Copyright 2026 Synthiam Inc.

# EZ-Script API

## All Commands

**Note:** After adding JavaScript and Python support to Synthiam ARC in 2017, EZ-Script is no longer actively maintained. The EZ-Script compiler and language are slow, outdated, and contain known bugs. Existing EZ-Scripts generally continue to run, but EZ-Script is not recommended for new development. For active support and modern features, use JavaScript or Python in ARC.

## Timing and Basic Flow

### Sleep(milliseconds)

Pauses script execution for the specified number of milliseconds.

```
Sleep(1000)   # Pauses for 1 second
```

### SleepRandom(lowMilliSec, highMilliSec)

Pauses for a random interval between the two specified millisecond values.

```
SleepRandom(1000, 5000)
```

## Servo and PWM Control

### Servo(servoPort, position)

Move a servo to a specified position (1–180).

```
Servo(D14, 25)
```

### SetServoMin(servoPort, position) / SetServoMax(servoPort, position)

Set hard limits for a servo so it never moves below the minimum or above the maximum. Positions are in the 1–180 range.

```
SetServoMin(D14, 40)
SetServoMax(D14, 100)
```

### ServoSpeed(servoPort, speed)

Set the speed for a servo (or PWM) when moving between positions. Speed values range from 0 (fastest) to 10 (slowest).
**Important:** On first use, set an initial Servo() position before calling ServoSpeed(); otherwise the system may assume the current position is 0 and behave unexpectedly. After initialization, set ServoSpeed() before calling Servo().

```
Servo(D14, 20)        # initialize servo position first
ServoSpeed(D14, 2)     # then set speed
```

### ServoSpeedRandom(servoPort, lowSpeed, highSpeed)

Apply a random servo speed between the specified values (0 = fastest, 10 = slowest). Same initialization notes as ServoSpeed().

```
ServoSpeedRandom(D14, 1, 4)
```

### ServoUp(servoPort, count) / ServoDown(servoPort, count)

Increment or decrement the servo's current position by the specified amount. Final position is clamped to 1–180.

```
ServoUp(D14, 1)
ServoDown(D14, 1)
```

### ServoRandom(servoPort, lowPosition, highPosition)

Move the servo to a random position between lowPosition and highPosition (1–180).

```
ServoRandom(D14, 10, 20)
```

### Release(servoPort) / ReleaseAll([boardIndex])

Release a single servo or release all servos so they no longer hold position. BoardIndex is optional and selects the EZ-B board when multiple boards are used.

```
Release(D14)
ReleaseAll()          # release all servos on default board
```

### PWM(digitalPort, speed) / GetPWM(digitalPort) / PWMRandom(digitalPort, lowSpeed, highSpeed)

Set or read PWM (Pulse Width Modulation) on a digital port. PWM simulates an output voltage (scaled between 0–5V). Values are percentages from 0 to 100.

```
PWM(D14, 90)
$x = GetPWM(D14)
PWMRandom(D14, 10, 90)
```

### Move(servoPort, "forward" / "stop" / "reverse")

Set motion for modified servos that support directional movement.

```
Move(D14, "forward")
```

## Digital and Analog I/O

### Set(digitalPort, ON|OFF|TRUE|FALSE) / SetRandom(digitalPort) / ToggleDigital(digitalPort)

Set a digital port state, pick a random on/off state, or toggle the port.

```
Set(D2, OFF)
SetRandom(D2)
ToggleDigital(D2)
```

### Digital_Wait(digitalPort, ON|OFF, [delay MS])

Pause execution until a digital port reaches the requested state. Optional delay controls how often the port is checked (milliseconds).

```
Digital_Wait(D12, ON)
Digital_Wait(D12, ON, 50)
```

### ADC_Wait(adcPort, HIGHER|LOWER|EQUALS, value, [delay MS])

Pause until an ADC port is higher/lower/equals the specified value. Optional delay controls polling interval.

```
ADC_Wait(ADC0, HIGHER, 50)
ADC_Wait(ADC0, HIGHER, 50, 50)
```

### ADC_Wait_Between(adcPort, low, high, [delay MS])

Pause until an ADC port value falls between low and high (inclusive). Optional delay controls

polling interval.

```
ADC_Wait_Between(ADC0, 20, 50)
ADC_Wait_Between(ADC0, 20, 50, 50)
```

### GetADC(port) / GetADC12(port) / GetDigital(port)

Read analog and digital values. GetADC returns 8-bit ADC; GetADC12 returns 12-bit ADC. GetDigital returns 0 or 1 for digital ports.

```
$a = GetADC(ADC0)
$b = GetADC12(ADC0)
$d = GetDigital(D0)
```

## Movement Panel Commands (Wheels, Drones, Movement Controls)

### Forward([speed], [milliSeconds]) / Reverse([speed], [milliSeconds]) / Stop()

Control movement via a configured Movement Panel. Provide optional speed (0–255) and duration in milliseconds.

```
Forward()
Forward(200)
Forward(255, 5000)

Reverse()
Reverse(200)
Reverse(255, 5000)

Stop()
```

### Left([speed], [milliSeconds]) / Right([speed], [milliSeconds])

Turn the robot left or right using the Movement Panel. Speed is 0–255. Optionally specify milliseconds to turn.

```
Left()
Left(200)
Left(200, 5000)

Right()
Right(200)
Right(200, 5000)
```

### Drone and Advanced Movement: Up([ms]) / Down([ms]) / RollRight([ms]) / RollLeft([ms]) / TakeOff() / Land() / DroneEmergency()

Use Movement Panel servo settings for drone actions. Provide optional durations in milliseconds. DroneEmergency() attempts to reset or power down the drone immediately—assign this to a dedicated joystick button for safety.

```
Up()
Up(1000)

Down()
Down(1000)

RollRight()
RollRight(1000)

RollLeft()
RollLeft(1000)
```

```
TakeOff()
Land()

DroneEmergency()
```

### Movement_Wait("forward" | "reverse" | "stop" | "left" | "right")

Pause until a movement command from the Movement Panel is executed. This waits for the movement state, even if the command originates from another running script.

```
Movement_Wait("FORWARD")
```

### WaitForServoMove(servoPort, [timeout MS])

Wait until the specified servo moves to a new position. Unlike Servo_Wait (which waits for a specific condition), this returns when the servo position changes. Optionally specify a timeout in milliseconds.

```
WaitForServoMove(D0)
WaitForServoMove(D0, 1000)
```

### Ping_Wait(triggerPort, echoPort, HIGHER|LOWER|EQUALS, distance)

Pause until the Ping sensor reading is higher/lower/equals the specified distance. Trigger and echo are digital ports.

```
Ping_Wait(D3, D4, HIGHER, 50)
```

## Speech, TTS, and Audio

### EZ-B Speaker (onboard)

Use these commands to speak via the EZ-B v4 speaker. They run in the background unless the Wait variants are used.

```
SayEZB("Hello, I am a robot")            # non-blocking
SayEZB("Speak on EZB #2", 1)             # specify EZ-B index 0-4

SayEZBWait("Hello, I am a robot")        # blocks until finished
SayEZBWait("Speak on EZB #2", 1)
```

SSML variants accept formatted SSML rather than plain text:

```
SaySSMLEZB("...")
SaySSMLEZBWait("...", 1)
```

### StopEZBAudio()

Stop any audio playing on the EZ-B v4 speaker.

```
StopEZBAudio()
```

### PC TTS and Audio

Speak using the PC sound card. Background commands are non-blocking; Wait variants block until speech completes.

```
Say("Hello, I am a robot")            # non-blocking via PC audio
SayWait("Hello, I am a robot")         # blocks until finished

SaySSML("...")
SaySSMLWait("...")

SpeakStop()                            # stops PC TTS audio
```

SpeakRSS and SpeakTwitter read remote content aloud:

```
SpeakRSS("https://rss.cbc.ca/lineup/world.xml")
SpeakRSS("https://rss.cbc.ca/lineup/world.xml", 3)
SpeakRSSDescription("https://rss.cbc.ca/lineup/world.xml", 3)
SpeakTwitter("EZ_Robot")
SpeakTwitter("EZ_Robot", 3)
```

### SpeakVolume(value)

Set the PC TTS synthesizer volume (0–100).

```
SpeakVolume(30)
```

### EZ-B Speaker Volume

Set or get the EZ-B v4 speaker volume. Valid ranges: 0 (quiet) to 100 (loud) and up to 200 for overdrive.

```
SetVolume(50)
SetVolume(100, 1)   # set on EZ-B index 1
$x = GetVolume()
```

### PlayAudio(filename) / StopAudio()

Play audio files (MP3 or WAV) through the default audio device.

```
PlayAudio("c:\\temp\\myAudio.mp3")
StopAudio()
```

### SoundNote(note, lengthMS, [signalType])

Play a musical note on the EZ-B v4 speaker for the specified milliseconds. Optional signal types: Sine, Square, Triangle, Pulse, Sawtooth, WhiteNoise, GaussNoise, DigitalNoise.

```
SoundNote("C2", 1000)
SoundNote("C2", 1000, "Square")
```

### MP3Trigger Shield Commands

Control the MP3 Trigger Shield over a digital serial port. Typical baud rates are 38400.

```
MP3TriggerPlayTrack(d0, 38400, 1)
MP3TriggerPlayTrack(d0, 38400, 1, 3000)    # optional pause disables SpeechRecognition for
given ms
MP3TriggerVolume(d0, 38400, 20)            # 0 = loudest, 255 = quiet
MP3TriggerPlayRandomTrack(d0, 38400, 1, 10)
MP3TriggerNext(d0, 38400)
MP3TriggerPrev(d0, 38400)
MP3TriggerStop(d0, 38400)
```

## I2C, UDP, Serial and UART

### I2CClockSpeed(boardIndex, rate)

Set the I2C clock speed on an EZ-B board. Default is 100000 (100 kHz). Many devices support up to 400000 (400 kHz).

```
I2CClockSpeed(0, 100000)
I2CClockSpeed(0, 400000)
```

### I2CWrite(boardIndex, deviceAddress, data...)

Start I2C, write data (hex, string or decimal), then stop. Device address must be in 0x00 format.

```
I2CWrite(0, 0x09, 0x02, 0x05, 0x06)
```

```
I2CWrite(0, 0x09, 244)
I2CWrite(0, 0x09, "This is text" + $variable)
```

### I2CWriteBinary(boardIndex, deviceAddress, variable)

Send binary data from an array variable to the device over I2C.

```
I2CWriteBinary(0, 0x09, $variable)
```

### I2CRead(boardIndex, 7bitDeviceAddress, bytesToExpect) / I2CReadBinary(..., variable)

Read ASCII or binary bytes from an I2C device. Specify the 7-bit hex address and how many bytes to read.

```
$val = I2CRead(0, 0x5e, 2)
I2CReadBinary(0, 0x5e, 2, $variable)
```

### SendSerial(digitalPort, baudRate, data...)

Send data over a digital port configured as serial (no input buffer). Data may be hex, string or decimal.

```
SendSerial(d0, 9600, 0x00, 0x04, 0x05)
SendSerial(d0, 9600, 244, 200, "a")
SendSerial(d0, 9600, "This is text")
SendSerial(d0, 9600, "Hello " + $name)
```

### SendUDP(hostname, port, data...)

Send UDP packets to a given hostname and port. Data may be hex, string or decimal.

```
SendUDP("192.168.0.1", 21, "HelloWorld")
SendUDP("192.168.0.1", 21, 0x20, 0x21, 0x22, 0x30)
SendUDP("192.168.0.1", 21, 0x20, 0x21, 0x22, "Hello")
```

### UART (EZ-B Peripheral UARTs)

Initialize and use the hardware UARTs on the EZ-B v4. UART receive buffers are 5,000 bytes. Ports: 0 (expansion connector), 1 (D5/D6), 2 (D18/D19).

```
UARTInit(0, 0, 9600)
UARTAvailable(0, 0)                # bytes available to read
UARTRead(0, 0, 10)                 # read ASCII bytes
UARTReadBinary(0, 0, 10, $variable)# read binary bytes into an array
UARTReadAvailable(0, 0)            # read all available ASCII bytes
UARTWrite(0, 0, "hello world")
UARTWriteBinary(0, 0, $variable)
```

See the UART Ports section below for TX/RX pin mappings.

### PC Serial (ComOpen / ComClose / ComWrite / ComRead...)

Open and interact with local PC serial ports. ComOpen begins buffering incoming data (128 KB buffer).

```
ComOpen("com1", 9600)
ComAvailable("com1")
$line = ComReadLine("com1")
ComWrite("com1", "this is data")
ComClose("com1")
```

ComReadBinary / ComWriteBinary allow binary data transfer via arrays.

## Files, Web and External Commands

### HTTPGet(url)

Perform an HTTP GET and return the response contents.

```
HTTPGet("https://192.168.0.10/decoder_control.cgi?command=35&onestep=5&user=admin&pwd=admin")
$temp = HTTPGet("https://192.168.0.15/GetTemperature.cgi")
```

### Exec(exeOrBat, [parameters]) / Browser(url)

Execute a local program or open the default browser to a URL.

```
Exec("C:\\Windows\\notepad.exe")
Exec("C:\\Windows\\notepad.exe", "C:\\MyFile.txt")
Browser("https://www.google.com")
```

### File operations

Read, write, append, delete and inspect files on the computer.

```
FileDelete("c:\\temp\\mylog.txt")
FileWrite("c:\\temp\\mylog.txt", "MyVariable: " + $x)       # appends text (no newline)
FileWriteLine("c:\\temp\\mylog.txt", "Servo Position: GetServo(d2)")  # append newline
FileReadClose("c:\\temp\\mylog.txt")    # close after reading
FileReadReset("c:\\temp\\mylog.txt")    # reset read position
$fileExists = FileExists("c:\\temp\\mylog.txt")
$fileEnd = FileReadEnd("c:\\temp\\mylog.txt")
$char = FileReadChar("c:\\temp\\mylog.txt")
$line = FileReadLine("c:\\temp\\mylog.txt")
$contents = FileReadAll("c:\\temp\\mylog.txt")
$randomLine = FileReadLineRandom("c:\\temp\\mylog.txt")
```

## Control Commands and Window Interaction

### ControlCommand(windowName, ControlCommandParameter, [values...]) — alias: CC(...)

Send commands to named controls in your project. Use the script editor's "Cheat Sheet" tab to view each control's available parameters and expected values.

```
ControlCommand("ADCGraph", pauseOn)
ControlCommand("SoundBoard", Track_3)
ControlCommand("ScriptManager", ScriptStart, "MyScript")
ControlCommand("SpeechRecognition", PauseMS, 3000)

# shorthand alias
cc("Auto Position", AutoPositionAction, "Action Name")
cc("Auto Position", AutoPositionFrame, "Frame Name")
cc("Auto Position", AutoPositionFrame, "Frame Name", 50, 3)
```

### GetControlValue(windowName, ControlCommandValues)

Retrieve a value from a named control. See each control's Cheat Sheet for valid parameters.

```
$x = GetControlValue("ADC Graph", "pause")
```

## Scripting, Flow Control and Debug

### Comments, Labels and Goto

Standard constructs for controlling script flow.

```
# single-line comment
:My_Label
Goto(My_Label)
Return()    # returns from a Goto()
```

### If / ElseIf / Else / EndIf

Supports comparisons (=, <, >, <=, >=, !=) and logical operators AND, OR. Close

condition blocks with EndIf.

```
If(GetDigital(D0) = 1)
  Print("One")
EndIf

If ($Day = 2 AND $Hour = 3)
  Print("Hello")
EndIf

If (GetServo(D5) > 20 OR ($x >= 3 AND $y < 2))
  Print("Yup!")
EndIf
```

### Repeat loops

REPEAT/ENDREPEAT is similar to FOR; REPEATUNTIL and REPEATWHILE provide condition-based loops.

```
REPEAT($x, 0, 5, 1)
  Print("x=" + $x)
ENDREPEAT

REPEATUNTIL($second = 30)
  Print("Second=" + $second)
  Sleep(500)
ENDREPEAT

REPEATWHILE($second <= 50)
  Print("Second=" + $second)
  Sleep(500)
ENDREPEAT
```

### Wait and synchronization

Various wait functions pause script execution until conditions change, with optional timeouts.

```
WaitForChange(GetServo(D0), 1000)
WaitFor($value1 = $value2, 1000)
WaitUntilTime(17, 30)   # blocks until 17:30 (24-hour)
```

### Halt(), Pause() and Print()

Halt exits the current script. Pause stops execution until another script resumes the control or until the user stops it. Print sends output to the debug console.

```
Halt()
Pause()
Print("This is some text")
Print("Today is " + $Day)
```

### Exec, LoadProject and CheckForUpdate

Execute external programs, load projects (replaces current project), and check for ARC updates.

```
Exec("C:\\Windows\\notepad.exe", "C:\\MyFile.txt")
LoadProject("MyTest.ezb")      # replaces the current project; subsequent commands are not
executed
$x = CheckForUpdate()
```

## Variables, Arrays and String Utilities

### Variable basics

Variables are global across ARC. Types are dynamic and determined by assignment.

```
$str = "This is a string"
$number = 6
$dec = 3.14
$byte = 0x52
$bool = true
$result = ($x > $y)
$number++    # increment
$number--    # decrement
$x = 123 << 1    # shift left
$x = 123 >> 1    # shift right
```

### Arrays

Create and manipulate arrays.

```
DefineArray($myArray, 10)
DefineArray($myArray, 10, 2)   # default fill
AppendArray($myArray, 10)
FillArray($myArray, 88)
$x = GetArraySize("$myArray")
Sort($myArray, Ascending)
```

### String and byte functions

Common string and byte utilities.

```
$len = Length("This string is 33 characters long")
$char = GetCharAt("Hello World", 2)        # zero-based index
$byte = GetByteAt("Hello World", 2)
$value = GetByte("H")
$value = GetAsByte("H")
$hex = ToHex(56)
```

Split and substring:

```
$contents = Split("One, Two, Three", ",", 1)
$value = SubString("Cat In The Hat", 4, 2)
```

### Counting, searching and conversion

```
$apples = Count("apple apple apple", "apple")
$found = Contains("Cat In The Hat", "Cat")     # case-insensitive
$index = IndexOf("Cat In The Hat", "In")       # index of first occurrence
$numeric = IsNumeric($x)
$string = ToString($hexData)
```

### Binary and bit helpers

```
$val = SetBits(1,0,0,0,0,0,0,0)
$str = ToBinaryString(254)
$bit = GetBit(255, 1)
```

### Clear and manage variables

```
ClearVariables()
ClearVariable("$MyVariable")
DumpVariables()   # prints all variables and values
```

## Math, Random and Utility Functions

### Math and scientific functions

Common math operations supported: Sin, Cos, Tan, ASin, ACos, ATan, Sinh, Cosh, Tanh, Abs, Sqrt, Ceil, Floor, Exp, Log10, Log, Max, Min, Round, Power, E, Pi, Now, Today.

```
$x = Sin(27)
$y = ASin(27)
$root = Sqrt(9)
$pow = Power(2, 4)
$rounded = Round(9.3848291, 1)
```

### Mapping and random

Scale values between ranges and generate random numbers.

```
$mapped = Map(0.5, -1, 1, 0, 180)     # maps 0.5 from -1..1 into 0..180
$x = GetRandom(10, 50)
$y = GetRandomUnique(10, 50)
```

### Min, Max

```
$lowest = Min(3, 5)
$highest = Max(3, 5)
```

## DateTime, Casting and Reserved Variables

### DateTime functions

Functions available to manipulate and format dates and times.

```
MinDate(date1, date2)
MaxDate(date1, date2)
MonthName(monthNumber)
AddDays(date, days)
AddMonths(date, months)
AddYears(date, years)
AddHours(date, hours)
AddMinutes(date, minutes)
AddSeconds(date, seconds)
TotalHours(date)
TotalMinutes(date)
TotalSeconds(date)
FmtTimeSpan(value, format)
FmtNum(value, format)
FmtDate(value, format)
```

See the referenced MSDN links in the full documentation for formatting options.

### Casting functions

```
CDBL()      # To Double
CInt()      # To Integer
CLong()     # To Long
CUint()     # To Unsigned Integer
CULong()    # To Unsigned Long
CDateTime()
```

### Reserved variables

These read-only variables are provided by the environment:

```
$direction  $date  $month  $year  $day  $dayName
$hour  $minute  $second  $monthName  $time  $pi
```

## Ports, Boards and Hardware References

### Multiple EZ-B Boards

ARC supports multiple EZ-B boards. Prefix ports with the board index to address a specific board (for example: Servo(2.d0, 8) moves D0 on EZ-B board #2). If no board index is specified, board 0 is assumed. When multiple boards are used, board 0 handles Movement

Panels.

### ADC Ports

Analog inputs labeled A0–A7 on the EZ-B read 0–5V:

```
ADC0, ADC1, ADC2, ADC3, ADC4, ADC5, ADC6, ADC7
```

### Virtual Servo Ports

Virtual servos are logical ports used by controls and movement panels — they do not control physical hardware directly.

```
V0 .. V19
```

### Servo / Digital Ports

Ports that can control servos, PWM, digital outputs, and provide digital input readings:

```
D0 .. D23
```

### UART Ports (EZ-B v4)

Three UARTs are available. Use UARTInit(), UARTRead(), UARTWrite() and UARTAvailable() to interact. Each UART has a 5,000-byte receive buffer.

```
UART0 TX: Expansion Connector
UART0 RX: Expansion Connector
UART1 TX: D5
UART1 RX: D6
UART2 TX: D18
UART2 RX: D19
```

### Digital Port Serial Baud Rates

When using SendSerial() on digital ports, these are standard supported baud rates:

```
300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200
```

## ControlCommand Values & Cheat Sheet

Use GetControlValue() and ControlCommand() to get and set control state. The script editor's "Cheat Sheet" shows all available parameters for each control in your project.
**Example:**

```
ControlCommand("ADCGraph", pauseOn)
$x = GetControlValue("ADC Graph","pause")
```

## Sound Notes Reference (EZ-B v4)

The following note names can be used with SoundNote():

```
C1, Db1, D1, Eb1, E1, F1, Gb1, G1, Ab1, A1, Bb1, B1,
C2, Db2, D2, Eb2, E2, F2, Gb2, G2, Ab2, A2, Bb2, C3, Db3, D3, Eb3, E3, F3, Gb3, G3, Ab3, A3,
Bb3
```

## EZ-Script Bugs

**Note:** After integrating JavaScript and Python into Synthiam ARC in 2017, EZ-Script is no longer being actively maintained. The EZ-Script compiler and language are slow, outdated, and contain several known bugs. Existing EZ-Scripts generally continue to run, but EZ-Script is not recommended for new development. For active support and modern features, use JavaScript or Python in ARC.

# Trailing backslash in string

If the last character of a string literal is a backslash (\), EZ-Script will produce a parser error. This most commonly occurs when you concatenate variables to create file paths: the final backslash is interpreted as escaping the closing quote, which leaves the string unterminated.

## Why this happens

The parser treats the trailing backslash as an escape character for the quote that marks the end of the string. As a result, the string is not closed correctly and an error is thrown.

## Example that produces an error

```
# This will produce an error
$x = "C:\temp\"
$y = $x + "asdf"
print($y)
```

## Workaround and correct approach

To avoid the error, do not end a literal with a backslash. Place the backslash at the start of the next literal or add it as its own separate literal when concatenating.

```
# This is successful
$x = "C:\temp"
$y = $x + "\asdf"
print($y)
```

## Recommendations

- Avoid leaving a trailing backslash in a string literal; include the separator at the start of the appended string or as a separate literal.
- When modifying or testing existing EZ-Scripts that build paths, check for trailing backslashes that could trigger this bug.
- For new scripts, prefer JavaScript or Python in ARC. They are actively maintained and have well-defined escaping and string behavior.