movement panel.

[opt:skillcategory:scripting]

Global Variables

JavaScript and Python variables are private to each script engine's namespace. In other words, a variable created in one JavaScript or Python script does not automatically exist in another script. For example, variables declared in the JavaScript or Python for the Camera control are not accessible from the JavaScript or Python for the WiiMote control. If you need to share data between scripts, use ARC's global variable storage.

Access Public/Global Variables

Public variables are stored in ARC's global variable manager, which is shared across all robot skills and compilers. You can access these variables from JavaScript and Python using the `getVar()`, `setVar()`, `setVarObject()`, and `varExists()` commands. Refer to the JavaScript or Python documentation for the exact syntax and usage details.

Commands

- **`setVar( "$variableName", value )`** — Creates or updates a global variable. If the value is an array, a global array variable is created. If the value is a boolean, it is stored as `1` (`true`) or `0` (`false`).
- **`setVar( "$variableName", value, "description" )`** — Same as above, with an optional human-readable description. See the *Variable Description* section below.
- **`setVarObject( "$variableName", value )`** — Stores a deep copy of the supplied object as the global variable. Use this when the value is a complex object that should not be changed by later script activity. The stored variable is an independent clone, not a reference.
- **`setVarObject( "$variableName", value, "description" )`** — Same as `setVarObject()` above, with an optional description.
- **`getVar( "$variableName" )`** — Returns the current value of the global variable. Strings are returned unquoted, numbers are returned as numeric values, and arrays are returned as arrays. You may also index directly into an array, for example: `getVar( "$myArray[2]" )`.
- **`getVar( "$variableName", defaultValue )`** — Returns the variable's value if it exists; otherwise, returns the supplied default value. Use this to avoid errors when a variable may not yet be defined.
- **`varExists( "$variableName" )`** — Returns `true` if the global variable has been defined; otherwise, returns `false`.

Variable Description

The optional third parameter of `setVar()` and `setVarObject()` is a description string. This

is a short, human-readable note that explains what the variable is used for. ARC stores the description alongside the variable in the global variable manager, and it appears in the Variable Watcher and variable dump output. This makes it easier for you and anyone else reading the project to understand the purpose of each variable at a glance.
Notes about the description parameter:

- The description is optional. If omitted, the variable is created or updated without changing any existing description.
- If you call `setVar()` again with a new non-empty description, the stored description is updated. Passing an empty string leaves any existing description unchanged.
- The description does not affect the variable's value or behavior. It is informational only.

Example in JavaScript:

```
setVar( "$BatteryLevel", 87, "Current battery percentage reported by the robot" );
setVar( "$LastUserName", "Sarah", "Name of the most recently recognized user" );
setVarObject( "$LastDetection", detectionObject, "Most recent object detected by the
camera" );
```

Blockly

Blockly generates JavaScript, so variables are private by default. To make a variable public, begin its name with a `$` (dollar sign).

Default Variables

A few global variables are reserved for specific functions within ARC's internal framework. These variables are initialized when ARC starts and are updated when key framework components change, such as movement direction, speed, audio playback, and speech status.

- **`$Direction`** — The current direction the robot has been instructed to move. The movement manager handles this variable. When any robot skill instructs movement, the current movement panel responds by moving the robot in that direction.
- **`$EZBPlayingAudio`** — Set to `true` when the EZ-B is playing audio through the audio stream, such as an MP3 or WAV from a soundboard. When no audio is playing, this variable is `false`.
- **`$IsSpeaking`** — Set to `true` when the `Say()` or `SayEZB()` commands are executed. Otherwise, it is set to `false`.
- **`$NavigationStatus`** — Stores the current navigation status while the Navigation Manager System (NMS) is navigating. This is used by navigation skills such as The Better Navigator for SLAM waypoint navigation.
- **`$SpeakingText`** — Works with `$IsSpeaking` by holding the current text being spoken. When nothing is being spoken, this variable is cleared and set to an empty string (`""`).

ControlCommand()

Robot Skill Controls Overview

The ARC project desktop contains many windows, and each window represents a robot skill control. A robot skill control is a small program that runs inside ARC and is typically created by a partner. Examples include GPT integrations, vision detection tools such as face, emotion, color, and glyph recognition, servo gait and animation controllers, speech recognition, and many more. Each robot skill control adds a specific behavior that expands what your robot can do.

Common robot skill controls include Connection, Camera, and Auto Position. Because each robot skill is a separate program, ARC provides a way for those skills to communicate with one another. That mechanism is the script command **ControlCommand()**. With

ControlCommand(), an event in one control can instruct another robot skill to perform an action. For example, when the Speech Recognition control detects the phrase "Follow My Face," it can tell the Camera control to enable face tracking.

Since ControlCommand() references robot skills by name, you can rename a robot skill when needed. Right-click the robot skill title bar and select Rename, then enter a new name.

Video Example

ARC workspace showing multiple robot skill control windows.

The example above demonstrates detecting speech recognition text and using cognitive services to respond to more complex messages. When building projects, review the manuals for each robot skill you plan to use, since every skill documents the commands and variables it exposes.

Detailed example showing speech detection with cognitive services integration.

In the Code

In the earlier example, the Speech Recognition robot skill used the phrase "Follow My Face" to instruct the Camera control to begin face tracking. The following examples show how that command is implemented in both script and Blockly form.

EZ-Script

EZ-Script implementation used inside the Speech Recognition control.

Blockly

Blockly implementation of the same behavior.

What Commands Are Available?

Each robot skill accepts a specific set of ControlCommand() parameters, and not every skill supports the same commands. ARC queries each robot skill by asking, "What control commands do you accept?" and displays the available commands in several places. The sections below show where you can find the available ControlCommand() options for robot skills within your project.

Cheat Sheet

Displayed while editing a script, the Cheat Sheet shows the available commands for the selected control.
Cheat Sheet inside the script editor listing control commands.

Right-Click Menu

Right-click inside the script editor to view commands that are relevant to the current control.
Right-click context menu in the editor that lists ControlCommand options.

Blockly

Blockly provides a utility block that lists ControlCommand() options for the selected control.
Blockly's Utility category includes a ControlCommand block.

How It Works

ARC uses a Robot Skill Control Manager to track all robot skill controls in the project. When a control is added, the manager assigns it a unique name. You can also edit a control's name on its configuration page. The ControlCommand() function uses the target robot skill's name as its first parameter, and the remaining parameters depend on the skill. Those parameters are documented in the Cheat Sheet, right-click menu, or Blockly block.

ARC ControlCommand framework showing how controls interact through the Robot Skill Control Manager.

ControlCommand() Is Non-Blocking

ControlCommand() does not wait for the destination robot skill to finish executing its action. For example, if your script instructs the Auto Position control to perform the "Wave" action, your script immediately continues to the next line without waiting for the wave to complete:

```
controlCommand("Auto Position", "AutoPositionAction", "Wave");
Audio.SayEZB("I am waving");
```

EZ-Script

EZ-Script example: instruct Auto Position to wave, then speak immediately.

Blockly

Blockly equivalent of the non-blocking command sequence.

Because the Auto Position control runs as a separate program, ControlCommand() only sends a one-way instruction: "Do this." It does not wait for an immediate response or block until the action is complete.

What If I Want to Wait?

Most ControlCommand() calls are non-blocking. A few commands explicitly support waiting, and those commands usually include "Wait" in the parameter name. For example, the Scripting control accepts a *ScriptStartWait* command that waits for the script to finish. These wait-capable commands are uncommon.

If a control does not provide a built-in wait command, you can often monitor variables that the control sets to indicate status. For example, some controls expose a status variable such as `$AutoPositionStatus`, which is 1 while the action is running and 0 when it finishes. Not every control provides a variable like this, but those that do usually list it on their configuration page. You can also view all project variables using the Variable Watcher.

RoboScratch and Blockly include explicit wait commands for common tasks, such as "Wait for Auto Position" or "Wait for Sound." These commands clearly indicate "(Wait)" in their titles. The example below shows how to wait for an Auto Position action to complete before speaking.

```
controlCommand("Auto Position", "AutoPositionAction", "Wave");
```

```
sleep(500);
while (getVar("$AutoPositionStatus"));
Audio.SayEZB("I finished waving");
```

EZ-Script

EZ-Script example that waits for the Auto Position status variable before proceeding.

Blockly

Blockly equivalent that waits for the Auto Position action to complete.

In the example above, the `sleep(500)` call gives the Auto Position control a brief moment to start the action. The loop that checks `getVar("$AutoPositionStatus")` keeps running until the status variable changes. When Auto Position finishes, it sets the variable to 0, the loop exits, and the script speaks the confirmation phrase.

Code Editor

Overview

Scripting is available in ARC using Blockly, JavaScript, Python, and EZ-Script. All scripting editors will support any three programming languages, including Blockly. The software also presents intellisense scripting input, which means the commands are displayed as you type.

Scripting Controls

Within the Add Skills dialog of ARC is a section for scripting skill controls. The skill controls within this category are standalone scripting skill controls. Keep in mind that nearly all skills have scripting capability within their configuration dialog.

Edit Scripts

Within the configuration dialog of nearly all skill controls is the ability to edit scripts to define specific behaviors, such as when to start tracking using the camera. The script editors are input fields with a pencil icon to the right. In this example, we are displaying the configuration dialog for the WiiMote.

Editor Dialog

The editor dialog consists of an input section on the left and reference assistance on the right. The window can be resized or maximized; remember the last size for future use.

Choose Programming Language

The programming language tab will be selected if there is existing code in the editor when it loads.
Alternatively, if you are editing a blank script, there will not be any code, and you can select the tab for the programming language you wish to use.
If there is existing code, changing the programming language will present a confirmation that you want to erase the current code and start over.

Choose Programming Language

Synthiam has developed the ARC software to scale between beginner and advanced users. Our mission is to make robot programming accessible and increase the likelihood of success. We have included multiple programming languages in ARC to be used based on your skill level.

Roboscratch

(Very easy)

Designed for learning the basics of programming, RoboScratch introduces a programming interface exclusive to ARC. With RoboScratch, function blocks are added to the workspace to create a sketch.
There are blocks such as waiting for the camera to see an object, waiting for speech, or executing a movement action.
Link blocks to instruct robots to perform behaviors and complete tasks.

Blockly
(Easy)

Blockly programming provides the user with graphical blocks representing programming functions or subroutines.
Blockly allows users to access advanced technologies creatively without knowing the programming syntax by typing with a keyboard.

Scripting
(Intermediate)

Syntax programming is available in three scripting languages JavaScript, EZ-Script, and Python. All robot skills support any scripting languages to interact with each other for custom behaviors.
Using the powerful ControlCommand messaging system, have events from robot skills send triggers to other robot skills within the project.

Programming
(Advanced)

Compile programs and libraries which interface with the ARC API. Make robot skills to distribute in the robot skill store, or use the existing framework to accelerate robot development.
Access existing robot skills or the ARC API framework with C++, C, C#, VB, and more.

There may be options to add code when editing a robot skill configuration. The code dialog window in ARC provides a tab option to select what language to use for this option. You can use multiple languages throughout a project - you do not need to use the same language.

Code Completion

Intelligent Code Completion (IntelliSense)

As you begin typing the script, the complete intelligent code will present a selection of functions based on your input. You may use the arrow Up/Down keys to select the appropriate command or continue typing. Pressing the ESC key will hide the dialog.

*Note: Remember that when a code snippet is generated by intellisense, the case sensitivity matters. There are naming conventions for programming languages, which is why case sensitive matters. It's also important to note that a capital 'B' is different than a small 'b' because programming languages use their ASCII values.

What is the . (period)?

- classes (which group functions) start with uppercase letters (i.e. **S**ervo)
- functions start with lowercase letter (i.e. **s**leep)

So if you look at the Console class, it has a dropdown with many options...

The "Console" is the class (which groups functions) and "log" is the function. If you press "Console." with a period at the end, you'll see all the functions that live under Console..

What is a Class?

In both JavaScript and Python, a `class` is a blueprint for creating objects. A class encapsulates data for the object (properties) and methods to manipulate that data. It's a way to bundle data and functionality together.

What is a Method?

A `method` is a function defined within a class that operates on the data contained in the class. Methods are used to define the behaviors of an object. The method is invoked on an instance of the class, allowing it to access and modify the properties of that instance.

Class.Method() Notation

In Synthiam ARC's context, when you see a notation like `Servo.setServoPosition()`, it means the `setServoPosition` method is being called on the `Servo` class. This method would typically be used to control a servo's position in a robotic project.
The period (`.`) separates the class name from the method name, indicating that the method belongs to the class.

Case Sensitivity

Both JavaScript and Python are case-sensitive languages. This means that `servo`, `Servo`, and `SERVO` would be considered different identifiers. Therefore, it is crucial to use the exact case as defined in the Synthiam ARC documentation when referring to classes and methods.
Example in JavaScript

```
class Servo {
    static setServoPosition(position) {
        console.log(`Setting servo position to ${position}`);
    }
}

// Correct way to call the method
Servo.setServoPosition(90);

// Incorrect - case sensitivity matters!
// servo.setservoPosition(90); // This will cause an error
```

Example in Python

```
class Servo:
    @staticmethod
    def setServoPosition(position):
        print(f"Setting servo position to {position}")

# Correct way to call the method
Servo.setServoPosition(90)

# Incorrect - case sensitivity matters!
# Servo.setservoposition(90) # This will cause an error
```

Remember to always refer to the Synthiam ARC documentation for the correct usage of classes and methods in your projects.

Shortcut Keys

This guide lists shortcut keys used in the Synthiam ARC script editor to help improve your productivity.

```
table { width: 100%; border-collapse: collapse; }
th, td { text-align: left; padding: 8px; border-bottom: 1px solid #ddd; }
th { background-color: #f4f4f4; }
code { background-color: #f4f4f4; padding: 2px 4px; }
```

Shortcut	Action
`Ctrl + F`	Search/Find within the document (F3 continues last search)
`Ctrl + H`	Replace text within the document
`Ctrl + A`	Select All content in the document
`Ctrl + X`	Cut the selected text or item
`Ctrl + C`	Copy the selected text or item
`Ctrl + V`	Paste the copied or cut text or item
`Ctrl + G`	Go to a specific line number in the document

These shortcuts can significantly speed up your coding or text editing process by reducing the need to navigate through menus or use a mouse for common actions.

Port Summary

The following tab from the Script Manual page is the Port Summary. This page displays how the ports of the EZ-B are being used. The summary is manually entered in the Project Details section. Creating a port summary helps you know what is connected to the EZ-B ports. You will define what is connected to each port, for example, Left Arm Servo or Right Leg Servo. Configure the port summary with either the robot project's Robot Designer or Properties tab. These are located in the My Robot top menu under Project.

Cheat Sheet

The third tab of the editor dialog is the Cheat Sheet. This is a very powerful tab, as it automatically generates code for you based on the project configuration. This tab analyzes the project and determines what functions are available from each control. The ControlCommand() script function is how you programmatically set parameters between ARC Controls. The Cheat Sheet tab will display all available ControlCommand() parameters for all controls in your project. Clicking on a ControlCommand() in the list will be added to your script editor.
Alternatively, you can right-click for a quicker menu list of control commands in the editor.

Console

The fourth tab is a temporary Console for Output while testing your script. The console will display the output of a Print() function or any compiler errors. When you press the Run button to test your script, the output will be displayed in this Console tab.

Variable Picker

The global variable picker in ARC, which supports various programming languages, is convenient for developers and users. It offers a comprehensive list of all variables presently assigned within the compiler environment. This feature is particularly valuable for managing and manipulating data across different sections of your code.
***Note:** *Default global variables are set for internal system components of ARC that can be understood by clicking here.*

EZ-Script

In the context of EZ-Script, variables are globally accessible and are denoted by a $(dollar sign). This global scope ensures that variables can be shared seamlessly among various controls and scripts within the ARC platform. The versatility of EZ-Script variables allows for efficient communication and coordination between different components of your project.

JavaScript & Python

The ARC environment provides dedicated methods for developers working with JavaScript or Python to interact with global Script variables: getVar(), setVar(), and setVarObject(). These methods enable seamless integration and data exchange between all programming environments and robot skills, enhancing the flexibility and interoperability of your project. setVar() is used to store generic variables globally, such as strings, numbers, and single-dimensional arrays. Where setVarObject() is used to store objects globally, such as multidimensional arrays and other JavaScript objects, other robot skills can use getVar() to obtain the variable's value from the global storage in any robot skill.

Variable Picker

Additionally, the variable picker not only displays the names of variables but also includes their current values. This real-time information is invaluable for debugging and monitoring the state of your program. In the case of arrays, the variable picker goes a step further by presenting the values encapsulated within each array. This level of detail aids developers in gaining insights into the structure and content of complex data structures.

A user-friendly feature of the variable picker is its interactive functionality. Clicking on a variable name within the picker instantly adds the corresponding text to your script editor. This streamlines the coding process, reducing the chances of errors and making it easier to reference and work with variables throughout your codebase.

In summary, the variable picker in ARC is a powerful tool for managing variables across different programming languages. It promotes code clarity, facilitates seamless integration between EZ-Script and external languages like JavaScript or Python, and enhances the overall development experience by providing real-time insights into variable values and structures.

Clipboard

Microsoft Windows includes clipboard functions native to all text controls. You can use the keyboard shortcut CTRL-C to copy the text by selecting the text. Later, you can use the keyboard shortcut CTRL-V to paste the text from the clipboard. Additionally, there is also the ability to CUT text, which removes the text from the editor into the clipboard. To CUT text, select the text and press the shortcut CTRL-X. Later, you can use the same pasting shortcut to paste the CUT text (CTRL-V).

CTRL-X (cut)

CTRL-C (copy)

CTRL-V (paste)

*Note: Be sure to hold the CTRL (control) key while pressing the secondary key (X, C or V). Do not let go of the CTRL (control) key until you have pressed the secondary key. Once you have pressed the secondary key, let go of both keys, and the action has been executed.

Block Coding

RoboScratch

Designed for learning the basics of programming, RoboScratch introduces a programming interface exclusive to ARC (Windows PC and Mobile). With RoboScratch, function blocks are added to the workspace to create a sketch. Specific parameters of each block, such as wait for the camera to see an object, wait for speech, or execute a movement action. Link blocks and instruct your robot to perform behaviors and complete tasks.

Show Me RoboScratch

What Is RoboScratch?

RoboScratch teaches linear programming without the complexities of loops and conditions. Even though ARC includes Blockly, we recommend RoboScratch for beginners to teach logical execution of commands. Programming is a similar to a food recipe, in that instructions are to be completed in order. RoboScratch provides a fun and engaging way of identifying how to organize instructions to complete tasks. Similar to Blockly, RoboScratch generates real usable code with friendly comments. The initiative of scratch-like development environments is meant to teach programming and order of instructions. The RoboScratch approach provides the ability to learn how to program quickly by viewing the auto-generated code. Think of RoboScratch as training wheels for programming!

Also, the user interface is a little different than traditional block programming, by placing instructions which are connected with lines demonstrating the execution path. You may recognize this approach similar to creating flow charts and process outlines. As you can see, the RoboScratch approach provides very strong educational value.

The Work Space

The workspace is where you create a sketch. Common functions and methods are displayed as *element blocks*. The element blocks can be added to the work space. Each element block includes attributes. The element blocks are connected with lines, which outlines the "path of execution". Starting from the first element, the program will executed each consecutive element and stop at the last element.

Elements

The elements available in RoboScratch (camera tracking, actions, sound, etc.) are only available when respective controls have been added to the project. For example, if you load a project that does not have a Camera Control, RoboScratch camera related elements will not appear. To add camera related elements, simply add and configure a Camera Control to the project. You can learn more about ARC Controls by clicking HERE.

The Code

The magic happens - pressing the Code tab will display the auto-generated code from the RoboScratch UI element blocks. The generated code also includes friendly and helpful comments which explain each line. Below is the auto-generated code from the above screenshot. This code display is read-only.

Get Started

Step 1

Load the default example project for your robot from either the top menu's File->Open option, or the EZ-Cloud->Open. Remember, the default example project is named after your robot, which will be JD, or Six, Roli, etc.. This was covered in the tutorials while assembling your robot. If your robot requires a servo profile, ensure you have loaded the one which was created during the assembly tutorials.

***Note:** The elements available in RoboScratch (camera tracking, actions, sound, etc.) are only available when respective controls have been added to the project. This means, if you load a project that does not have a Camera Control, any RoboScratch camera related elements will not appear. You can learn more about ARC Controls by clicking HERE.

Step 2

Establish a connection to your robot. This was also covered in the assembly tutorials. Connect to the robot's WiFi network, press the Connect button in ARC connection control, and the Start button on the Camera Control to begin streaming the video.

Step 3
Press the RoboScratch window button to load the RoboScratch designer view.

Step 4
Press the RoboScratch window button to load the RoboScratch designer view. As you can see, the work space will be empty so you can begin right away!

Step 5

We will start by demonstrating how to make the robot move forward for 3 seconds, stop, and speak a phrase.

Step 6
Locate and press the Move element from the side menu. This will add the Move Element to the work space. Select FORWARD from the Move Element's list of movement directions in the drop down.
Optionally, you can hover the cursor over the blue question mark of an element in the work space to read additional details.

Step 7
Locate and add the Sleep element from the side menu. Enter the value of 3000 milliseconds, which is 3 seconds. Computers work very fast in the speed of milliseconds. Also, the term Sleep is used in computer programming, not delay as you may expect. The

two have different meanings, which can be identified by reading the help by hovering your cursor over the blue question mark on the Sleep element.

Step 8
Locate and add the Move element again, and this time select STOP.

Step 9
Now we will make the robot speak by adding the Say And Wait Until Finished element. For fun, add your own custom text into the field to have your robot speak your name!

Step 10
Now press the START button in the top left and the robot will begin executing the command of each element. Notice how the RoboScratch elements will highlight in Yellow as they are being executed.

View The Code
The goal of RoboScratch is to teach you how to program. This is done by converting your RoboScratch elements into real code! The code can be viewed by pressing the Source Code tab.

Finished!

You have now mastered RoboScratch! Start experimenting by creating your own programs by adding elements. Hover your mouse cursor over the blue question marks to find out what an element does.

Blockly

ARC introduces Blockly Programming for beginners. We recommend that early beginners start with RoboScratch before Blockly. Once you have become comfortable with RoboScratch, moving to Blockly is the next step on your programming adventure!

Copy Blocks

You can copy a group of blocks by selecting the outer scope block. An outer scope block is a block that surrounds other blocks, such as an IF condition, REPEAT loop, or FUNCTION. Right-click on the outer surrounding block and select DUPLICATE.

Variables

Blockly has two types of variables: local/private variables and global variables. A global variable will have a dollar sign($) in front of the variable name (i.e. $MyVariable), and a local/private variable does not have a dollar sign($) (i.e. myVariable).

In this example, we first assign a local/private variable followed by a global variable. Notice how the generated syntax differs. The global variable has a ($) dollar sign and sets the variable using the setVar() command. The setVar() command will set the variable into the global storage area.

Global vs Private Variable

A private variable is only available within the script. No other ARC scripts can access a private variable because it only exists within the script's scope. This is useful if your program creates many variables that are not needed by any other scripts.

A global variable is available to any script within the entire ARC project. This means that any other script can read or change the variable contents. This is useful when sharing data across other scripts, such as the robot's state.

How To Use

Step 1

Before using Blockly, load your robot project into ARC. Like RoboScratch, ARC requires an existing project to be loaded and connected to the robot. This gives Blockly commands and actions to start programming with.

Step 2

Load the Blockly Interface. The Blockly UI can be found in two places.
a) The Blockly tab when editing scripts for robot skills.

b) The top File Menu next to RoboScratch and the Virtual Desktops.

Step 3

Begin building a program! Using block programming makes programming more accessible in comparison to Syntax Style Programming. You never worry about spelling mistakes or incorrect syntax in a block programming environment. Block programming focuses on learning the logic of Loops, IF conditions, and Variables.