AI Assistant Agent

Overview

The AI Script Assistant is an in-app coding partner built directly into the ARC script editor. Instead of switching to a browser, copying code from a chatbot, and renaming variables to match your project, you ask the assistant a question from inside ARC and get back a script that already uses the names you defined.
It generates JavaScript or Python (you pick), respects the ARC scripting API, and remembers the conversation so each follow-up builds on the last reply.

How It Works

1

Ask in Plain Language

Open the assistant from any script editor and type your request — "sweep the servo from 1 to 180," "fix this script, it hangs after the second loop," or "explain what this code does."
2

ARC Sends Project Context

Your prompt is bundled with live project state — globals, EZB ports, robot parameters, controlCommand() targets, and the current script — so the AI has ground truth instead of guesses.
3

Paste, Run, Iterate

The reply comes back in a fenced code block, ready to drop into your script. Don't like it? Ask for changes — the conversation continues until the code is exactly what you want.

Why It's Different

Generic AI chatbots invent function names, hallucinate APIs, and force you to find-and-replace placeholder variables before the code will run. The ARC AI Script Assistant ships with the full ARC scripting reference, plus your project's authoritative state, on every turn. The code it returns uses real APIs and real names that already exist in your project.

What You Get

Project-aware code Scripts use the global variables, port names, and skill commands that already exist in your project.
JavaScript or Python The assistant targets only the language you picked — no mixed-dialect output, no surprise EZ-Script.
Fix, refactor, explain Paste broken code and ask for a fix. Paste working code and ask for a simpler version or an added feature.
Multi-turn conversations "Make it slower." "Now add a sensor check." Each follow-up builds on the previous answer.
Full conversation history Every prompt, reply, attached script, token count, and credit cost is saved and searchable in your account.
Transparent credit billing Pay only for tokens you actually use. Failed requests are free. Per-turn cost shown in your history.

Who Benefits

- **Beginners** — describe a behavior in English and get a working script without learning the full scripting API first.
- **Experienced builders** — skip the boilerplate. Generate a sweep, a sensor loop, or a state machine in seconds, then refine it.

• **Educators** — students learn by asking the assistant to *explain* what code does, not just by copying it.
• **Project owners** — refactor old scripts, port between JavaScript and Python, or document existing code on demand.

Ready to try it?

The AI Script Assistant ships with Synthiam ARC. Download ARC, open any script, and ask the assistant for the code you want.
Get ARC →
View AI Credits

How To Use

Why this matters: The AI does not see your wiring, your servo limits, or what your skills are for. It sees the names and descriptions you write. A port called `d2` with no description is just `d2`. A port called `d2` described as *"left shoulder servo, range 30–150"* becomes a precise, hard constraint the AI will respect.

Project Configuration

1

Describe Your Global Variables

When you call `setVar()`, pass a third argument that explains what the variable means. The assistant reads this description as authoritative meaning. Without it, the AI is left guessing what `$mode` or `$batt` represents.
// Weak — AI has no idea what this is for setVar("$mode", "idle"); // Strong — AI now knows the purpose AND the valid values setVar("$mode", "idle", "Current robot behavior mode. One of: idle, patrol, follow, sleep."); setVar("$batt", 12.4, "Current battery voltage in volts, sampled every 5 seconds.");
2

Fill Out the AI Description on Every Script

Every script in ARC has an **AI Description** field in its configuration screen. Use it to write one or two plain sentences describing what the script does and when it runs. This is what the AI sees when it considers calling, modifying, or referencing that script.
× Weak: (empty) ✓ Strong: "Runs when the bumper sensor triggers. Stops all motors, backs up 30cm, then turns 90 degrees right."
3

Describe Auto Position Actions

Every Auto Position action has a description field. Fill it in. The AI uses these descriptions to choose the right action when a prompt asks for a pose or motion — without them, it can't tell *Wave* from *Salute* from *Point*.
× Weak: Action1 ✓ Strong: Wave — "Raises right arm to shoulder height and waves the hand left-right twice. Returns to home pose. Takes ~3 seconds."
4

Describe EZ-Bits in the Robot Designer

If your robot is built from EZ-Bits, open each bit in the Robot Designer and describe what its servos do, where they're located, and what ports they use. The EZ-Robot JD Humanoid template is a great reference — you'll see entries like *"D4 — left arm elbow"*, *"D5 — left arm wrist rotate."*
✓ Good descriptions look like: D0 — head pan (left/right), center=90, range 20–160 D1 — head tilt (up/down), center=90, range 60–120 D4 — left arm elbow, straight=90, fully bent=30 D12 — right gripper, open=70, closed=145
5

Use the Port Descriptions Tab (No EZ-Bits Needed)

Not using EZ-Bits? Open **Project → Properties → Port Descriptions**. Add each port you use and write a description of what it does. The AI reads this list the same way it reads EZ-Bit descriptions — it's the canonical "what is plugged into what" reference.

✓ Example port descriptions: D0 — pan servo, scanning lidar mount, range 0–180 D1 — tilt servo, scanning lidar mount, range 30–120 D8 — relay for headlight LEDs ADC0 — battery voltage divider, multiply by 3.2 for volts v0 — left drive motor, −100=full reverse, 100=full forward v1 — right drive motor, −100=full reverse, 100=full forward

6

Rename Robot Skills to Match What They Do

Robot skills are addressed by name through `controlCommand()`. Generic names ("Servo Movement Panel," "Soundboard 1") tell the AI nothing — descriptive names tell it everything. **Right-click any skill's title bar and choose *Rename***, then give it a name that describes its purpose.

× Generic: "Servo Movement Panel" → controlCommand("Servo Movement Panel", "Run") ✓ Descriptive: "HeadServo" → controlCommand("HeadServo", "Run") × Generic: "Soundboard 1" ✓ Descriptive: "GreetingSounds"

Prompting Tips

Project setup gets the AI the facts. Good prompting gets the AI the *intent*. The clearer your request, the closer the first reply lands to what you actually wanted.

1. Be specific about what the script should do

State the action, the inputs, the outputs, and any timing. "Move the arm" is ambiguous. "Sweep the left elbow from 30 to 150 degrees over 2 seconds, then back" is buildable.

× Weak: "make the robot wave" ✓ Strong: "Use the Auto Position 'Wave' action, then return to home pose. Speak 'hello' through the SayEZB skill while waving."

2. Name the skills, ports, or variables you want used

If you already know which skill or port should do the job, say so. The AI will use exactly what you named instead of guessing.

✓ Strong: "Read ADC0, convert to volts using the formula in the port description, and store it in $batt every 10 seconds."

3. State the trigger and the stop condition

Tell the AI when the script starts, when it stops, and what should happen in between. Loops without exit conditions are the #1 source of bad generated scripts.

× Weak: "patrol the room" ✓ Strong: "Drive forward at speed 40 until the front sonar reads under 30cm, then stop and turn right 90 degrees. Repeat until $mode is no longer 'patrol'."

4. Ask for a fix on the existing script — don't start from scratch

If you have working code that's *almost* right, leave it in the editor and ask for the specific change. The assistant gets the current script as context — it will modify what you have instead of rewriting it.

✓ Strong: "This script works but the head jitters at the end. Slow the last servo move down to take 1 second."

5. Ask for one thing at a time

Long compound requests confuse the AI and produce monolithic scripts that are hard to debug. Build behavior in small turns — get a working sweep, then ask to add a sensor check, then ask to wrap it in a loop. Each turn produces tested, focused code.

6. Use follow-up turns to iterate

The assistant remembers the conversation. After the first reply, just say "make it slower," "add a check for the bumper," or "use Python instead." You don't need to restate the whole problem.

7. Ask the assistant to explain code you didn't write

Inherited a script from a community project? Paste it in and ask "explain what this script does, line by line." The assistant will walk through it using your project's real variable and port descriptions.

8. Pick the right model for the job

Use a small, fast model for quick fixes and one-line changes. Use a larger reasoning model when you need multi-step logic, careful constraint handling, or a refactor across a long script. You only pay for the tokens you use — matching model to task saves credits.

Quick Checklist

- Every global variable has a description in its `setVar()` call
- Every script's AI Description field is filled in
- Every Auto Position action has a description
- EZ-Bits or Port Descriptions list every port you use, with purpose and range
- Every robot skill is renamed to describe what it does
- You write specific prompts that name the inputs, outputs, and stop conditions
- You iterate one change at a time in the same conversation

Project Configured. Time to Build.

Open any script in ARC, click the AI Script Assistant button, and describe what you want. The work you just did makes every reply better.
Get ARC →
View AI Credits