Blockly API

Overview

Introduction to Blockly

ARC introduces Blockly programming for beginners. If you are new to programming, we recommend starting with
RoboScratch. Once you are comfortable with RoboScratch, Blockly is the next step to learn structured programming concepts such as loops, conditions, functions, and variables while avoiding syntax errors.

Video: Getting started with Blockly in ARC.

Copy Blocks

To duplicate a group of blocks, select the outer scope block that surrounds the group (for example, an IF block, a REPEAT loop, or a FUNCTION block). Right-click on that outer block and choose **Duplicate**. This duplicates the block and all its nested blocks together.

Example: duplicating an outer scope block with nested blocks.

Variables

Blockly in ARC supports two types of variables:

- **Local / Private variables** — These do not use a dollar sign and exist only within the current script (for example, `myVariable`).
- **Global variables** — These use a dollar sign prefix (for example, `$MyVariable`) and are stored in the project's global storage so any script can access them.

In the examples below, the left image shows assigning a local variable; the right image shows assigning a global variable. Notice that global variables are set using the `setVar()` command which stores the value in global storage accessible to other scripts.

Local variable assignment inside a single script.
Global variable assignment using setVar(), stored in the project's global storage.

Global vs Private Variables

A **private (local) variable** is only available within the script where it is defined. Other scripts in the ARC project cannot read or modify it. Use local variables to keep script-specific data isolated and avoid unintended interactions.

A **global variable** is accessible by any script in the ARC project. Global variables are useful for sharing state or data between multiple scripts — for example, the robot's mode, shared sensor values, or flags indicating overall program state.

How to Use Blockly

Step 1 — Load an ARC Project

Before using Blockly, load your robot project into ARC and ensure the robot is connected. Blockly commands and actions operate on the currently loaded project, so the project provides the context and resources (motors, sensors, devices) your program will control.

Step 2 — Open the Blockly Interface

The Blockly UI is available in two places within ARC:

1. Within the script editor: open the **Blockly** tab when editing a robot skill.
2. From the top File menu: select **Blockly** (next to RoboScratch and Virtual Desktops).

Open the Blockly tab while editing a script to start building blocks.
Access Blockly from the top File menu next to RoboScratch and Virtual Desktops.

Step 3 — Build Your Program

Start assembling blocks to create logic using loops, conditions, functions, and variables. Block-based programming removes syntax concerns (such as typos or missing punctuation), allowing you to focus on program flow and problem-solving. As you build, frequently test your program on the robot to validate behavior and debug logic.

Example: a Blockly program using loops and conditions to control robot behavior.

Block Definitions

This manual describes all Blockly blocks available in Synthiam ARC's Blockly visual programming environment.

Use this page as a quick reference when building robot behaviors with Blockly drag-and-drop blocks.

Getting Started

Blockly lets you create programs visually by snapping blocks together. Each category contains a different type of block,
such as logic, loops, movement, camera, servos, audio, variables, and more.

- **Statement blocks** perform actions, such as moving a robot or waiting for speech.
- **Value blocks** return data, such as a number, text, true/false, or a sensor reading.
- **Input sockets** allow you to place one block inside another.
- **Mutator blocks** have a small gear icon that lets you add more options, such as extra conditions or text items.

Quick Navigation

Logic

Logic blocks help your program make decisions based on conditions.

If / Else If / Else
Block type: `controls_if`
Checks a condition and runs the blocks inside the **do** section when the condition is true. You can expand this block using the gear icon to add more **else if** or **else** sections.

Compare
Block type: `logic_compare`
Compares two values and returns either true or false.
Operators: =, !=, <, <=, >, >=

Logic Operation (AND / OR)
Block type: `logic_operation`
Combines two true/false values using **and** or **or**.

- **AND** means both conditions must be true.
- **OR** means either condition can be true.

Not
Block type: `logic_negate`
Reverses a true/false value. If something is true, it becomes false. If it is false, it becomes true.

Boolean
Block type: `logic_boolean`
A simple true or false value block.

Loops

Loop blocks repeat actions until a condition is met, a counter finishes, or forever.

Repeat N Times
Block type: `controls_repeat_ext`
Repeats the enclosed blocks the specified number of times.

Break
Block type: `break`
Immediately exits the current loop.

While / Until
Block type: `controls_whileUntil`
Repeats blocks while a condition is true, or until a condition becomes true.

Repeat For
Block type: `repeat_for`
Counts from one number to another using a variable and a step amount.

Loop Forever
Block type: `loop_forever`
Repeats the blocks forever until a **Break** or **Halt** block stops execution.

Label
Block type: `labelblock`
Creates a named marker in your program that other blocks can jump to.

Goto (Continue Label)

Block type: `gotoblock`
Jumps execution to a label with the matching name.

Audio

Audio blocks let your robot speak, play sounds, and wait for speech recognition results.
Play Audio
Block type: `play_audio`
Plays a sound from the project's soundboards and continues immediately.
Play Audio (Wait)
Block type: `play_audio_wait`
Plays a sound and waits until it finishes before continuing.
Stop Audio
Block type: `audio_stop`
Stops audio currently playing from the EZ-B speaker.
Say EZB / Say EZB (Wait)
Block types: `say`, `say_wait`, `sayWithIndex`, `say_waitWithIndex`
Speaks text through an EZ-B speaker. Some versions continue right away, while the **Wait** versions pause until speech finishes.
Say PC / Say PC (Wait)
Block types: `say_pc`, `say_pc_wait`
Speaks text through the computer's sound device.
Wait For Speech
Block types: `wait_for_speech`, `wait_for_speech_ext`
Waits for one of the expected phrases to be spoken and returns the detected result, or `timeout` if nothing is recognized in time.
Wait For Speech Range
Block type: `wait_for_speech_range`
Waits for a spoken number within a specific range.
Wait For Any Speech
Block type: `wait_for_any_speech`
Waits for any speech input and returns what was heard, or `timeout`.
Wait For Any Number Speech
Block type: `wait_for_any_number_speech`
Waits for any spoken number, including negative and decimal values.

Camera

Camera blocks help your program detect, follow, capture, and react to what the robot sees.
Set Camera Tracking
Block type: `camera_tracking`
Sets the camera tracking mode, such as Face, Object, Motion, Glyph, or color tracking.
Movement Tracking / Servo Tracking
Block types: `movement_tracking`, `servo_tracking`
Turns robot movement tracking or servo tracking on or off so the robot can follow tracked targets.
Wait For Detection Blocks
Block types: `wait_for_face`, `wait_for_color`, `wait_for_glyph`, `wait_for_object`, `wait_for_multi_color`, `wait_for_ycbcr_color`
These blocks pause execution until the selected target is detected by the camera.
Take Photo
Block type: `take_photo`
Captures a photo from the camera and saves it to the My Pictures folder.
Tweet Camera Image
Block type: `tweet_image`
Posts the current camera image to a configured Twitter account with a message.

Camera Control Command
Block type: `camera_command`
Sends a ControlCommand() to the camera device using one of the configured project commands.
Camera Variables

The Camera category also includes pre-built compare blocks for checking things such as:

- Whether the camera is tracking
- Whether the tracking type is Face
- Whether the tracking type is Object
- Whether the tracking type is Color
- Whether the tracking type is MultiColor
- Whether the tracking type is Motion
- Whether the tracking type is Glyph

Functions

Functions let you organize reusable logic into named blocks of behavior.
Call Function (with return)
Block type: `procedures_callreturn`
Calls a function and returns a value back to the current block.
Call Function (no return)
Block type: `procedures_callnoreturn`
Calls a function that performs actions but does not return a value.
Return
Block type: `return`
Ends the current function and optionally returns a value.

Math

Math blocks provide numbers, calculations, rounding, constraints, and random values.
Number
Block type: `math_number`
A number constant block.
Arithmetic
Block type: `math_arithmetic`
Performs addition, subtraction, multiplication, division, or power operations.
Random
Block type: `random`
Returns a random number between a minimum and maximum value.
Advanced Math / Trigonometry / Constants
Block types: `math_single`, `math_trig`, `math_constant`
Provides more advanced math operations such as square root, logarithms, sine, cosine, tangent, pi, and more.
Number Property Check
Block type: `math_number_property`
Checks if a number is even, odd, prime, positive, negative, whole, or divisible by another value.
Change Variable By
Block type: `math_change`
Adds a value to an existing variable.
Round
Block type: `math_round`
Rounds a number normally, up, or down.

Constrain
Block type: `math_constrain`
Keeps a number within a minimum and maximum range.

Movement

Movement blocks control robot driving actions such as forward, reverse, turning, and stopping.
Move Forward / Move Reverse
Block types: `move_forward`, `move_reverse`
Moves the robot continuously using left and right wheel speed values.
Move Forward (Timed) / Move Reverse (Timed)
Block types: `move_forward_time`, `move_reverse_time`
Moves the robot for a specified number of milliseconds.
Turn Right / Turn Left
Block types: `move_right`, `move_left`
Turns the robot left or right using a speed value.
Stop
Block type: `move_stop`
Stops all robot movement immediately.

Servo

Servo blocks control servo movement, speed, acceleration, limits, and Auto Position actions.
Auto Position Blocks
Block types: `autoPositionNoLoop`, `autoPositionNoLoopByActionName`, `autoPositionLoop`, `autoPositionLoopByActionName`, `auto_position_background`
Run Auto Position actions by dropdown or by name, either waiting, looping for a duration, or running in the background.
Move Servo
Block type: `move_servo`
Moves a servo on a selected port to a target position.
Get Servo Position
Block types: `getservo`, `getservorealtime`
Reads the current servo position either from cached data or directly from the hardware in real time.
Set Servo Speed / Acceleration / Velocity
Block types: `servo_speed`, `setAcceleration`, `setVelocity`
Controls how the servo moves and how quickly it gets there.
Set Min / Max Position Limit
Block types: `setMinPositionLimit`, `setMaxPositionLimit`
Prevents a servo from moving outside a safe range.
Release Servo
Block type: `release`
De-energizes the servo so it can move freely.

Text

Text blocks create strings, combine text, print messages, and prompt users for input.
Text String
Block type: `text`
A text constant block.
Text Length
Block type: `text_length`
Returns the number of characters in a string or items in an array.
Print
Block type: `text_print`

Prints a value to the output console.
Join Text / Append Text
Block types: `text_join`, `text_append`
Combines strings together or adds more text to a variable.
Prompt For User Input
Block type: `wait_for_prompt`
Shows a prompt to the user and waits for a response or timeout.

Ports

Port blocks read or control EZ-B digital and analog ports.
Get Digital / Set Digital
Block types: `get_digital`, `set_digital`
Reads or writes a true/false value on a digital port.
Get ADC (8-bit) / Get ADC (12-bit)
Block types: `get_adc`, `get_adc_12`
Reads analog values from ADC ports with standard or higher precision.
PWM Output
Block type: `port_pwm`
Sets a PWM output value on a digital port.

Sensor

Sensor blocks provide readings from attached hardware and the EZ-B controller.
Get Ping
Block type: `get_ping`
Reads distance from an ultrasonic sensor using trigger and echo ports.
Get Voltage
Block type: `get_voltage`
Returns the EZ-B controller voltage.
Get CPU Temp
Block type: `get_cpu_temp`
Returns the EZ-B controller CPU temperature.

Utility

Utility blocks perform general-purpose actions such as waiting, sending commands, running custom code, or stopping the script.
Sleep
Block type: `sleep`
Pauses the script for the specified number of milliseconds.
Control Command
Block type: `control_command`
Sends a ControlCommand() to another skill or control in the project.
JavaScript
Block type: `raw_javascript`
Inserts raw JavaScript directly into the generated code.
Halt
Block type: `halt`
Immediately stops script execution.

Variables

Variables store values that your program can use and change while it runs.
Get Variable
Block type: `variables_get`
Returns the current value stored in a variable.
Set Variable

Block type: `variables_set`
Assigns a value to a variable.
Clear Variable
Block type: `clear_variable`
Clears or resets the selected variable.

Tip: Start simple. Build small block sequences first, test often, and then combine them into larger behaviors.

If something is not working, print variable values to the console and test one section at a time.